SYNCPLAYER — AN ADVANCED SYSTEM FOR MULTIMODAL MUSIC ACCESS

Frank Kurth, Meinard Müller, David Damm, Christian Fremerey, Andreas Ribbrock, Michael Clausen
Universität Bonn
Institut für Informatik III
Römerstr. 164, D-53117 Bonn, Germany
{frank,meinard,damm,fremerey,ribbrock,clausen}@cs.uni-bonn.de

ABSTRACT

In this paper, we present the SyncPlayer system for multimodal presentation of high quality audio and associated music-related data. Using the SyncPlayer client interface, a user may play back an audio recording that is locally available on his computer. The recording is then identified by the SyncPlayer server, a process which is performed entirely content-based. Subsequently, the server delivers music-related data like scores or lyrics to the client, which are then displayed synchronously with audio playback using a multimodal visualization plug-in. In addition to visualization, the system provides functionality for content-based music retrieval and semi-manual content annotation. To the best of our knowledge, our system is moreover the first to systematically exploit automatically generated synchronization data for content-based symbolic browsing in high quality audio recordings. SyncPlayer has already proved to be a valuable tool for evaluating algorithms in MIR research on a larger scale. In this paper, we describe the technical background of the SyncPlayer framework in detail. We also give an overview of the underlying MIR techniques of audio matching, music synchronization, and text-based retrieval that are incorporated in the current version of the system.

Keywords: MIR systems and infrastructure, multimodal interfaces and music access, synchronization

1 INTRODUCTION

Classical MIR techniques typically work with a single type of music representation. Examples include most query-by-example scenarios like audio identification (where a short query signal is to be identified w.r.t. a large database of audio signals), melodic queries (where a note sequence is to be matched to a melody database), polyphonic search (where an excerpt of a score is searched in a score database), or text-based search (where a sequence of query terms is searched in a database of textual metadata). As a consequence of such approaches, all steps in the typical MIR chain of query formulation, content-based retrieval, and display of query results are based on the same type of data representation. However, it is evident that there is no single data representation that is equally suitable or even a natural choice to be used for all of the latter tasks. As an example, consider a user querying a melody database. While it may be the most natural option to just hum a melody resulting in a *waveform-based* query, most successful algorithms for melody-based retrieval work in the domain of *symbolic* music. On the other hand, retrieval results are most naturally presented by playing back an actual *recording* of the piece of music containing the melody, while a *musical score* or a *piano-roll* will frequently be the most appropriate form of visually displaying a query result.

Up to now, research on integrating different types of music representation within MIR scenarios has mainly focused on the aspects of query formulation and the actual retrieval algorithms: considering query formulation, many approaches like existing query-by-humming techniques (for example Pauws (2002)) use signal processing techniques to convert acoustic data to the symbolic domain in a preprocessing step. Concerning retrieval, methods have been proposed to query music in a cross-domain fashion. For example, Pickens et al. (2002) present a system for querying polyphonic audio recordings in a database of polyphonic music given in the symbolic domain.

In contrast to this, there still is a significant lack of MIR systems for simultaneously *presenting* (which includes display, visualization, playback, and browsing) music and content-related data in an adequate *multimodal* form. Generally, such data is presented based on a single type of music data only. As an example, consider the display of query results in content-based audio retrieval: results are presented using acoustic playback along with a visualized waveform (presentation based on waveform data only). In MIDI-based retrieval, query results are typically presented as synthesized MIDI along with a MIDI-based piano-roll.

A particular lack of appropriate systems for content-presentation is found in several areas of MIR *research*. For example, there are only few user-friendly interfaces for analyzing results of the multitude of proposed algorithms for automatic music annotation such as score-to-audio synchronization (see below).

As a first step towards a proper multimodal presentation of music and content-related data, we recently proposed a prototypic client-server based system for synchronized playback and display of acoustic recordings together with content-related metadata (see Kurth et al. (2004)). The system's capabilities were demonstrated in a karaoke-like scenario where the displayed data consists of lyrics information. In contrast to other systems such as a similar system by Philocode LLC[1], which is available as a plug-in to various audio players (e.g., WinAmp or iTunes), SyncPlayer identifies a selected musical recording and a playback position therein entirely content-based. Hence it does not rely on the availability of further metadata such as ID3 tags.

In the present paper, we describe how the SyncPlayer framework has been redesigned to constitute an integrated MIR system incorporating functionality for multimodal display of symbolic data, content-based querying and semi-manual content annotation. To achieve this, we suitably adapt and combine recent techniques from audio identification (Clausen and Kurth (2004)), music synchronization (Müller et al. (2004)), and text retrieval. To the best of our knowledge, our system is the first to systematically exploit *automatically generated* synchronization data for content-based symbolic browsing in high quality audio recordings.

The main contributions of our system are summarized as follows:

- We propose an *integrated system* for symbolic (including textual) music searching and browsing with acoustic audio playback based on high quality recordings of the selected pieces of music.

- Our novel MultiVis (Multimodal Visualization) plug-in offers functionality for output as well as a *piano-roll-like interface for visualization and MIDI-based browsing*. This interface turns out to be a valuable tool for larger scale evaluation of research results in the domain of music synchronization.

- The newly developed *query engine* allows for textual search in a lyrics database, a ranked representation of query results, and playback of the exact positions of all matches. Our query engine is generic in that it is extensible to musical content-based search (for example, melody-based queries).

- An extended version of our Sync File Maker plug-in allows for *semi-manual annotation of lyrics* (or general textual) metadata to audio recordings. This functionality is supported by means of signal processing techniques for "slow-motion playback" of the underlying audio recording. By offering additional functionality for feature extraction, the plug-in can be used to extend and maintain the underlying audio database.

- The SyncPlayer framework is based on a *flexible client-server architecture*, which makes it suitable for arbitrarily distributed environments.

[1]http://www.philocode.com/minilyrics/

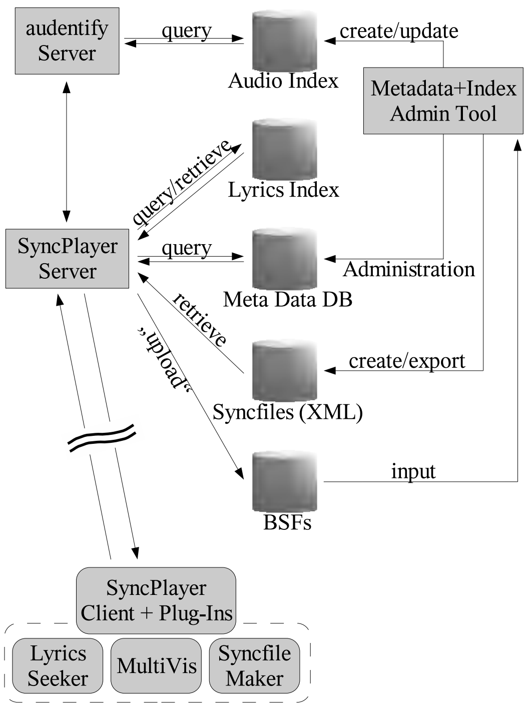


Figure 1: Overview of the SyncPlayer system architecture

Our paper is organized as follows. In the next section we describe the SyncPlayer system, consider its modes of operation, and give a detailed account on the constituting software modules and their interaction. In the subsequent sections we summarize the audio identification and music synchronization techniques that are used in the SyncPlayer framework. Audio identification is considered in Section 3, where we give a suitable modeling for the type of identification problem relevant to the SyncPlayer scenario. Moreover, we sketch a novel fast index-based search algorithm, which is capable of dealing with very large data collections. Section 4 briefly discusses algorithms for music synchronization used in our system. We furthermore describe a new version of our annotation module for semi-manual text-to-PCM synchronization. In Section 5 we consider content-based music queries. We propose a novel query module which, in the current version of our system, allows for text-based querying and browsing. In the concluding Section 6, we give an overview of our ongoing work and propose some problems for future research.

2 SYNCPLAYER SYSTEM

In this section we describe the SyncPlayer system in detail. After giving an overview of the basic operation mode, we describe the architecture of the software system. We give a detailed account on the client-server based framework and the plug-in modules used for playback, visualization, annotation, and querying. We also describe an administration tool for creating and updating the audio index, which is used for the audio identification task.

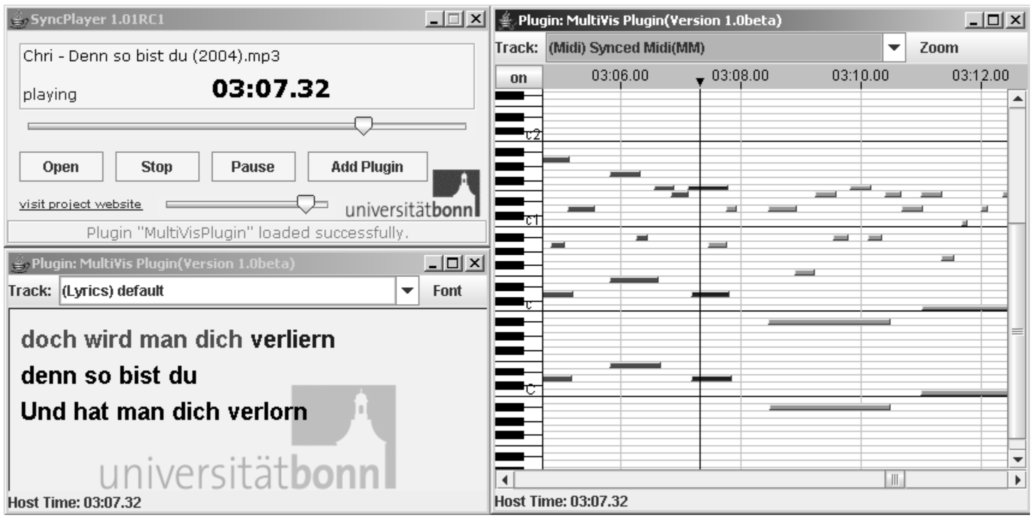


Figure 2: SyncPlayer client interface (left) together with two instances of the MultiVis plug-in, one displaying lyrics metadata, the other showing piano-roll data of an audio recording.

2.1 Basic Mode of Operation

Upon starting the SyncPlayer client application, the user selects an audio file from his local music collection, and acoustic playback starts automatically. Currently, MP3 and WAV audio are supported. At any time before or during playback, the user may launch any of the supplied plug-in modules. The different plug-ins offer functionality for visualization, semi-automatic annotation, and querying. The regular operation mode uses the MultiVis plug-in, which provides functionality for visualizing audio-related information *synchronously to acoustic playback*. Currently, two types of visualization are supported, one displaying textual information (e.g., lyrics or commentaries) in a karaoke-like fashion, the other giving a piano-roll representation of the musical notes occurring in the audio recording. The current playback position is indicated in both types of visualization by highlighting corresponding text and note positions, respectively. Available types of visualization (lyrics or piano-roll) are displayed in the track selection box of the MultiVis plug-in and may be subsequently chosen by the user. Depending on the musical contents and existing metadata for the selected audio recording, different types of visualizations (e.g., lyrics only, piano-roll only, both lyrics and piano-roll, multiple lyrics tracks) may be available. Note that an arbitrary number of MultiVis plug-ins may be launched simultaneously, each operating in an independently selectable visualization mode. Figure 2 shows the SyncPlayer client application and two instances of the MultiVis plug-in for an audio recording with available lyrics and piano-roll metadata. During playback, the piano-roll moves, while a locator is positioned on the current notes. All notes are highlighted at the point of their onset. Past and future notes are displayed in different colors.

2.2 General System Architecture

The SyncPlayer framework consists of several software components, which are summarized as follows:

- The user operates the *client application*, which performs acoustic playback and plug-in operation (visualization, annotation, querying).
- A remote computer system runs the *server application*, which is used for identifying the audio recordings the user selects. Furthermore, the server system is used for retrieving metadata related to the identified recordings such as lyrics or musical notes, which are then used by the client-side visualization plug-ins.
- A server-side *administration system* is used to create and maintain an index used for audio identification as well as for the various types of metadata.

The framework uses different types of music-related data:

- A *local collection of audio recordings*, which are selectable for playback.
- A (possibly large) *database of audio recordings* stored on a server. This collection is used in a preprocessing step for creating an audio index.
- This *audio index* is permanently accessible by the server and is used for fast audio identification
- The global metadata (e.g., title, artist, album, etc.) for the pieces contained in the audio database are stored in a *metadata table*, which is accessible by the server application.
- The content-related metadata (lyrics, piano-roll data, etc.) for the database items are stored in multi-track *Sync Files* (one Sync File for each audio recording; a Sync File may contain multiple tracks, each containing different metadata). There are two Sync File formats: *Binary Sync Files* (BSFs) are generated during the synchronization step (cf. Section 4). They store content-related metadata along with synchronization information (i.e., data relating actual time positions

to chunks of metadata) as well as audio feature data used for creating the audio index. *Textual (XML-based) Sync Files* contain content-related metadata along with synchronization information in a human-readable format.

Figure 1 gives an overview of the SyncPlayer system architecture. In the following subsections, interaction of the system components is illustrated in more detail.

2.3 Client System

When the user selects an audio recording and launches the MultiVis plug-in, SyncPlayer automatically tries to identify both the audio recording and the current playback position within the audio recording. The identification is performed content-based, i.e., based on the PCM waveform, making it independent of the availability of metadata (such as ID3 tags) or specific data formats. Moreover, the identification method as described in Section 3 is robust against signal distortions, lossy compression, and cropping. For audio identification, the client system first performs a feature extraction step. The features are then transmitted to the server system, which performs the actual identification. Note that the feature-based approach saves both bandwidth and helps preventing legal problems, which could result if parts of original audio recordings were transmitted from client to server. Upon successful identification, the client system is enabled to retrieve content-related metadata from the server.

Most of the client system is implemented in Java. As feature extraction generally is a time-critical task, it has been implemented in a C++-based module. This module is then accessed using the Java Native Interface (JNI) technology. Client-server communication is performed using Java's Remote Method Invocation (RMI) framework. Within this framework, the client communicates with the server using an interface of predefined functions. As a particular benefit of RMI, network communication is almost transparent. Hence, client and server may be located at virtually any point of the network (even on the same computer system) without requiring major changes in SyncPlayer's system configuration.

2.4 Server System

The SyncPlayer server consists of a scheduler component and a module for audio identification (the `audentify` server), which is also accessed using the Java RMI framework. When receiving a request for audio identification, the scheduler directs the request to the `audentify` server, which in turn performs audio identification by querying the audio index using fast search algorithms as described in Section 3. Similar to feature extraction, the actual index search is implemented in C++. Upon successful identification, `audentify` returns a unique file ID, which is subsequently used by the scheduler to retrieve the content-related data. For this, the scheduler first retrieves global metadata from the metadata table stored in a MySQL database. In particular, the metadata table contains references to existing Sync Files.

2.5 Plug-In Modules

Subsequently, any of the running plug-ins may request the global as well as content-related metadata stored in the retrieved Sync Files. Global metadata consists of the number of different tracks stored in a particular Sync File as well as content identificators (currently, lyrics and MIDI are supported). Access to the Sync Files is possible by requesting the server to deliver all metadata available for a certain time interval $[s, e]$. In the current version of our system, time stamps may be specified in milliseconds. The server returns a sorted list of metadata events consisting of pairs (t, d) where $t, s \leq t \leq e$, is a time stamp and d is some context-based metadata related to the time stamp t. Note that although such metadata might as well be given in sample precision, technical reasons such as different sampling rates and different feature extraction methods suggest to use milliseconds as a common basis for specifying time stamps.

Currently, three types of plug-ins are supported: in addition to the MultiVis plug-in, the Sync File Maker plug-in can be used for semi-manual text-to-audio synchronization, feature extraction, and Sync File creation (see Section 4 for details). Furthermore, the Lyrics Seeker plug-in provides text-based search in lyrics data (see Section 5).

2.6 The Index Admin Tool

The Index Admin Tool is a server-side Java application that is used to organize existing Binary Sync Files, metadata and audio recordings. It can be used to define groups containing arbitrary subsets of Binary Sync Files and generate both textual Sync Files as well as audio indexes from those groups. For audio identification, the `audentify` server uses the particular audio index that has been configured in the Index Admin Tool. Exported textual Sync Files are used by the SyncPlayer server to deliver content-related metadata to the client application.

3 AUDIO IDENTIFICATION

When the user starts playback of a specific audio recording and launches a visualization plug-in, the recording has to be identified before the retrieval of content-related data is possible. In order to avoid noticeable delays in displaying these data, identification has to be perfomed very efficiently. Furthermore, as audio recordings are available in different qualities or may have been edited prior to playback, identification should be robust against basic signal processing operations such as resampling, lossy compression, cropping, or equalization.

In the last five years, several powerful audio identification methods have been proposed such as described by Allamanche et al. (2001); Wang (2003); Clausen and Kurth (2004). We refer to the survey paper Cano et al. (2002) for a more detailed overview of existing methodologies.

In this section we first describe how we extend the identification technique of Clausen and Kurth (2004) to be suitable for the SyncPlayer framework. Then, we look at the more general scenario of audio matching and outline how future developments might allow identification of a musical work regardless of the specific interpretation.

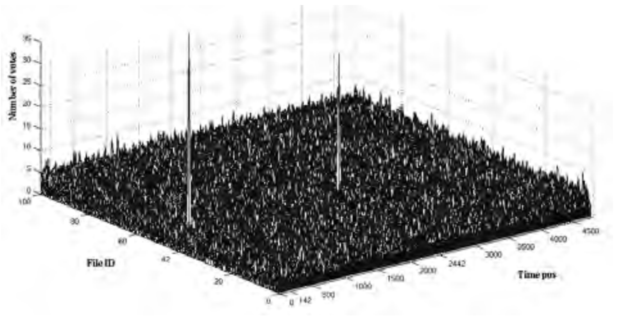


Figure 3: Voting matrix used for fast fault tolerant audio identification. The prominent peaks indicate two matches.

3.1 Identification Algorithm

Assume that a database of N audio recordings is represented as a sequence $\mathcal{D} = (x_1, \ldots, x_N)$ of finite energy signals $x_i, 1 \leq i \leq N$. From those signals, a compact audio index is constructed in a preprocessing step. For this, each signal is processed by a feature extractor F, which transforms a signal x_i to a feature document $F[x_i]$. The feature document $F[x_i]$ is composed of features $[t, j]$, each consisting of a feature class j and a time stamp t, indicating where the feature occurs within the signal. For what follows, we assume that F first transforms a signal x_i using a short time Fourier transform (STFT), resulting in a 2D time-frequency representation $\hat{x_i}$. Let $(f_1, \ldots, f_K)$ denote a sequence of K pitch classes. Then, for each local maximum of $|\hat{x_i}|$ at time position t and pitch f_j, a feature $[t, j]$ is added to $F[x_i]$. Applying feature extraction to all audio signals, we obtain a collection $F[\mathcal{D}] = (F[x_1], \ldots, F[x_N])$ of feature signals. Note that the latter construction just sketches the idea of our event-driven approach to feature extraction. For details, we refer to Clausen and Kurth (2004).

The audio index is then constructed by suitably adapting inverted file indexing. In particular, for each feature class j, we create an inverted file

$$H_{F[\mathcal{D}]}(j) := \{(t, i) \mid [t, j] \in F[x_i]\}$$

consisting of all pairs (t, i) such that a feature of class j occurs at position t within the ith audio signal, i.e., $[t, j] \in F[x_i]$. In this setting, audio identification is a simple task: consider a short audio signal q, e.g., an excerpt of a recording that the user selects for playback. Applying feature extraction, we obtain a feature query $F[q]$. Then one easily shows that an intersection

$$H_{F[\mathcal{D}]}(F[q]) := \bigcap\nolimits_{[t,j] \in F[q]} H_{F[\mathcal{D}]}(j) - t \qquad (1)$$

of suitably adjusted inverted files

$$H_{F[\mathcal{D}]}(j) - t := \{(\tau - t, i) \mid (\tau, j) \in H_{F[\mathcal{D}]}(j)\}$$

returns a set $H_{F[\mathcal{D}]}(F[q])$ of pairs (t, i), each corresponding to an occurence of the t-shifted query features $F[q]$ within the ith document. It turns out that for suitably sized queries q of only a few seconds of length, one may assume that each of those feature-based matches (t, i) identifies q as a specific segment of the audio signal x_i. For details, we again refer to Clausen and Kurth (2004).

To make audio identification robust to signal distortions, it is appropriate to allow a certain percentage p of feature mismatches. In terms of the query method presented in Eq. (1) this means we have to determine all pairs (τ, i) contained in $(100 - p)$ percent of the intersected lists $H_{F[\mathcal{D}]}(j) - t$. Instead of using a previously proposed dynamic programming approach (Clausen and Kurth (2004)), which may be too inefficient for solving this time-critical task for very large data sets, we propose a novel fast search technique based on a variant of geometric hashing, see Wolfson and Rigoutsos (1997). The method is outlined as follows: assume T is the maximum first coordinate of an element occuring in any of the lists $H_{F[\mathcal{D}]}(j) - t$. Further assume that all elements of those lists are positive (in practice, both conditions may easily be enforced in a preprocessing step). We then construct an integer array M of dimension $T \times N$ that will be used in the subsequent voting scheme. In a second step, for each $[t, j] \in F[q]$ we process the list $H_{F[\mathcal{D}]}(j) - t$. For each (τ, i) contained in that list, the entry (τ, i) of matrix M is increased by one. After this step, each of those entries $M[\tau, i]$ contains the amount of features of a t-shifted version of $F[q]$ matching document $F[x_i]$. Fig. 3 shows an example of the voting matrix M with two prominent peaks indicating two matches.

To account for slight time-distortions of query q and original signal x_i, adjacent entries of M may be pooled in a postprocessing step. This essentially amounts to calculating sums $M'[\tau, i] := \sum_{\ell=-\lambda}^{\lambda} M[\tau + \ell, i]$ for all positions τ. After this, all entries of M' exceeding a certain threshold are identified as matches. Note that since internal memory for M may be allocated in advance and no further postprocessing is necessary, this procedure is very efficient. To achieve a further speed-up, M may be organized in a hierarchical manner: at a coarse level, one identifies (for example rectangular) regions of M containing only few votes, which are then rejected at an early stage. As detailed by Ribbrock (2005, to appear), the latter approach amounts in a considerable gain in efficiency as compared to previous approaches like Wang (2003).

3.2 Towards Audio Matching

In the SyncPlayer scenario, audio identification in the above sense appears to be quite restrictive, particularly in the case of classical music. Because of the multitude of different available performances (and recordings) of one and the same piece of music, it is unrealistic to assume that a user owns the same version of a piece of music as is stored in the server's database. Hence, it will be of great interest to identify a piece of music regardless of the specific performance. We will call this problem *audio matching*. Müller et al. (2005) describe an approach to audio matching that works well for a broad class of Western music. The approach is based on using rough harmonic progressions, which is often sufficient for characterizing a piece of music. Although the latter approach to matching is still to be sped up by suitable indexing mechanisms, it should be feasible to integrate audio matching into future

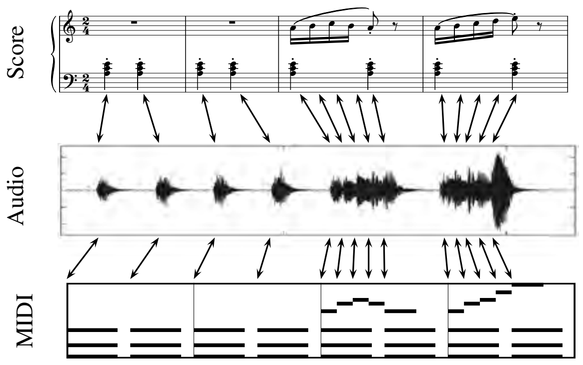


Figure 4: Synchronization of music data in the score (top), audio (center), and MIDI (bottom) data formats representing the same piece of music (first four measures of Etude no. 2, op. 100, F. Burgmüller).

versions of the SyncPlayer framework to achieve independence of the actual performance of a piece of music.

4 MUSIC SYNCHRONIZATION

In our SyncPlayer system, synchronous display of content-related musical data relies on appropriate methods for generating corresponding *synchronization data* in a preprocessing step, which is then stored in the previously described Sync Files. This amounts to linking the respective data (like lyrics or score data) to the actual musical recordings. In the last three years, research has focused on developing first approaches for automatically performing these so called *synchronization* tasks. Important examples include the synchronization of music data in the audio, MIDI and score domain as proposed by Soulez et al. (2003); Hu et al. (2003); Turetsky and Ellis (2003); Müller et al. (2004); Raphael (2004). In this section, we first give a brief introduction to music synchronization and then summarize our method for automatically synchronizing score-to-audio data. Finally, we describe the Sync File Maker plug-in, which was developed for semi-manual synchronization of audio recordings to lyrics data.

4.1 Score-to-Audio Synchronization

In score-to-audio synchronization, we consider the score of a particular piece of music as well as an audio recording of the same piece. Then the synchronization problem may be stated as follows: given a musical onset time in the score representation, determine the corresponding physical onset time within the audio representation. Synchronization of score and audio data is illustrated in the upper part of Fig. 4. Note that score and audio data differ fundamentally in their respective structure and content. On the one hand, the score consists of note parameters such as pitches, note onsets, or note durations, leaving a lot of space for various interpretations concerning, e.g., the tempo, dynamics, or multi-note executions such as trills. On the other hand, the waveform-based audio recording of a particular performance encodes all the information needed to reproduce the acoustic realization—note parameters, however, are not given explicitly.

To synchronize score and audio data, our approach described in Müller et al. (2004) proceeds in two stages: in a first step, semantically meaningful descriptors are extracted from the score and audio data streams making them locally comparable. In short, the audio signal is analyzed using an IIR subband filterbank consisting of one filter for each musical pitch. Subsequent processsing results in one *onset signal* for each of the subbands, where note onsets show up as prominent peaks. Finally, a peak picking step is used to obtain a score-like representation of the audio signal, which consists of one note event (p, t, s) for each note candidate of strength s detected in subband p at onset time t. It is straightforward to convert the score data to a similar format. In a second step, the two data streams are synchronized by minimizing a suitable cost functional using an alignment technique based on dynamic time warping (DTW). Here, the score data is used as a kind of ground truth in that only those note events extracted from the audio signal are considered that have an explicit counterpart in the score data stream.

Symbolic data is frequently stored in the widely used semi-symbolic MIDI format, which, in contrast to score formats is capable of accurately representing onset positions of a particular performance. Fortunately, the latter synchronization techniques may be adapted to the problem of MIDI-to-audio synchronization. The bottom part of Fig. 4 illustrates this type of synchronization.

The score material considered in Müller et al. (2004) was converted to the MIDI format and then synchronized to actual piano performances. To make these results available to our SyncPlayer, we exported the synchronization data to SyncFiles and indexed the audio recordings as described in Section 3.

4.2 Text-to-Audio Synchronization

Currently, text-to-audio synchronization at the lyrics level still requires manual interaction. To assist this process, our framework provides the Sync File Maker plug-in. Basically, the plug-in allows the user to load a text file containing lyrics data for a selected audio recording. During playback, the user may then specify lyrics positions by pressing keys on his keyboard at the point of their acoustic output. As this process is frequently very difficult due to playback speed and rhythmic complexity of a particular audio recording, we extended the plug-in by integrating a time-scaling functionality. Using a sliding bar, the user may specify an adequate playback speed. The plug-in then outputs a slower version of the original audio recording having the same perceptible pitch (i.e., we use to time-scaling to suppress pitching effects). Our time-scaling method is an adapted real-time version of the WSOLA-method by Verhelst and Roelands (1993), which basically consists of the follwing steps: the original signal is cut into overlapping parts using a sliding window of a time-varying step-size of $s_1' \in [s_1 - \delta : s_1 + \delta]$ samples. The resulting signal parts are then used to construct a new signal. To achieve time scaling, adjacent parts are overlapped using a new fixed step size s_2. In order to obtain an s-times as slow version of the original signal, we choose

$s_2 := s \cdot s_1$. To suppress audible artifacts, in each step a local autocorrelation between the current signal part and its local neighborhood within the original signal is used to determine the latter "best fitting" s'_1, i.e., we choose s'_1 to maximize autocorrelation. Using fast convolution, we designed an algorithm for performing this step faster than in real time (Java-based on an 1.6 GHz Intel P4 platform).

Methods for automatically synchronizing music to lyrics data constitute an upcoming field of research. A recent approach by Wang et al. (2004) presents first results for line-level based text-to-audio synchronization.

5 QUERY ENGINE

Many types of content-based retrieval like full-text-, melody-, or score-based retrieval (see Clausen and Kurth (2004)) allow for an exact localization of a given query within a matching document. In the SyncPlayer scenario, it is then possible to provide the user with an adequate presentation of the queries' occurence in the retrieved document. In particular, we may use SyncPlayer's MultiVis module to display any of the matches while synchronously playing back a high-quality audio recording of the matching positions.

As it is quite natural for many non-expert users to formulate queries by specifying fragments of the lyrics occuring in a song, we integrated a prototypical text-based query engine into the SyncPlayer framework.

Our technique for lyrics retrieval is term-based and also uses inverted files as an index structure. In a preprocessing step, we generate the inverted files from our Sync Files $\mathcal{S} := (S_1, \ldots, S_N)$: basically, for each term t, the inverted file $H_{\mathcal{S}}(t)$ contains all pairs (p, i) such that t occurs as pth lyrics term within Sync File S_i. Using inverted files, query processing may be performed using the same type of intersection as in Eq. (1): a textual query $q := (t_1, \ldots, t_k)$, which is a sequence of words (terms), is then used to calculate the set of all matches

$$H_{\mathcal{S}}(q) := \bigcap\nolimits_{j=1}^{k} H_{\mathcal{S}}(t_j) - j. \qquad (2)$$

This yields all positions of occurrences of the query q within the lyrics stored in the Sync Files. To account for typing errors, we preprocess each query term t_j and determine the set T_j of all terms in our inverted file dictionary[2] that have a close edit distance to t_i. Then, instead of only considering the exact spelling t_j by using $H_{\mathcal{S}}(t_j)$ in (2), we consider the union $\cup_{t \in T_j} H_{\mathcal{S}}(t)$ of occurrences of all terms close to t. To account for word errors like inserted or omitted words, we preprocess all word positions occurring in (2) by a suitable quantization. This amounts to replacing j by $\lfloor j/Q \rfloor$ and each element (p, i) of an inverted file by $(\lfloor pj/Q \rfloor, i)$ for a suitably chosen integer Q prior to calculating the intersection ($Q = 5$ was used in our tests). The latter yields a coarse approximation of the set of all matching positions, which may then be refined in a postprocessing step. There, we compare q with the exact order of occurrence of all query terms and their actual proximity at the matching position, a process which is very similar to Google's proximity-based ranking.

[2]i.e., the set of all terms with an existing inverted file

Figure 5 shows the Lyrics Seeker plug-in for textual queries (right). A query string can be entered at the top, a list of ranked query results is displayed below. Lyrics positions of matched query terms are highlighted. Upon selecting a matching position, the MultiVis plug-in is launched (left) and playback starts at the position of the match. Note that also in this scenario legal reasons require that a user has access to the actual audio recordings, i.e., the recordings have to be stored on the computer running the SyncPlayer client.

To conclude this section we remark that it should be possible to improve lyrics-based retrieval significantly by using some elaborate text retrieval techniques such as stemming, analysis of phrases or thesauri. As a particular adaptation to lyrics-type data, it should moreover be beneficial to consider some more Google-like ranking strategies such as term occurrence at prominent places (such as song title, chorus or hook line).

6 CONCLUSIONS AND FUTURE WORK

The proposed SyncPlayer framework constitutes a powerful tool for accessing music-related contents in that it combines different modalities like acoustic, graphical and textual representations in a synchronous fashion. This becomes possible by suitably integrating elaborate methods from music- and text-based retrieval with signal processing techniques. We would like to stress that the current version only constitutes the next step towards a framework for integrated querying, display, and annotation of musical content. There are various interesting directions for extending the proposed SyncPlayer framework as well as the plug-in modules in future work:

- Extension of the query engine to support melody- or general polyphonic score-based queries. For this we plan to integrate our own retrieval algorithms in the respective fields, see Clausen and Kurth (2004).

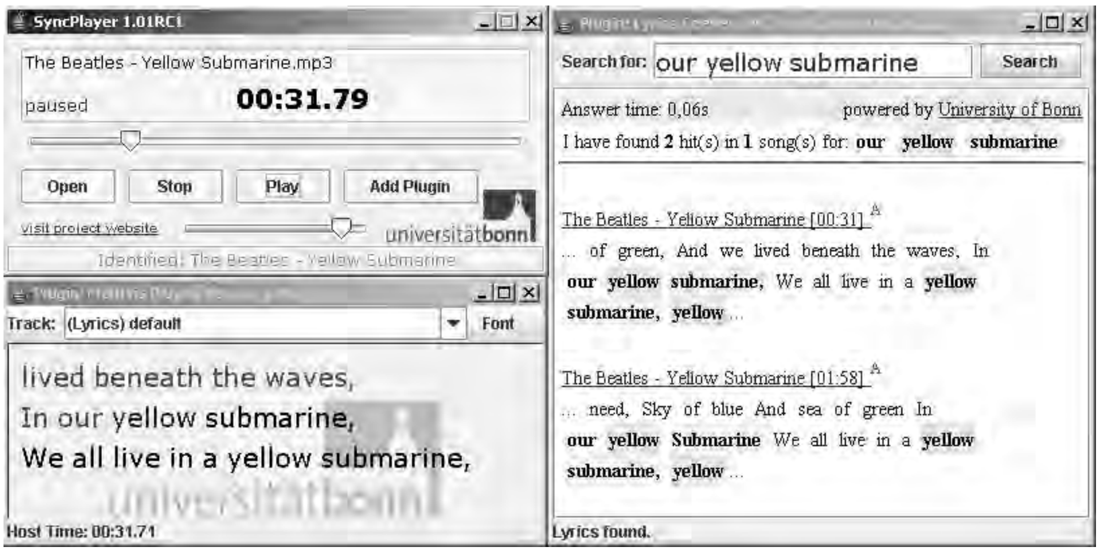
- Adaptation of audio matching techniques developed by Müller et al. (2005) to allow for version-independent audio identification,

- Extension and integration of more elaborate algorithms for automatic synchronization of music and lyrics to audio recordings.

- Improvement of the Lyrics Seeker plug-in to incorporate advanced ranking mechanisms and concepts for fault tolerant queries.

- Support text-based *browsing* functionality with synchronous playback.

For using SyncPlayer in commercial or at least "rights-critical" scenario, it would moreover be beneficial to incorporate some kind of digital rights management. Thus it could be possible to allow for streaming audio recordings in a client-server setting. This is of particular interest in retrieval scenarios, where a user generally wants to listen to his query results.



The Beatles - Yellow Submarine.mp3
paused 00:31.79
Identified: The Beatles - Yellow Submarine
Track: (Lyrics) default

lived beneath the waves,
In our yellow submarine,
We all live in a yellow submarine,

Host Time: 00:31.71

Search for: our yellow submarine

Answer time: 0,06s powered by University of Bonn
I have found **2** hit(s) in **1** song(s) for: **our yellow submarine**

The Beatles - Yellow Submarine [00:31]
... of green, And we lived beneath the waves, In **our yellow submarine,** We all live in a **yellow submarine, yellow** ...

The Beatles - Yellow Submarine [01:58]
... need, Sky of blue And sea of green In **our yellow Submarine** We all live in a **yellow submarine, yellow** ...

Lyrics found.

Figure 5: Lyrics Seeker plug-in: text-based query (right, top) and retrieval results (right, bottom). Left: SyncPlayer client (top) and MultiVis plug-in (bottom), which is launched upon selection of a query result.

SOFTWARE AND DEMO

The client version of our SyncPlayer is available from our webpage[3]. Note that for legal reasons we are generally unable to provide audio recordings for download. However, to test full functionality of the MultiVis plug-in, the web page offers some of our own recordings for free download.

REFERENCES

E. Allamanche, J. Herre, B. Fröba, and M. Cremer. AudioID: Towards Content-Based Identification of Audio Material. In *Proc. 110th AES Convention, Amsterdam, NL*, 2001.

P. Cano, E. Battle, T. Kalker, and J. Haitsma. A Review of Audio Fingerprinting. In *Proc. 5th IEEE Workshop on MMSP, St. Thomas, Virgin Islands, USA*, 2002.

M. Clausen and F. Kurth. A Unified Approach to Content-Based and Fault Tolerant Music Recognition. *IEEE Transactions on Multimedia*, 6(5), Oct. 2004.

N. Hu, R. Dannenberg, and G. Tzanetakis. Polyphonic audio matching and alignment for music retrieval. In *Proc. IEEE WASPAA, New Paltz, NY*, October 2003.

F. Kurth, M. Müller, A. Ribbrock, T. Röder, D. Damm, and C. Fremerey. A Prototypical Service for Real-Time Access to Local Context-Based Music Information. In *ISMIR, Barcelona, Spain*, 2004.

M. Müller, F. Kurth, and M. Clausen. Audio Matching via Chroma-based Statistical Features. In *ISMIR, London, GB (submitted)*, 2005.

M. Müller, F. Kurth, and T. Röder. Towards an Efficient Algorithm for Automatic Score-to-Audio Synchronization. In *ISMIR, Barcelona, Spain*, 2004.

S. Pauws. CubyHum: a fully operational query by humming system. In *ISMIR, Paris*, 2002.

J. Pickens, J. P. Bello, G. Monti, T. Crawford, M. Dovey, M. Sandler, and D. Byrd. Polyphonic Score Retrieval Using Polyphonic Audio. In *ISMIR, Paris*, 2002.

C. Raphael. A hybrid graphical model for aligning polyphonic audio with musical scores. In *ISMIR*, Barcelona, October 2004.

A. Ribbrock. *Schnelle Algorithmen zur Konstellationssuche in Multimediadaten*. PhD thesis, Department of Computer Science, University of Bonn, 2005, to appear.

F. Soulez, X. Rodet, and D. Schwarz. Improving polyphonic and poly-instrumental music to score alignment. In *ISMIR, Baltimore*, 2003.

R. J. Turetsky and D. P. Ellis. Force-Aligning MIDI Syntheses for Polyphonic Music Transcription Generation. In *ISMIR, Baltimore, USA*, 2003.

W. Verhelst and M. Roelands. An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech. In *Proc. ICASSP*, volume 2, pages 554–557, 1993.

A. Wang. An Industrial Strength Audio Search Algorithm. In *ISMIR, Baltimore*, 2003.

Y. Wang, M.-Y. Kan, T. L. Nwe, A. Shenoy, and J. Yin. LyricAlly: Automatic Synchronization of Acoustic Musical Signals and Textual Lyrics. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 212–219, New York, NY, USA, 2004. ACM Press.

H. Wolfson and I. Rigoutsos. Geometric Hashing: An Overview. *IEEE Computational Science and Engineering*, 4(4):10–21, 1997.

[3]http://www-mmdb.iai.uni-bonn.de/projects/syncplayer/