# Audio-Text Synchronization inside mp3 files: A new approach and its implementation

Marco Furini* and Lorenzo Alboresi

Computer Science Department
University of Piemonte Orientale
Spalto Marengo 33
15100 Alessandria Italy
furini@mfn.unipmn.it

**TR-INF-2003-07-04-UNIPMN**

**Abstract**

The large usage of multimedia portable devices has contributed to rapidly increase the demand for multimedia entertainment services. In this paper we focus on the karaoke service: several systems have been proposed but they are too difficult to be directly used over audio devices. Conversely, in this paper we propose a very simple approach to provide a karaoke-like service over any audio device that can play out mp3 files. Our approach is essentially new, as no additional files, beyond the mp3 file, are necessary. A simple description language has been designed and the resulting audio-text timing synchronizations are transparently stored inside the mp3 file. The effectiveness of our approach is proved through a developed java mp3 player. The simplicity of our approach along with the java portability allow a straightforward use of our player over any OS and over any audio device that supports java applications.

**KEY WORDS**
Entertainment, Novel Applications, Multimedia Technology, Karaoke, Mp3 audio file

## 1 Introduction

The popularity of the MPEG layer III audio files (mp3) [1] on the Internet confirms that mp3 became the dominant format for high quality digital music.

*Corresponding author. E-Mail: furini@mfn.unipmn.it

The large usage of these mp3 audio files has contributed to the development of portable devices designed for listening to mp3 audio files away from the computer. The success of these devices affected the design of several other devices that became mp3-compliant (e.g., palm, walkman, cellular, iPod, dvd).

In the near future, with the introduction of broadband wireless technologies (UMTS [2], WiFi[3]), Mp3 audio files will be more used, as users will be connected to the Internet regardless of their location, allowing them to have access to digital musical stores any time and any where. This will certainly increase the success of mp3 files. To confirm this trend, several wireless devices are now released with mp3 compatibility and several on-line music stores appeared on the web, so that consumers can directly buy and download digital music in mp3 format (for instance the iMusicstore [4]).

The success of mp3 files will further increase if textual information, such as lyrics, would be stored inside the mp3 file. In fact, this will allow new services to be proposed. For instance, any user with an mp3 compliant device could have a high quality audio karaoke-like service, by only using an mp3 file. In this way, music listeners may sing a song while listening to it; music producers may distribute a complete song (music and lyrics) within a single file and audio device producers may provide users with new multimedia services.

Karaoke is, in fact, a multimedia entertainment service that receives a large interest by providing users with on-screen lyrics information synchronized with audio playout. The success is highlighted by the presence of several karaoke model architectures (for instance, [5, 6, 7, 8, 9, 10]) that aim at combining different resources, such as audio, video and text. Roughly, different streams are stored over a support (for instance over a cd) and the karaoke system reads and combines these streams. As we better explain later, most of these karaoke systems do not provide high quality audio or need other resources in addition to the audio data (e.g., most of them use separate files for audio and textual information). This is a limitation as high quality audio is largely used and the need to have different resources in the Internet scenario is a complex, and sometimes annoying, approach. The service would be much more consumers appealing if audio and text were in the same file.

The contribution of this paper is the design of an mp3 karaoke-like service. Our idea is to insert lyrics and audio-text timing synchronizations inside an mp3 file, so that the karaoke-like service may be provided by only using the mp3 file. Our approach is essentially new in the sense that no additional files (i.e., text files) are required and that all the necessary information are provided within the mp3 file.

In fact, the insertion of textual information inside the mp3 file is not straightforward: the mp3 format [1] does not allow to store textual information inside it. For this reason, several investigations have been done in order to store textual information within an mp3 file and today, the most accepted way to achieve this, is the so-called ID3 tag [11]. This tag allows to have only a fixed-size of 128-bytes characters (organized in title, artist, album, year, comment and genre) at the end of the file. Needless to say, 128 bytes are not sufficient to store lyrics and audio-text timing synchronizations. Further, the location of this information (at the end of the file), allows to access these information only if the entire file is available (i.e., after the download and not while streaming the audio). Fortunately, a second version (ID3v2), has been released. ID3v2 allows to store textual fields at the beginning of the mp3 file, introduces additional fields (track, composer, copyright, comment, url) and removes the 128-bytes length constraint.

Since ID3v2 is largely used by mp3 players, we decided to use one of the ID3 tag to store lyrics and audio-text timing synchronizations inside the mp3 file. To this aim, we develop a synchronization description language (similar to SMIL [12] language), in order to describe audio-text timing synchronizations. These information are inserted in the *comment* field of the ID3v2 tag. Hence, comment-field is trojan used. In this way, the mp3 file format is not modified and the mp3 file can be played with any mp3 player.

However, to have a karaoke-like service, the player should be able to read the comment-field (filled with lyrics and audio-text timing synchronizations) and to display lyrics on the user's screen according to the read audio-text timing synchronizations. To this aim, we developed an mp3 player. This player has been written in Java language, in order to use our player over many different OS-systems and devices. For instance, the recent released cellulars from several vendors (e.g., Nokia, Motorola, Sony Ericsson, Panasonic) are Java Compatible and could immediately run our player and provide the users with high quality karaoke service.

The remainder of the paper is organized as follows. In Section 2 we present the characteristics of our approach along with a brief description of the mp3-ID3 tag; in section 3 we present an implementation of our approach and conclusions are drawn in section 4.

### 1.1 Related Work

There are several examples of karaoke-like services [5, 6, 7, 8, 9, 10] and one of the main concerns in the design of a karaoke system is the adopted synchronization strategy.

A possible (and traditional) solution to play out synchronized digital data is the MIDI technology that can be used to play karaoke clips. Another important technology to synchronize multimedia streams is the RealMedia [13], which allows to synchronize client-server multimedia streaming. Karaoke/Surestream is an example [6] and Karaoke Online [5] is another example.

Many are the karaoke societies that use SMIL [12], SureStream [6], FLIPS [14] and other client-server technology to provide streams synchronization.

All these systems provide karaoke by using different files (one for the audio object and the other for audio-text timing synchronizations) or by providing a non high-quality audio. Conversely, our goal is to provide a karaoke-like service using high-quality mp3 audio files and by storing audio-text timing synchronizations information within the mp3 file (i.e., only one file).

## 2 ATS Approach

In this section we explain characteristics and properties of our Audio-Text-Synchronization (ATS) approach, which aims at providing a high quality audio karaoke-like service using mp3 files. Our approach uses only one file (i.e., the mp3 file) to provide a karaoke-like service. The mp3 file is modified in order to contain lyrics and audio-text timing synchronizations information. However, the introduced modifications do not cause any problem to the audio playout. Hence, even though the mp3 file contains lyrics and timing information, it can still be played out by any mp3 player without any problem.

The mp3 file is hence self-contained: the karaoke-like service may be provided by any mp3 player able to manage the textual information (audio-text timing synchronizations) stored inside the file.

In this section we show how the mp3 is modified and we also present a very simple description language we designed in order to describe lyrics and audio-text timing synchronizations. By describing these information and by putting them into the mp3 file, we produce an mp3 file ready to provide a karaoke-like service.

In the following we show that our mechanism is very simple to implement, causing our approach to be exploited by the following categories of users:

- Music listeners may want to sing a song while listening to it;
- Music producers may wish to exploit our approach to distribute a complete song (music and lyrics);
- Audio device producers may want to provide users with the capability of reading what they are listening to.

### 2.1 Audio and Text with mp3 file

The MPEG layer III (mp3) [1] has been released in 1992 and provides high-fidelity audio at low bit-rates with little or no perceptual difference between the original signal and the reconstructed signal. It uses psychoacoustic models to remove the least perceptually relevant portions of the signal.

The achieved high compress ratio and the large usage of P2P systems have contributed to elect the mp3 as the dominant format for audio in the Internet. The success of this format caused several portable audio devices to be mp3-compliant and nowadays mp3-compatibility is provided with almost any audio device (from Hi-Fis to walk-mans, from audio car systems to cellulars).

This large success has been obtained even though the mp3 audio file format has been designed to store only audio information: no textual information can be stored within the file. This limitation produced several studies in order to find a way to insert textual information inside the mp3 file without compromising its effectiveness.

The mp3 file was first modified with the introduction of the so called ID3 tag [11]: a 128-bytes long tag able to contain different textual fields (Table 1).

| | |
|---|---|
| Song title | 30 characters |
| Artist | 30 characters |
| Album | 30 characters |
| Year | 4 characters |
| Comment | 30 characters |
| Genre | 1 byte |

Table 1: Format of the ID3v1 Tag.

The tag was inserted at the end of the file, hence the information could not be displayed until the player has access to the complete file. This does not cause any problem is the mp3 file is completely stored before its play out, but it arises problems if the mp3 file is played out in streaming (in this case the player can read the information only at the end of the play out).

For this reason, a second version was released with the name of ID3v2 [11]. This second version removes the length constraints, allowing fields to have any arbitrary length, introduces several other fields (track, comment, url, composer, copyright) and puts the textual information prior the audio data. This allows players to immediately have access to these textual information, regardless of the type of service provided to users (either streaming or download). Hence, players can play out the audio content and can show textual information on users' screen.

The ID3 tag is still in expansion and future versions of this tag will allow to store additional information, like pictures and lyrics, inside the mp3 audio file. The success of this approach is highlighted by noticing that, nowadays, almost all the mp3 players are compatible with ID3v1 and most of them are compatible with ID3v2.

Due to the success of the ID3 tag, we decided to not modify the mp3 structure in order to store textual information inside the mp3 file. We decided to use the *comment* field of the ID3v2 tag to store both lyrics and audio-text timing synchronizations. In this way the mp3 format is not affected and hence any mp3 player can read the mp3 file without any problem. Lyrics and audio-text timing synchronizations can be inserted in the comment-field as this field may have any arbitrary length.

## 2.2 Audio-text timing synchronizations

Storing textual information inside the mp3 file is necessary, but not sufficient to provide a karaoke-like service. In fact, it is necessary to well describe the audio-text timing synchronizations. In this section we present the characteristics of the language we designed to this purpose.

Currently, the most used synchronization language in multimedia stream is the SMIL language (Synchronized Multimedia Integration Language) [12]. SMIL is XML-derived mark-up language and it is designed to integrate continu-

```
<body>
  <par>
    <audio src="song.wma"></audio>
    <seq>
       <text begin="5s" dur="3s"
                   region="region1_1">
          You and me
       </text>
       <text dur="3s" region="region1_1">
          We used to be together
       </text>
       ...
       <text dur="2s" region="region1_1">
          Hush, hush don't tell me
       </text>
    </seq>
  </par>
</body>
</smil>
```

Table 2: Example of a karaoke SMIL file.

ous media into synchronized multimedia presentation. With SMIL it is possible to manage the timing behavior of a presentation, by defining which multimedia objects are to be loaded in a specific regions. To this aim, tags such as `<video>`, `<audio>` and `<text>` are exploited. To synchronize, two methods are available: `<par>` to concurrently execute multimedia objects in their region and `<seq>` to execute multimedia object according to a predefined sequential timing schedule. In Table 2 we show an example of a SMIL file that provides the user with a karaoke effects: audio is played out and lyrics are displayed on the screen according to specified timing descriptions. In this way, the user is provided with a karaoke-like service.

Since SMIL synchronizes different media streams and since our goal is to produce a single audio stream with audio-text timing synchronizations inside, we don't directly use SMIL, but we use it as a basis to design our SMIL-like language (ATS language). The ATS language will be used to describe the timing synchronizations between audio and text.

Since we are using the comment-field to store our textual information, we use two tags: one at the beginning of the information (`<begintext>`) and the other at the end (`</begintext>`). All the information within these two tags must be in the form **show-time** <**text**>, meaning that **text** should be displayed at showtime, where showtime is expressed in time units. It is worth noting that in our implementation, a time unit corresponds to 100 milliseconds (from several experiments we noticed this threshold is sufficient to describe the timing relation of each lyrics sentence. However, this value can be changed without problems).

In Table 3 we show an example of audio-text timing synchronizations de-

```
<begintext>
   55<You and me>
   77<We used to be together>
   105<Everyday together always>
       ...
   1937<Hush, hush darlin' Hush,>
   2000<Hush, hush don't tell me>
</begintext>
```

Table 3: Example of our ATS description language.

scription with our ATS language. The textual information contain both lyrics and audio-text timing relations. For instance, *77<We used to be together>* means that after 77 time units, the singer is singing *We used to be together* and so the sentence has to be displayed on the user's screen.

These timing synchronizations can be manually edited into the comment-field of the ID3 tag or, as we describe later, can be automatically inserted by the mp3 player we implemented.

Although these information are present within the mp3 file, a karaoke-like service is not yet provided to the end-user. In fact, there must be a player able to manage the comment-tag field and to determine lyrics and timing synchronizations. In the next section, we present an mp3 player implementation able to manage these information and to provide a karaoke-like service.

## 3 ATS player implementation

In this section we present details of the mp3 player we developed in order to provide a karaoke-like service to the user. The goal of our player is to play out the mp3 file, to read the textual information contained in the comment-tag field, to interpret them and to display them on the user's screen when necessary. This means that these information have to be shown according to the audio-text timing synchronizations present in the comment-tag field.

Our player has been written in Java language, in order to enhance software portability. In this way our player can be used over any OS or device that supports Java. Hence, our player can be run over several modern portable devices that support Java applications. For instance, several vendors (e.g., Nokia, Motorola, Sony Ericsonn) released cellulars that support Java applications and most of these devices already provide users with mp3 player. Hence, our karaoke-like service could be introduced in these devices without any difficultie. Before explaining the details of the implementations, we first introduce a tool that allows users to automatically insert lyrics and timing synchronizations inside the comment-field tag.

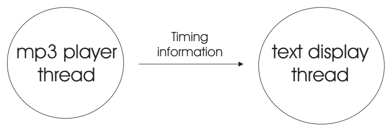


Figure 1: Threads synchronizations.

## 3.1 A tool for audio-text synchronizations

As we previously mentioned, lyrics and timing synchronizations may be manually edited and inserted into the comment-field. In fact, these information are completely described by textual information and hence any text-editor can be used to edit them. However, to avoid any possible mistake, we designed a tool that allows the user to automatically edit and store these information inside the mp3 file.

To run this tool, user must have two separate files: one with the mp3 audio and the other with the song lyrics. Lyrics must be written row by row (as they are written inside any cd booklet). With these two files, the user can listen to the mp3 audio file and have to press a button each time the singer executes a new row of the lyrics. At the end of the song, lyrics and audio-text timing synchronizations are automatically inserted inside the comment-field. From now on, the mp3 file contains lyrics and timing synchronizations and the text file is no longer used.

These information will be used by a player to provide a karaoke-like service: the mp3 player has to read the comment-field and to manage the information in order to provide the user with a karaoke-like service.

## 3.2 Player implementation

In this section we describe the characteristics of our mp3 player implementation.

As we mentioned, our player has been written in Java language and has to: i) play out an mp3 file, ii) read the textual information contained in the comment-field, iii) provide the users with a karaoke-like service.

In the following, we don't present any detail of the mp3 play out, as these details goes out of the paper contribution. Instead, we focus our description on the karaoke-like service implementation.

To this aim, we remind that lyrics and audio-text timing relation are present inside the mp3 file in the comment-tag. These information have a fixed structure, which starts with `<begintext>` and ends with `</begintext>`. All the information between these tags are considered as lyrics to be displayed. In particular each row must have the format `display_time <text>`, meaning that `text` has to be displayed at time `display_time`.

When an mp3 file is selected, our player opens it and checks whether the comment-field contains karaoke information or not. If present, the information are inserted into an array, indexed according to the display-time value. From now on, the mp3 play out is coupled with a karaoke-like service (each row is displayed on the user's screen according to its display time).

To provide the karaoke-like service, our implementation considers two threads: one handles the mp3 play out and the other manages the display of the text. Throughout the paper, we refer to the thread T1 as the thread used to play out the mp3 audio file, and to the thread T2 as the thread used to display lyrics according to the timing synchronizations.

In a very basic implementation, the two threads are independent: thread T1 plays out the song and thread T2 displays the text according to its local timer. While being very simple and effective, this solution has some drawbacks: the cpu scheduler may introduce some delay causing a skew time between the two threads and further, user cannot act on the mp3 play out (by pausing, or jumping to another song position) without compromising the karaoke-like service. In fact, since the two threads are independent, if the user acts on thread T1 modifying the audio play out, thread T2 should be modified too, but, since T2 uses its local timer and since this timer is not modified by thread T1, the text is no longer synchronized with the audio play out.

To avoid these problems, thread T2 displays text according to a timer-message received by thread T1. Hence, our player architecture is the one depicted in Fig. 1: Periodically, T1 thread sends its play out time to the other thread, and hence T2 is awakened by thread T1. Upon the reception of the timer message, T2 retrieves the row of text that has to be displayed and displays it on the user's screen. Timing messages are sent within a time unit from each other (i.e., in our implementation every 100 milliseconds).

The benefit of this solution is that the client can act on the mp3 play out (by pausing it or jumping to another position) without causing any problem to the text synchronization. In fact, text is not displayed according to the T2 thread timer, but to the mp3 playout time. Hence, user can browse the mp3 file and the text will be always synchronized with what is sung. This is achieved because thread T1 always sends the current play out time and T2 uses this time to show the correspondent text.

In Fig. 2 we show a detailed description of our implementation. Thread T1 starts playing out the mp3 file and periodically sends a timer-message to thread T2. This thread reads the time-message and displays the text according to the textual information contained in the comment-tag field. This mechanism allows to provide a karaoke-like service on the user's screen. If the user modifies the audio play out (for instance, before time 400 it rewinds back to time 200) the displayed text is still synchronized with the audio play out because T1 always sends the audio play out time to T2).

In Fig. 3 we show a snapshot of our player. A region of the screen is used to display some information about the mp3 song that is played out and another region is used to display lyrics currently sung. The first row shows the words that are currently sung, while the second row shows the words that are going to be sung. When the singer begins singing the sentence of the second row, this row becomes the first one and a new sentence appears in the second row. This goes on until the end of the song; this provides the user with a karaoke-like

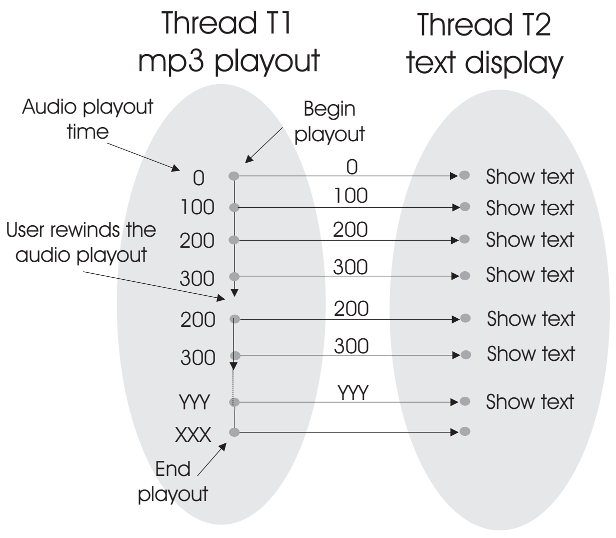


Figure 2: ATS approach: an example.

service.

## 4 Conclusions

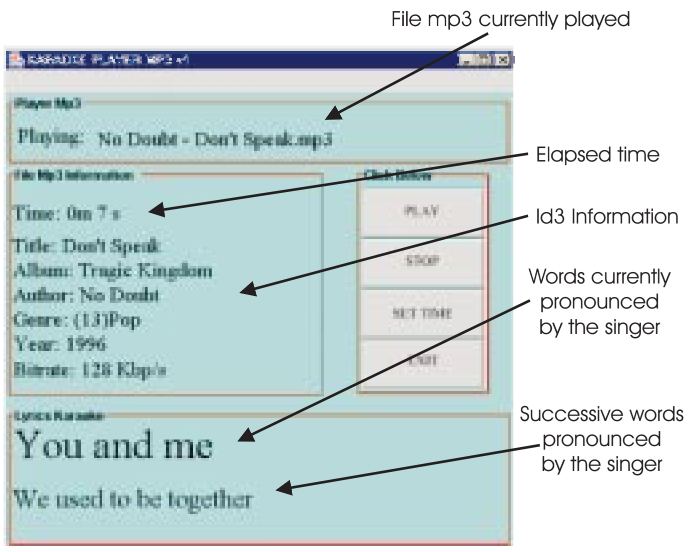
In this paper we proposed a new approach to provide users with a particular multimedia entertainment service: the karaoke service. Our goal is to provide this service with an mp3 file filled in with all the necessary information. Our approach is new in the sense that no additional files (i.e., text files) are required and that the mp3 file is provided with all the necessary timing synchronizations information.

To describe these timing synchronizations between lyrics and audio, we designed a simple description language, based on the SMIL language. Timing information are stored in a transparent way inside the mp3. In this way, the mp3 format is not affected.

Since the karaoke-like service can be provided by players that can manage the timing synchronizations, we develop an mp3 player to prove the effectiveness of our approach.

Since timing information are transparently stored inside the mp3 file, if audio-text timing synchronizations are not present, our player acts as a normal mp3 player.

Our mp3 player has been written in Java language. Java portability along with the simplicity of our approach allow to directly insert our player over any audio device that can support java applications and can provide high quality audio to the user. For example, several vendors are releasing cellulars with mp3 player and java compatibility.


File mp3 currently played
Elapsed time
Id3 Information
Words currently pronounced by the singer
Successive words pronounced by the singer
Player Mp3
File Mp3 Information
Click Button
PLAY
STOP
SET TIME
EXIT
Lyrics Karaoke

Playing: No Doubt - Don't Speak.mp3

Time: 0m 7 s
Title: Don't Speak
Album: Tragic Kingdom
Author: No Doubt
Genre: (13)Pop
Year: 1996
Bitrate: 128 Kbp/s

You and me

We used to be together

Figure 3: The player.

In conclusion, our approach provides a new multimedia service without requiring any additional resource. It is worth noting that our approach is effective not only for entertainment, but also for other purposes, such as speech descriptions, lessons, movie-subtitles, messages, etc.

## References

[1] http://mpeg.telecomitalialab.com/

[2] http://www.umts.org

[3] http://www.wi-fi.org/

[4] http://www.apple.com/music/store/

[5] Karaoke On Line, http://www.thinks.com/karaoke/

[6] Real Networks Inc., http://www.realnetworks.com

[7] Y. Lee, D.H.C. Du, W. Ma, SESAME: A Scalable and ExtenSible Architecture for Multimedia Entertainment, in Proc. IEEE 20th International Computer Software and Applications Conference, Seoul, 1996.

[8] C. Liu, The construction of a multimedia application on public network, in Proc. SPIE High-Speed Networking and Multimedia Computing Conference, 1994.

[9] W.H. Tseng, J.H. Huang, A high performance video server for karaoke system, IEEE Transaction on Consumer Electronics, 40(3), 609-618, August 1994.

[10] M. Roccetti, P. Salomoni, V. Ghini, S. Ferretti, S. Cacciaguerra, Delivering Music over the WIreless Internet: From Song Distribution to Interactive Karaoke on UMTS Devices, Chapter 24, Wireless Internet Handbook, B. Furth, M. Ilyas Editors, CRC Press, 2003.

[11] IDv3 Website. http://www.id3.org/.

[12] W3 Reccomendation, Synchronized Multimedia Integration Language (SMIL) 2.0 Specification, http://www.w3.org/TR/smil20/, 2001.

[13] Real Media, http://www.real.com

[14] FLIPS, http://www.cs.umn.edu/Research/dms/html/
dlearn.html