

MusiDB

A personalized search engine for music

Ruud Stegers, Peter Fekkes, Heiner Stuckenschmidt*

Vrije Universiteit Amsterdam, De Boelelaan 1081a, The Netherlands

Received 14 August 2005; accepted 20 September 2005

Abstract

The increasing use of structured information on the web demands new ways of searching and integrating data from different sources. In this paper, we focus on the use of unique representations of data objects in terms of public repositories (in this case MusicBrainz) and the use of recommendation mechanisms as a basis for supporting information access. We have implemented a prototypical system with the corresponding functionality in the area of digital music. We discuss the challenges of providing integrated access to structured web resources and the solutions adopted in the MusiDB system.

© 2005 Elsevier B.V. All rights reserved.

Keywords: MusiDB; Search engine for music; MusicBrainz; Personalisation; Amazon.com

1. Introduction

Finding information on the World Wide Web is more and more becoming a problem. Besides the sheer amount of information available, the multitude of potential sources and their inherent distribution is hampering the access to information. For the case of textual information this problem is addressed in terms of efficient search engines like Google that are able to index and retrieve large amounts of information based on keywords. Although the increasing use of database-generated information on the web comes with new possibilities – the richer structures offer the possibility to answer more complex queries in a more accurate way – when trying to combine structured data from different sources, it is necessary to handle differences in the structures used by the individual sources. Also, in order to be able to retrieve relevant information, the user has to know about the way content is organized in the different sources.

So-called semantic portals [10] have been proposed as a solution to the problem of accessing structured web data from different sources in a uniform way. Semantic portals normally use ontologies as a neutral representation of the structure of

information contents. The information from different sources is linked to this neutral representation creating a shared view. Functionality for searching and accessing data can be defined independent of the individual sources using the shared ontology. Recommendation techniques can be used as a way to provide the user with relevant data from the sources available [9]. The idea of recommendation is to actively recommend relevant content based on knowledge about the user and his or her preferences. Many of these techniques can be applied without having to access the actual content and without knowledge of the organization of information in the sources. Collaborative Filtering techniques, for example, are solely based on information about how users rate the relevance of certain pieces of information [5]. This means that information from a source that the user has never seen can be identified to be relevant and recommended to the user.

Combining a recommendation system with a semantic portal may be beneficial for two reasons.

- The use of a portal for making multiple information sources accessible at one-point supports the application of recommendation techniques, because working at the portal level provides a broader basis for recommendations both in terms of users and data.
- The use of recommendation techniques support the access to information in a semantic portal because recommendations

DOI of original article: [10.1016/j.websem.2005.05.007](https://doi.org/10.1016/j.websem.2005.05.007).

* Corresponding author.

E-mail address: heiner@cs.vu.nl (H. Stuckenschmidt).

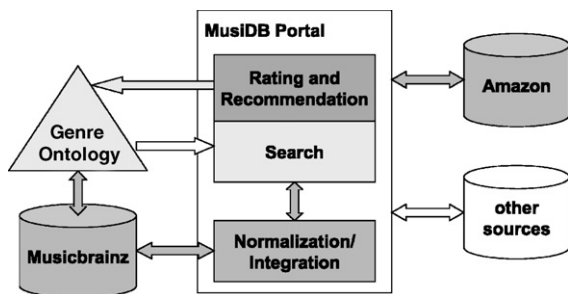


Fig. 1. Architecture of the MusiDB System.

can provide links between content in different sources based on user preferences that can be used for integration purposes.

In this paper, we describe a partial implementation of a semantic portal that combines access to multiple sources with the use of recommendation techniques (see Fig. 1). In particular, all parts of the architecture shown in grey have been implemented. We distinguish between parts of the system that are stable (dark grey) and experimental parts (light grey) that illustrate the underlying idea, but do not provide the complete functionality. Parts of the architecture shown in white are not implemented yet. The system, called MusiDB, is meant as an exploration of the possibilities that come with the application of these techniques.

We chose digital music as a domain because this area is rapidly gaining importance: A significant number of online shops and auctions offer music either in the classical form of compact discs that can be ordered online or in terms of music files that can be downloaded individually. In addition, online music databases and community web sites have emerged that provide information about a wide range of artists, albums and individual songs (e.g. MusicBrainz or mp3.com). Combining these sources into a single portal would provide great advantages for the user. The MusiDB application described in this paper uses the MusicBrainz RDF database [11] as a basis for accessing different providers of digital music. MusiDB implements a simple shared ontology to represent information on albums. With its large amount of information (247,963 albums of 145,997 artists) MusicBrainz can be assumed to cover the content of many providers of digital music. Therefore, the search and recommendation functionality of the system uses the information from MusicBrainz as the primary representation to find relations between artists, albums and songs to expand incomplete user queries. The system then links content from different sources to the instances returned by MusicBrainz. In the current implementation of the MusiDB system, we show how external sources, in our case the Amazon web service, can be linked with MusicBrainz information in order to respond to user queries for a particular song with a list of available albums, their content and price. We further show that recommender functionality can easily be added to such a system. In an experimental addition to the recommender system, we implemented a functionality that automatically assigns artists and albums from MusicBrainz to an ontology of musical genres based on user ratings. This functionality has the potential to be used for topic based search and recommendation.

When building the system, several challenges were encountered that are characteristic for the development on the semantic web and which will be discussed in the remainder of this article:

- The first challenge is to deal with incomplete and inaccurate data. The MusicBrainz repository is maintained by its users. Since the data is not screened and checked before it is added to the repository, inconsistencies will occur. This cannot be prevented and has to be dealt with when trying to match an album with the Amazon data.
- The second challenge is to predict how users would rate certain items based on the information available. MusiDB bases its prediction on other users that have similar preferences for music styles and other albums.
- The last challenge, although less interesting from the semantic point of view, is how to effectively implement the application.

The structure of the paper is as follows. The article starts with a description of data sources and the ontology used as a common view. Both the MusicBrainz repository and the Amazon web services will be introduced, followed by a discussion on how these two are integrated. The next section will then describe the process of predicting ratings and providing recommendations. The association of music styles with artists and albums will also be discussed. After that, the internal structure of the application will be revealed. Both the way the web browser is used to run the application and the underlying business logic are discussed. The paper concludes with a brief discussion on possible enhancements.

A screen video of the application can be found in [14]. It shows how the application is used and demonstrates how the features mentioned in the paper add up to a complete application. The system can be used online at [17]. The complete source code is also available at [15], although at the moment without any support.

2. Data sources

The two data sources used are the MusicBrainz repository [11], and the Amazon database [13]. In order to effectively combine these, a simple data structure has been designed which is used to uniformly represent the data. Fig. 2 shows the structure.

All data is collected and combined real-time based on the user query. Therefore, the top-node has been called 'Result'. *Result* is associated with a list of *Artists*. Each *Artist*-node contains the name of the artist, a unique ID, a rating and a list of relevant *Tracks* by that artist. *Track* contains the title plus a list of the *Albums* on which it can be found. The *Album* then contains its own unique ID, rating, the track number and the duration of this track on the specific album as well as optional images, links and pricing information.

The primary source of data is the MusicBrainz repository. Where possible, the data is augmented with information from external sources (in particular, Amazon). In the next section, the data sources are discussed, followed by a discussion on how to integrate these different sources.


```

<?xml version="1.0" encoding="UTF-8" ?>
- <ProductInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://xml.amazon.com/schemas3/dev-heavy.xsd">
- <Request>
+ <Args>
</Request>
<TotalResults>491</TotalResults>
<TotalPages>50</TotalPages>
- <Details url="http://www.amazon.co.uk/exec/obidos/ASIN/1594200351/myWebsite-22?dev-
  t=D3DKQYCJXVOIQW%26camp=2025%26link_code=xm2">
  <Asin>1594200351</Asin>
  <ProductName>The Last Season: A Team in Search of Its Soul</ProductName>
  <Catalog>Book</Catalog>
- <Authors>
  <Author>Phil Jackson</Author>
  <Author>Michael Arkush</Author>
</Authors>
<ReleaseDate>21 October, 2004</ReleaseDate>
<Manufacturer>Penguin Press</Manufacturer>
<ImageUrlSmall>http://images-eu.amazon.com/images/P/1594200351.02.THUMBZZZ.jpg</ImageUrlSmall>
<ImageUrlMedium>http://images-eu.amazon.com/images/P/1594200351.02.MZZZZZZZ.jpg</ImageUrlMedium>
<ImageUrlLarge>http://images-eu.amazon.com/images/P/1594200351.02.LZZZZZZZ.jpg</ImageUrlLarge>
<ListPrice>£13.35</ListPrice>
<OurPrice>£12.01</OurPrice>
<Media>Hardcover</Media>
<Isbn>1594200351</Isbn>
<Availability>Usually dispatched within 24 hours</Availability>
- <SimilarProducts>
  <Product>0743254260</Product>

```

Fig. 5. Parts of Amazon search results for the term ‘Jackson’.

takes an artist and track title as input, and gives a list of all matching albums as output. MusicBrainz allows for partially specified names. To indicate the reliability of a result, a ‘relevance’ value is added to each result. To be able to uniquely identify all albums and artists in the database, MusicBrainz assigns all items in the database a unique identifier. MusiDB uses the same ID to store ratings in the database.

2.2. Amazon.com

Amazon is a web-shop selling – amongst others – music CDs. To make it possible for partners to provide information about items in the Amazon web-shop, an XML based developer interface is provided. The application provides a search query and Amazon returns an XML document containing information about matching items, grouped 10 per page. A development kit is available with code samples [13]. Amazon uses the Amazon System Identification Number (ASIN) number for unique identification of CDs.

Fig. 5 shows a piece of the result of a query.

3. Integration of data sources

Since we use two different data sources, both with a different unique key for records and both with their own structure, some other kind of integration mechanism is required. In a perfect world, an album could be uniquely identified by its name. Unfortunately, different versions of the same album are released, sometimes with different tracks. This makes it impossible to use only the album title as matching criterion. To make sure that albums referred to in MusiDB at least contain the requested song, an additional check is performed on the track title.

Even when two sources describe the same album, small differences in the stored titles also cause problems. Therefore, the

titles are simplified before comparison. The following types of frequently occurring mismatches have been identified:

- Many false negatives were caused by additions to the title. These include indications of the disc number (e.g. ‘Disc 2’) and information on the recording (e.g. ‘live’). Many of these are written between round brackets. To eliminate this category of false negatives, all portions between round brackets are removed from both the track and the album titles.
- Differences in the use of commas, colons and other symbols are also big cause of mismatches. All symbols (including spaces) are removed from the titles.
- The use of capital letters is another cause of mismatch, which is solved by converting the titles to lowercase.
- In the album titles ‘a’ and ‘the’ are removed since they are another source of error.

The search starts with a user entering an artist name and track title. MusiDB forwards this information to MusicBrainz to obtain a complete list of all the matching albums. MusicBrainz supports an approximate match, and a relevance value is added (see Fig. 3). Tests show that using only the matches with a relevance of more than 60% gives a satisfying result. Of the selected results, the artist name, album title, track number, track title and duration are stored in memory.

For every artist, Amazon is queried to provide a list of all albums by that artist. Each Amazon result is tested against all stored results of that artist. The above-mentioned simplifications of the titles are applied first. Then the track titles are compared. This is to guarantee in an early stage that the right track is on the CD. To allow again for additions to the title (other than the ones between brackets), three different match levels are discriminated:

- 100% match, the track titles are identical.
- 60% match, the first 60% of the characters of the track title are identical.
- No match, none of the above conditions is fulfilled.

If the track title matches for at least 60%, the album title is compared. Otherwise the Amazon result is considered to be about a different album. If the track title match was only 60%, a 100% match of the (simplified) album title is required. In case of a 100% track name match, one album title may be longer than the other, but the start should match. Given that a high level of similarity for both the album and the track name is required, the chance of false positives is extremely low. To ensure that the user will receive valid data, even in the unlikely case of a false positive, Amazon information takes precedence over MusicBrainz information when storing the information.

4. Personalisation

Parallel to the search, MusiDB implements personalisation mechanisms that support the user in finding interesting music. The idea is provide personalized recommendations to users based on their preferences. In the literature, a number of recommendation techniques that try to predict the interest of the user into a particular item are mentioned:

- *Content-based* [1]: items with properties similar to the ones that the user liked in the past are recommended.
- *Demographic* [1]: items that users with properties similar to the current users liked in the past are recommended.
- *Collaborative* [5]: the choices of people that liked similar objects as the current users are recommended.

These methods have different advantages and disadvantages [3]. Content-based methods, for example, are limited to a rather static view of the user’s preferences and is not able to react adequately to new trends, which is important in the domain of popular music. Demographic recommenders are better able to deal with changing preferences, but they require some information that the user is not always willing to provide. Collaborative filtering is a good alternative to demographic filtering as it compares users thereby reacting to trends but it does not rely on information about the users and the items besides information about how users rated items. For this reason, collaborative filtering was chosen for the MusiDB system.

Based on the idea of collaborative filtering, two types of personalisation are used in the MusiDB application. Firstly, there is the prediction whether or not the user will like an album or artist. Secondly, MusiDB selects new albums to recommend to the user. Both of these are based on what is known about the user at that moment. The application does not force the user into answering a fixed set of questions before it can make a prediction. This does have consequences for the algorithm used, since there is absolutely no guarantee about the information that will be available. In order to be able to handle this uncertainty, a new approach had to be developed.

4.1. Prediction

A number of approaches for comparing and predicting user ratings have been proposed [2]. The basic idea behind the prediction is the following: if the user has a lot in common with another user, their opinions on an album or artist may very well be the same. *K-nearest neighbour* exploits this fact by looking for similar instances for which the requested attribute is known [4]. To determine the distance between two instances, a number of variables must be taken into account. Using a fixed set has a number of drawbacks:

- It would require the user to provide values for these variables (e.g. by answering questions regarding music) before any prediction could be made.
- The selected questions (variables) should provide enough information to be able to separate different classes of users.
- Since information on album and artist ratings is collected anyway (to be able to predict a value, reference attributes from other users must be available), a lot of information is wasted by not taking this data into account when selecting the nearest neighbours.

The solution implemented in MusiDB takes another approach which might be called ‘*weighted N-nearest neighbour*’, where *N* stands for the entire population of users. So, every other user in the population has a weighted vote, based on the number of common ratings. The algorithm to determine the predicted rating of user *u* for artist *a*, is as follows:

1. Start with $R = 0$ and $VoteCnt = 0$.
2. Create a list l_a of users that have a rating for artist *a*.
3. For every user $p \in l_a$ with rating r_p for *a*.
 - 3.1. For every common rating between p and u with the same rating value:
 - a. Add r_p to R .
 - b. Increase $VoteCnt$ by 1.
4. Calculate the predicted rating $r_{predicted} = R/VoteCnt$.

See also Fig. 6.

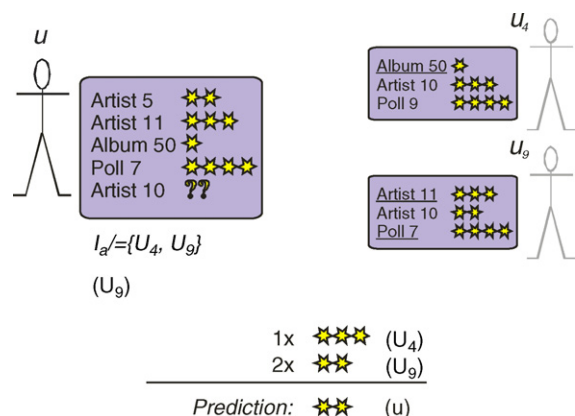


Fig. 6. Predicting a rating.

There are many albums and artists in the database. Since each user only rates a very small portion of these items, it may be a problem to acquire enough votes for a reliable prediction. The chance of two independent users having ratings in common is just not big enough. MusiDB implements three mechanisms to avoid unreliable predictions.

Firstly, MusiDB uses an aggregation mechanism in the form of a poll to quickly gather common data. The poll consists of questions about preferences for music styles. Answering these queries helps to compute the recommendations but, as we will discuss later when linking artists and albums to genres, the user is not obliged to answer them.

The second method also related to the poll is to motivate the user to provide input. To encourage people to participate in the poll, the tactic of “sudden onset” is used to attract visual attention to the poll. Though sudden onsets are used in reaction time experiments, the use in the MusiDB aims for the same effect, i.e. sudden onsets can be very powerful attractors of attention [12]. The idea is not to show a poll all the time, but to show a poll at certain intervals to the user, each time attracting the user’s attention. To make it easy to provide album and artist ratings, stars are added to each artist and album in the search result which can be clicked on to provide the system with rating information for that item (see Fig. 8).

The third way of avoiding unreliable predictions is by defining a reliability threshold. This threshold defines the minimum amount of votes that are required for a reliable prediction. If the number of votes is lower than the required number, then more users will get a vote. This is done by also granting users a vote that has common ratings with a difference of 1 star as the next algorithm shows:

1. Start with $R=0$ and $VoteCnt=0$.
2. Create a list l_a of users that have a rating for artist a .
3. For every user $p \in l_a$ with rating r_p for a :

- 3.1. For every common rating between p and u , with a difference of 1 star at most:
 - a. Add r_p to R .
 - b. Increase $VoteCnt$ by 1.
4. Calculate the predicted rating $r_{\text{predicted}} = R/VoteCnt$.

If the threshold is still not met, there are two options. One option is to take the whole population average into account. The other is not to show a prediction at all. The second option seems to be the best from a statistical point of view; the first was used during testing.

4.2. Recommendation

The recommendation algorithm takes a similar approach as the rating predictions, but instead of calculating one prediction for a specific album, it calculates predictions for all albums that have been rated by other users that have a common rating.

1. For every rated item i of user u , create a list l_i of users that have given i the same rating.
2. For every user $p \in l_i$ add the rated albums of that user to list l_a together with the rating.
3. Remove all unique albums that occur less than t_n times from l_a . (Where t_n is a pre-defined reliability threshold.)
4. For every remaining unique album $a \in l_a$ calculate the average rating r_a for that album.
5. Remove any albums with an average below t_a . (Where t_a is the minimum required predicted rating for recommendations.)
6. Select the best album that has not been shown recently.

The data gathered in the prediction and recommendation process can be used to derive secondary information. As stated in Section 1, the ratings may provide links between information from different sources. In MusiDB, an experimental addition

```
<?xml version="1.0" ?>
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:mdb="http://www
  xmlns:dc="http://purl.org/dc/elements/1.1/">
- <mdb:artist rdf:about="http://musicbrainz.org/artist/f27ec8db-af05-4f36-916e-3d57f91ecf5"
- <mdb:styles>
- <rdf:bag>
- <rdf:li rdf:resource="http://www.cs.vu.nl/~rstegers/iwa/styles/POP.xml">
  <mdb:association>0.98112522432469</mdb:association>
  </rdf:li>
- <rdf:li rdf:resource="http://www.cs.vu.nl/~rstegers/iwa/styles/ROCK.xml">
  <mdb:association>0.75</mdb:association>
  </rdf:li>
- <rdf:li rdf:resource="http://www.cs.vu.nl/~rstegers/iwa/styles/OCCASIONAL.xml">
  <mdb:association>0.5</mdb:association>
  </rdf:li>
- <rdf:li rdf:resource="http://www.cs.vu.nl/~rstegers/iwa/styles/MILITARY.xml">
  <mdb:association>0.41296117202215</mdb:association>
  </rdf:li>
- <rdf:li rdf:resource="http://www.cs.vu.nl/~rstegers/iwa/styles/NEW_AGE.xml">
  <mdb:association>0.35291289864636</mdb:association>
  </rdf:li>
- <rdf:li rdf:resource="http://www.cs.vu.nl/~rstegers/iwa/styles/HIP_HOP.xml">
  <mdb:association>0.25</mdb:association>
  </rdf:li>
```

Fig. 7. Example artist classification: Michael Jackson.

uses the correlation between answers given to the poll questions and ratings given to albums and artists, to link the music styles from the poll to the artists and albums. The associations are calculated real-time and are exposed via an RDF interface. This way MusiDB is able to combine information from different sources and provide a whole new source of data on the internet.

At the moment there is not enough empirical data available to really assess the quality of the associations. This has two reasons. First of all, a statistically significant result needs more users that rate albums and artists. The second reason is that in the current experimental implementation we only use a limited set of rather broad genre categories like ‘Rock’ and ‘Pop’. Nevertheless, we managed to achieve some interesting results that provide evidence that the mechanism is useful. Fig. 7 shows the RDF output of associations computed for the artist ‘Michael Jackson’. As we see, the strongest correlation exists with the Genre ‘Pop’ which is the expected result.

This way of categorizing artists and music can in turn be used to support the search and recommendation functionality of the portal as it could be used to provide the functionality of retrieving the most relevant artists for a given genre. This functionality is not implemented in the current system, but it can easily be done based on the information shown above.

5. Implementation

Implementing a system that has to combine potentially huge amounts of data, even taking into account all inputs present in the system, is a challenge on its own. Further, to make it easy to add other data sources to the system, there should be a structured and open architecture. And finally, the last problem to tackle is to implement a browser interface to provide all the functionality. In the next sections all these problems, with their solution, will be addressed.

5.1. Handling data

In the MusiDB, there are three different data sources. First there is the search result that comes from MusicBrainz. Second, there is the data from Amazon. Third, there is the potentially vast number of user ratings. MusiDB utilizes the Perl library provided by MusicBrainz for access to the repository. The biggest problem seems not to be the amount of data to transfer, but the number of TCP/IP connections that have to be built. In that respect, RDF is not the optimal way to retrieve data. The same holds for the matching with Amazon. Amazon returns the result in blocks of 10. For every page retrieved, a new connection has to be opened. Unfortunately, no good solution has been found, so the search mechanism is slow.

With a growing number of users, the amount of rating data available will also grow rapidly. Since specialized programs are available to store and retrieve such large amounts of data, MusiDB also leaves this task to such a (SQL) database server. Since for every prediction, every other user has a potential vote, it would most certainly clog the system if all this data was transferred every time a prediction is made. Caching all data is not really an option, since that would easily introduce consistency

problems. Therefore, MusiDB performs all required calculations server-side at the SQL server, by applying specialised SQL. The calculations as described earlier require much matching, counting and averaging. The SQL language provides for these functions and by using these, the data-transfer between the application and the server is limited to the queries and the resulting predictions.

5.2. Architecture

To allow for an easy extendible architecture that fits well with existing web servers, the business logic of MusiDB has been implemented in Object Oriented Perl. The structure of the objects has been kept very close to the data model (Fig. 2). This led to an actual hierarchy of objects, the *Results* object at the base, containing a list of *Artist* objects. For every result returned by MusicBrainz, the artist list is checked. If the artist does not yet exist in the list, it is added. The identified artist node is made active and checked for the track title that was found. If required, the track is added in the form of a *Track* object. Under this *Track* object, an *Album* object is created with information on the album the track was found on. The result of this process is a tree with artists, tracks and albums.

The described hierarchy makes it possible to assign each object partial responsibility in case new data is added. For example, if new data from Amazon has to be added, the *Results* object calls all its children – the *Artist* objects – to make them contact Amazon in search for results for the respective artist. These objects will in turn, call all their children with the results to allow track title and album title matching (by the *Track* object and the *Album* object, respectively). This process is called the ‘Waterfall of Data’. During this process gaps in the information can be filled. For adding the rating predictions the same process is used. Only now both the *Artist* and the *Album* object call the prediction calculation routines for their content and forward the call to the children. The *Track* object only forwards the call and has nothing further to do.

For outputting the consolidated data, the hierarchical structure helps too. The structure of the tree fits perfectly onto the structure of HTML. Every object displays the data it contains, providing a placeholder in which the child object can insert its own data. Currently, output modes are HTML and plain text, both implemented through this mechanism. Since the structure of the displayed data closely follows the tree structure, this is somewhat limiting. For example, it is hard to have the track displayed first with the different artists performing it.

5.3. Web interface

The MusiDB GUI consists of two input text fields for, respectively, a song title and an artist name to initiate a search. A login field is available for returning users. Frequently a poll will show up in a separate area of the window, to ask the user for feedback on genres. Within the search results albums and artists can be rated by clicking on stars behind the items. The number of stars that are lit either represent the rating given by the user, or a prediction in case no such rating is available. When enough

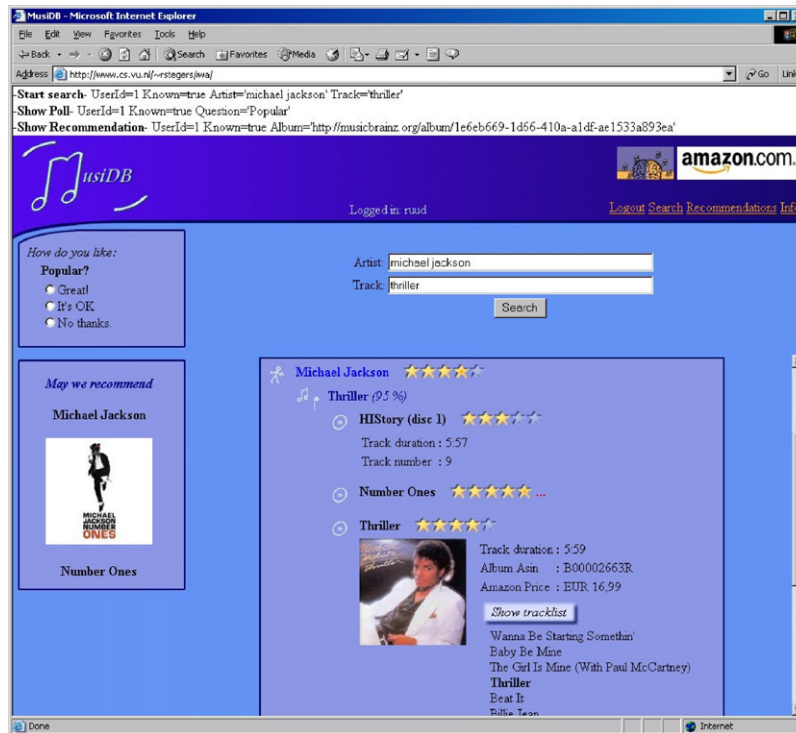


Fig. 8. Screenshot of the MusiDB Web Interface.

information is present about the user, a recommendation will be displayed. A Screenshot of the interface is shown in Fig. 8. The complete screen video of its use can be accessed at [14].

By using a web interface to access the data, everybody with a compatible browser can access all the information regardless of the operating system and hardware platform. The stateless nature of the HTTP protocol does however introduce some problems. In principle, every page is just static after it is displayed. It is waiting for user interaction to initiate a new page.

In MusiDB, however, a threaded architecture was desired to allow for simultaneous searching, provision of polls and recommending albums (Fig. 9). By using frames, every individual frame can be filled independent. MusiDB adds control frames – which are usually invisible – to make timer controlled updates to

the visible frames.¹ Answering a poll question will load a Perl script in the poll control frame that has two functions. One, it updates the database. Two, it produces a page with timed JavaScript that displays a blank poll window that after some time will use the control frame to display a new poll.

Since HTTP is stateless, it is required to incorporate all session data in all the communication between the client and the server. This means that every newly loaded page should contain the required references: every link is augmented with parameters containing, for example, the UserId. This is done transparently by (server side) PHP. As MusiDB now has a Perl script that generates an HTML page with PHP that generates JavaScript, the whole thing might easily get very hairy. To prevent this, all page control code has been written using very readable functions, encapsulating the PHP/JavaScript code it produces.

6. Conclusion

MusiDB has proven the possibility to use state-of-the-art web technologies like RDF models, web services and recommendation techniques to build applications that combine multiple live data sources. At the same time, the system is a proof of concept for the use of MusicBrainz as a universal repository for music information. It also provides an added value to MusicBrainz in terms of rating and recommendation mechanism that use MusicBrainz data.

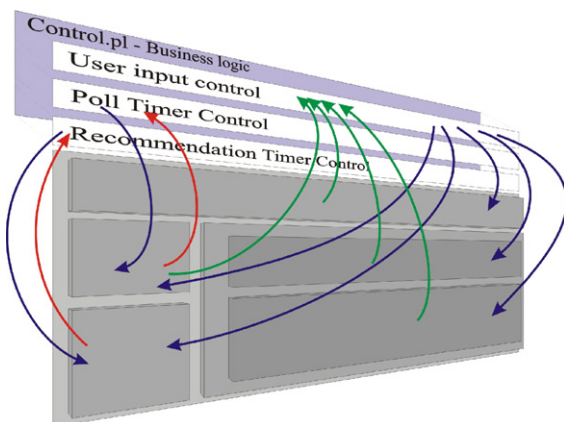


Fig. 9. Thread-like control mechanism.

¹ This required timer construction is the source of the current compatibility problems of the page with browsers other than Internet Explorer. There is no standardized way to implement timed-events in JavaScript.

Some issues need to be addressed, primarily concerning the efficiency of the search process. By using the MusicBrainz libraries, more data is gathered than strictly required. Parsing the returned RDF data and only following the links that contain required data may reduce the time spent on searching MusicBrainz. The fact that the Amazon interfaces force the application into fetching the data 10 results at a time also introduces a significant delay. This delay might be solved by using multiple threads that simultaneously fetch the Amazon results. However, this would require concurrency control to be added into the object hierarchy.

The most advanced feature of MusiDB is the functionality that determines the correlation between items, artists and albums and certain genres based on an active user poll. We believe that using user ratings as a basis for assigning information items to categories is a promising approach. In order to get reliable results, a large user base is needed, but in cases where such a user base is given (e.g. in user communities interested in digital music) this way of classifying information is the method of choice, especially, because any assignment of an artist to a genre contains some personal judgement.

At the moment, the system actually does not make much use of the knowledge representation and reasoning capabilities of semantic web languages. Most of the data is encoded in XML and the recommendation methods work directly on the data. An interesting line of future research is to explore the possibilities provided by semantic web technologies for enhancing integration and recommendation. A good example of how richer representations can be used to provide an integrated view on data and to support semantic-based methods on top of this integrated view is the work on the TAP knowledge base [7,8]. As TAP also covers the domain of artists and music, it could be used as an additional source of knowledge in the MusiDB system. In recent work Middleton and others also show how categorizations of data can help to improve recommendation algorithms [6].

There are many directions in which the system itself could be extended. Besides the obvious task of adding more information sources, the provision of more sophisticated search functionalities which can also include genre information is a useful extension. Future enhancements of the system could be to keep

track of user actions regarding the proposed recommendations and listed albums; when a recommendation is clicked on, or when a listed album is clicked on (after which a new browser window is opened to Amazon), the system could consider this album as an album the user especially likes.

References

- [1] C. Basu, H. Hirsh, W. Cohen, Recommendation as classification: using social and content-based information in recommendation, in: Proceedings of AAAI-98, Menlo Park, CA, USA, 1998, pp. 714–720.
- [2] J.S. Breese, D. Heckerman, C. Kadie, Empirical analysis of predictive algorithms for collaborative filtering, in: F. Gregory, Cooper, Serafin Moral (Eds.), Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, 1998, pp. 43–52.
- [3] R. Burke, Hybrid recommender systems: survey and experiments, in: User Modeling and User-Adapted Interaction, vol. 12, issue 4, Springer, 2002, pp. 331–370.
- [4] T.M. Cover, P.E. Hart, Nearest neighbour pattern classification, *IEEE Trans. Inform. Theory* 13 (1967) 57–67.
- [5] D. Goldberg, D. Nichols, M. Oki Douglas Terry, Using collaborative filtering to weave an information tapestry, in: Communications of the ACM, vol. 35, issue 12, ACM Press, New York, USA, 1992, pp. 61–70.
- [6] R. Guha, R. McCool, TAP: a semantic web test-bed, *J. Web Semant.* 1 (1) (2003).
- [7] R. Guha, R. McCool, E. Miller, Semantic search, in: Proceedings of the 12th International Conference on World Wide Web, Budapest, Hungary, ACM Press, New York, USA, 2003, pp. 700–709.
- [8] S.E. Middleton, N. Shadbolt, D. De Roure, Ontological user profiling in recommender systems, *ACM Trans. Inform. Syst.* 22 (1) (2004) 54–88.
- [9] P. Resnick, H. Varian, Recommender Systems, in: Introduction to Special Section of Communications of the ACM, vol. 40, issue 3, March 1997.
- [10] S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H.-P. Schnurr, R. Studer, Y. Sure, Semantic community web portals, *Comput. Networks* 33 (1–6) (2000) 473–491.
- [11] A. Swartz, MusicBrainz: a semantic web service, *IEEE Intell. Syst.* January–February (2002) 76–77.
- [12] S. Yantis, Attentional capture in vision, in: A.F. Kramer, M.G.H. Coles, G.D. Logan (Eds.), *Converging Operations in the Study of Selective Attention*, American Psychological Association, Washington, DC, 1996, pp. 45–76.
- [13] http://www.amazon.com/gp/browse.html/104-7781315-3496759?_encoding=UTF8&node=3487571.
- [14] <http://prauw.cs.vu.nl/MusiDB/> (download.avi file and play locally).
- [15] <http://prauw.cs.vu.nl/MusiDB/MusiDB.zip>.
- [16] <http://www.musicbrainz.org/tagger/download.html>.
- [17] <http://www.cs.vu.nl/~rstegers/MusiDB/> (requires MS internet explorer).