memoxo - browserbased recording of audio and video messages

Rudolf Hartjes[1], Markus Höckner[2⋆]
and Christine Strauß[1]

[1] University of Vienna, Department of Business Studies
Brünner Straße 72, 1210 Vienna Austria
[2] University of Vienna, Multimedia Information Systems
Liebiggasse 4/3-4, 1010 Vienna Austria
{rudolf.hartjes,markus.hoeckner,
christine.strauss}@univie.ac.at

Abstract. *Memoxo* is a free web-application that allows users to record, send or publish audio and video messages straight from their browser. This paper addresses *memoxo's* usage as well as its technical aspects and application scenarios. *Memoxo* is based on the Red5 media server and satisfies the Web Accessibility Standards, which is likely to provide alternative means of communication for people with disabilities. Furthermore, this paper presents *memoxo's* detailled core functionalities.

1 Introduction

Developing a web-application is not as simple nowadays as it had been a few years ago, due to the fact that characteristics such as extensibility, reusability, flexibility and reliability have gained critical importance. As a consequence, Model/View/Control (MVC) frameworks and such "new" technologies as ORM, I18N or other Web 2.0 techniques have found their way into web-application development. In 2005, new web platforms including YouTube entered the cyberspace and launched a communication revolution. Media content such as audio and video data has been essential to the latest Web 2.0 developments and has motivated users to produce this media content themselves. As a consequence, new protocols, container formats, and codecs for audio and video have been developed, while a substantial increase in the available internet communication bandwidth took place in parallel.

By implication, the demand for developing new means of communication is continuing to increase and has contributed to the launch of a large number of new video streaming web-applications. Most of these focus on Video on Demand (VoD) services, such as online TVs or private video platforms. So far, the process of *capturing* video and audio files has mostly been tied to applications that must be installed on a computer's hard drive. The concept of browser-based media recording is not new in and of itself, as many different social networks or

⋆ and contributors: Michael Bartosch, Lisa-Maria Niehoff

microblogging platforms (e.g., Facebook or Twitter) have already integrated such a service into their online strategy. Nevertheless, these services are closely tied to their provider and their provider's policies on data storage and copyright. For this reason, *memoxo* has been realized as a prototype[3] of a browser based media-recording web-application to independently capture audio and video files online. *Memoxo* is a free web-application that focuses on maximizing its flexibility with regard to the parameters of its purpose or usage.

One of the main goals during the application's development process was to give the user full control and copyright over the self-created media content. Whether the user intends to send the created content as a message to another person, to publish it, to post it, to delete it, or simply to save it as a note to him/herself is in the end a decision to be made at the sole discretion of the user.

Section 2 of this article introduces *memoxo's* core functions, as well as its user interface, while section 3 provides its technical details, such as protocols and streaming technology. Section 4 sketches out two possible (extended) application scenarios, i.e., accessibility and elearning, and section 5 provides a conclusion to this article.

[3] http://www.memoxo.com

2 *Memoxo* core functions

Memoxo's main purpose is to provide online recording of audio and video content and various actions of communicating this content or, respectively, these messages. This core purpose consists of four different types of actions: record, send, store, and publish.

2.1 Record

The process of recording is *memoxo's* most important function in order to capture media content. *Memoxo* offers two different recording options:

- Recording of audio content alone
- Recording of video content (including audio)

This explicit differentiation was made due to two reasons: *(i)* users without a camera should still be able to use *memoxo* to record audio messages, *(ii)* as audio messages require less memory and streaming bandwidth, the audio recorder is especially suitable for users with low bandwidth. Both the audio and audio/video recorder involve the same four control mechanisms:

- Start recording
- Pause recording
- Resume recording
- Play recorded data

The actions of "pause recording" and "resume recording" were applied in order to give the user time for both thinking and speaking while recording a message by allowing him/her to interrupt and resume the recording process at any time. This functionality is likely to benefit both the file size of the recorded content and the quality of the message due to the fact that it helps to avoid idle recording times. All four control mechanisms can also be operated by keyboard rather than using the GUI (Fig. 1), which enables for example blind users to record messages by means of their preferred input device. Once a user has finished the recording process, he/she is provided with several options for sending, publishing or saving the recorded content, as described in the following paragraphs.

2.2 Send

In order to send the recorded media content, a user must perform several mandatory tasks such as filling in the email address of the intended recipient and providing a subject line for the message. To enhance the process of defining a recipient, *memoxo* offers an address book in which the user can save his/her contacts and add them to the recipient list with a single mouse click (or keystroke). Furthermore, the address book doesn't have to be populated by manual entries, as a user may simply import contacts from other email or communication providers such as Gmail, Hotmail or Yahoo. In addition to specifying a recipient and the

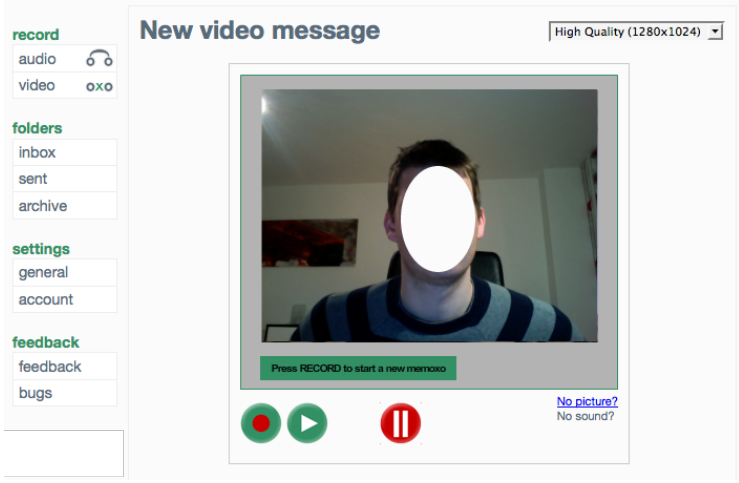

Fig. 1. graphical user interface of *memoxo's* audio/video recorder

subject line, a user can add additional data to a message, such as attached files or a text string. Once the user has entered the required fields, the message can be sent, which means that the recipient will receive an email containing the sender's email address, the subject line, optional data (text-message, file attachments) and a text-link. This link points to the recorded message content on *memoxo's* media server (described in section 3). After a message is sent, its creator can convert the recorded media data into any desired format and download the message during a predefined time window or actively delete the message.

2.3 Store

If a user decides to neither send nor publish, but rather to store the recorded content, he/she is able to do so without entering a recipient, but the user must nevertheless provide a subject line. In this case, the subject line merely serves as a descriptor for the recorded content, which will be saved in the user's personal message archive. An archived message can also be sent or published at a later point of time, just as a sent message can subsequently be saved to the user's archive.

2.4 Publish

Another way of communicating a recorded message apart from sending it to one or more recipients lies in publishing the content. In order to do so, the user (who *must* be the creator of the message) can "unlock" the recorded media in order to receive a snippet of HTML code (embedding code). This code can be pasted into any website, blog, forum, eLearning platform or social network. Due to the

fact that published videos have a tendency to spread and circulate over time, *memoxo* also provides its users with the possibility of "locking" the recorded media again at a later point in time. Doing so renders any embedded instance useless, as the associated media file is no longer publicly accessible.

3 Conceptual model of *Memoxo*

3.1 *Memoxo* architecture

The concept of such a web-application is simple and clear as figure 2 shows.

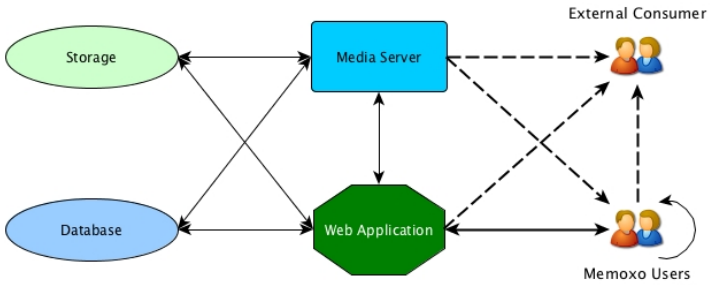

Fig. 2. conceptual model

On the one hand, the model consists of the "*memoxo* user", who is capable of creating and receiving audio and video messages. It does not matter whether this user sends messages internally to *memoxo* registered users or to the "outside world" to non-registered users. A random security code is needed in order to guarantee that only the intended receiver is actually able to access the message. On the other hand, the model also incorporates non-registered users, who are not in a position to create audio or video messages using *memoxo*. However, this group can still receive messages, as *memoxo's* web-application sends the recipient a notification email that contains a link to the web-application, which in turn automatically starts the message. If the message's creator publishes the message (e.g., in a forum), the message can also be received and can be "read" in the Web. In this scenario, the recipient does not connect to the web-application itself, but rather directly to *memoxo's* media server. To perform these actions, the web-application has to be connected to the media server in order to send and receive A/V data. The message itself is stored on a storage platform, while a database is needed for authentication purposes and also to store other information such as facts about the creator, the message type (audio or video), the user's address book, and all other internal data for the web-application. Providing a clear and structured implementation for this storage is a major challenge, as otherwise things could become complicated and the borders between the media server and web-application could suffer, leading to the loss of a great deal of flexibility and extensibility.

3.2 Technical issues of the Web Frontend

Because the Web frontend is what the user sees, its layout is very important; this matter will be discussed later. In the following section, we focus instead on the technical issues involved. The choice of which specific programming language to use is very difficult these days, as each language brings with it certain advantages and disadvantages. The need to use a framework for implementing the project in particular proved to be a stumbling block. Consequently, we tested a number of frameworks, including Seagull (PHP), Zend (PHP), Jifty (Perl), Catalyst (Perl), and Ruby on Rails, as well as others. The important requirements were:

1. Extensibility
2. Reusability
3. Flexibility
4. Reliability
5. Performance
6. Available plugins
7. Available modules

Most of the frameworks we tested have in common that they are MVC (Model View Control) frameworks, which ensures that handling is both clear and well structured: the model for processing the data, the controller to prepare it and the view to present it. As the use of HTML templates or ORM (Object Record Mapper) to communicate with the database is common among the languages, there were no really major differences between them. In the end, we selected the perl framework Catalyst [1] mainly due to the fact that:

- Perl is a very powerful programming language,
- A lot of plugins are available for it,
- A huge repertoire of modules [2] exist, and
- The web-application can be deployed for almost every commonly-used operating system.

In particular, the possibility of using "Template Toolkit" [3] for the view provided a main reason for using Catalyst. "Template Toolkit" is a powerful template engine written in Perl and C that, for example, makes it possible to produce HTML code with a very high level of reusability. Also, the possibility of using different directives, virtual methods, filters and plugins makes "Template Toolkit" a very powerful engine and its use as the View in a "Catalyst Web Application" is even recommended [4]. Given that the aspect of preparing the HTML code for multilingual use is very interesting, it is worth noting that preparing the web-application for more than one language is very simple in combination with the Catalyst plugin "I18N". To prevent umlaut bugs and to prepare the entire web-application for non-Latin characters, the complete HTML code was written in UTF-8.

The design of a database as the main data store for such information as user accounts and the save a record feature was also of crucial importance.

A MySQL database capable of storing UTF-8 data was created. Such UTF-8 conformance was a major prerequisite for the database, due to the need to provide a multilingual frontend and to prevent conversion. For the ORM, we applied the Perl package DBIx, which is supported by Catalyst and ensures the possibility of changing the database system in the future. Because not every part of the web-application should be accessible by anonymous users, *memoxo* users must login in order to use the service. Session keys are used to identify logged-in users, with the individual keys expiring after a certain time in order to prevent abuse. The key and the session-based data (e.g., preferences) are stored in the database, with Catalyst's "Session Plugin" used for this purpose. The last step in designing and programming the frontend involved deploying the web-application at a web server, for which Catalyst offers several different methods [5]. Currently, *memoxo* runs as a mod_perl application in combination with an Apache web server. The major disadvantage of this approach may lie in the fact that all packages and modules are loaded in the local memory: a full server restart is required if something is changed in the source code or a new package has been installed. Because mod_perl is also very memory-thirsty, running Catalyst as a FastCGI application represents the next logical step. The static content of the web-application is managed by the cache of the Apache web server. To prevent the rendering of the templates for every request, they are also available in some sort of cache.

3.3 Streaming

As *memoxo's* main purpose is to provide the browser-based recording of audio and video, a simple web-application is solely a container for *memoxo's* main functions.

Most projects in the Web use external applications to do the work for them; for example, the VLC player is very well-known for streaming applications. But for our scenario, only a simple browser should do the work, without the need to rely on any third-party applications.

The client As browsers themselves are not capable of recording videos or sound from the peripheral devices of a computer or laptop, a method is needed for grabbing the signals from the microphone or webcam and sending them to the *memoxo* media server. One possible way to do so lies in using Adobe Flash. Although one consequence of using Adobe Flash is that every *memoxo* user would need to install the "Adobe Flash Plugin", this is not really an issue, as most Web users already have installed it on their computers. After all, otherwise they would not be able to watch YouTube videos or play Flash-based games. However, the issue of how to transfer the audio and video signals to the server does represent a challenge. Adobe Flash permits the writing of "Action Scripts" that enable work within the Flash-application, such as manipulating the layout, creating buttons, or interacting with the user. Luckily, Flash also provides possibilities to use a computer's audio and video peripheral devices. Actually there are two

possible ways of writing Action Scripts: Action Script version 2.0 and the newer version 3.0, which unlike its predecessor is fully object-orientated and allows the programmer to choose from a richer repertoire of available Action Script methods.

Consequently, we were able to capture the signals from the peripheral devices and send them in real-time to the media server. While establishing a connection to a media server is also no big challenge, transporting the message in real time remained a challenge.

The HTTP protocol was not designed for such work and other protocols such as SMTP and FTP were also not viable options. Instead, the solution was to use RTMP in combination with Flash [6].

RTMP (Real Time Messaging Protocol) is a proprietary network protocol developed by Adobe Systems that, with the help of Adobe Flash, makes it possible to transport audio and video signals over the Internet to a media server or vice versa. This protocol uses port 1935 for communication, which sometimes presents a problem because of the firewall configurations on a Web Server. RTMPT, which is based on the HTTP protocol, was developed to override this firewall problem. As it is often also important to provide secure connections, the HTTPS-based RTMPS was also developed. These three protocols make it possible to send audio and video data from a client to a server according to Adobe Flash. Adobe Systems launched "Flex" to reflect the fact that many software developers wanted to implement Rich Internet Applications (RIA) but did not like to work with Adobe Flash due to a wide range of differing opinions and prejudices. The client still needs the Adobe Flash Player to use the application, because Flex's output is also a Flash application.

Instead, the big difference with Flex lies in its implementation. In contrast to Flash, Flex's user interface is not "painted", but rather written in MXML, which is a XML-based, declarative language that allows the creation of a User Interface in a different way. The Flex-compiler translates the MXML to Action Script code and the result is a Flash application. The fact that Flex is almost open source and the way in which Flash RIA are built have enourmously raised its acceptance among programmers.

The server As mentioned previously, the media server has to "understand" the RTMP stream. Although an Apache web server is a media server, it is not built for this type of work.

There are three well-known servers that matched our needs:

1. Adobe Flash Media Interactive Server (FMIS)
2. WOWZA Media Server
3. Red5

The Adobe FMIS is cost intensive. However, it supports a large number of codecs, is very powerful and makes implementing an application very easy. In addition, the fact that Flash is a product developed by Adobe Systems helps significantly in developing as it ensures compatibility. The WOWZA Media Server is

also a commercial product, but it is not as expensive as Adobe FMIS. WOWZA Media Server, which garnered a lot of international prizes in 2009, also supports a huge amount of different codecs such as H.264 and is often used in VoD (Video on Demand) projects. In contrast to the two preceding media servers, Red5 is an open source product; its current version is 0.9. Red5 is often used in non-commercial projects because of the fact that it is free and supports a large number of codecs for publishing [7]. *Memoxo's* characteristic as a non-profit project fostered our decision to use Red5 as a media server. Learning how Red5 works costs some time, as users must implement the application for recording on their own; only the server's core is delivered. Nevertheless, users are able to adapt the application to their specific needs. The Red5 application is written in Java and uses the core's specific classes to do the work.

- Listen to port 1935.
- If a stream starts, lookup in the database and ascertain whether the user is allowed to record.
- If the user is allowed, create a FLV file and put the stream into it.
- When the user stops streaming, stop writing and mark it in the database as "recorded".

This is one of the main ways in which *memoxo* and other web-applications work [8].

The documentation in Red5 is rather short, but a very powerful community supporting the development of such projects exists. The community also created some demo applications that supported the implementation of an application such as *memoxo*. Configuring the server itself was straight-forward and starting it was not such a big problem. But Red5 requires a lot of memory and CPU power. Since Red5 supports multi-threading, the CPU power is not such an eminent issue, but the memory issue still remains, unless there is sufficient bandwidth to handle the enormous traffic. At least one to two gigabytes of memory is required to run a Red5 media server [9], otherwise the server starts to swap and failures in encoding or decoding could ensue. *Memoxo's* major duty is to record audio and video files. Currently, Red5 only supports recording FLV files, which is a container format developed by Adobe Systems and has became famous through platforms like YouTube [10]. Recording a video was no problem in previous times when analogue media was used: put a cassette into a camcorder and press record. But in the digital era, recording is not as simple as it may at first appear, due to the fact that audio and video offer different codecs and choosing the "right" one is not a simple task. In fact, Red5 supports only two different codecs for audio recording: Speex and Nellymoser. Speex [11] is a very fast codec that has an extremely low bandwidth usage and is open source. Accordingly, Speex codec is very often used in mobile or VOIP applications [12]. This codec's disadvantage lies in the fact that it currently only supports 5kHz recordings in combination with Red5. While this is enough for recording the human voice, this codec is not the best choice for music. Hopefully, this limitation will soon be fixed. In contrast, Nellymoser is a closed codec (developed by Nellymoser Inc.) that serves

as the default codec for FLV. However, the fact that Nellymoser is a closed codec is a major drawback, as it impossible to convert the FLV into another container format. Due the fact that Red5 is an open source project, some major bugs occur from time to time. However, the response to such bugs is quite fast and the development team of Red5 tries to fix them as quickly as possible. Moreover, users do not have to wait for the next release of Red5 when a bug is fixed; instead, they can check out the latest version of the source code from their SVN (Subversion) repository and build the server on your own, which helps a lot in developing. To conclude, Red5 is a very powerful media server that makes it possible to implement almost every application that a media server needs.

In the following we describe two extended application scenarios in which *memoxo's* characteristics, as described in section 3, may provide enhancements.

4 Extended Application Scenarios

Given its non-restrictive nature regarding the purposes of a user's audio or video recording, the range of application scenarios for *memoxo* is particularly widespread. The following scenarios describe further capabilities of *memoxo*, based on its core-purpose to record, send, publish and store audio and video messages.

4.1 Web Accessibility

The Web enables persons and organizations around the world to retrieve and provide information by various means of interaction. The basic principle of Web Accessibility is to include the whole range of internet users, whether they experience some sort of disability or not. The range of impairments that are likely to prevent a person from communicating via the Internet encompasses numerous disabilities, such as visual impairments, cognitive disabilities or motor deficiencies. The W3C's Web Accessibility Initiative therefore defines Web Accessibility as a state in which

> people with disabilities can perceive, understand, navigate, and interact with the Web, and can also contribute to the Web. [13]

To efficiently use the Internet, people with disabilities are dependent on the use of assistive technologies or devices (i.e. refreshable braille displays, custom keyboards, screen readers or eye gaze systems) in order to compensate their particular kind of impairment. Nevertheless, when it comes to message interaction, the concept of written text dominates and therefore urges people to use conventional input devices such as a keyboard. Although a manual translation from spoken language into written text is imaginable, it is just another interim stage from spoken word to text. Therefore, *memoxo's* simple mechanism to record, send and archive audio messages straight from a user's browser allows people with visual or motor disabilities to avoid the exhausting use of textual input devices and to stay within their preferred form of communication. Furthermore, portals that are accessible in the same manner as *memoxo* are believed to attract a larger number of users [14]. Due to *memoxo's* simple navigation and control mechanisms, blind users would be in the position to record and send messages by the use of a few keystrokes rather than typing the whole message, as memoxo's recoding device is fully operable by keyboard. A short how-to is provided within the audio/video recorder in order to introduce disabled users to *memoxo's* functionalities. Due to the use of Cascading Style Sheets (CSS), this introduction can be made invisible for "regular" users, but still be visible to assistive devices such as screen-readers. In addition, deaf users too could benefit from using *memoxo*, as it provides a solid base for the (video) transmission of sign language.

4.2 eLearning

On eLearning platforms, students' success is based on various aspects, one being quality feedback from their tutors. Especially with regard to assignments that

include large visual components, textual feedback is often found to be incomplete [15]. Furthermore, studies on the quality of video-feedback in eLearning situations have shown that video-feedback is experienced as a valuable and personalized method by both tutors and students [16]. Therefore, *memoxo* (integrated in an eLearning system) could enable the tutor to provide quality video feedback in which parts of a student's visual work can be commented or corrected by what needs to be improved, rather than describing the location and kind of mistake by typing an email. Additionally, *memoxo* could allow a tutor not just to give feedback to a single student, but also to address a whole class of students by publishing recorded assignments or feedback within the eLearning system.

5 Conclusion

As a free browser-based web service, *memoxo* allows its users to record, send or publish audio and video content. *Memoxo's* main character is defined by its user-centered approach, which means that the users have full control and rights with regard to the media content they record. The *memoxo* project does not aspire ownership or long time storage of user-created data; instead, its designated objective is to provide users unlimited control and unrestricted rights regarding their recorded media content. Based on accessibility standards and Red5, *memoxo* uses the latest technology in order to provide a fast, reliable and easy-to-apply user experience. As initial implementations show, *memoxo's* purpose is defined by the users' intentions on what is to be recorded and how it is communicated. Also, we believe that the fact that *memoxo* is an independent service that does not seek any gain from recorded messages and is not tied to any particular media platform makes *memoxo* a trustworthy portal for Internet users to record and distribute original digital media content.

References

1. "Catalyst web framework." `http://www.catalystframework.org/`, 2010.
2. "CPAN search." `http://search.cpan.org/`, 2010.
3. "Template toolkit." `http://template-toolkit.org/`, 2010.
4. J. Rockway, *Catalyst*. Packt Publishing, 1 ed., 2007.
5. K. Diement and M. S. Trout, *The Definitive Guide to Catalyst: Writing Extensible, Scalable and Maintainable Perl-Based Web Applications*. Apress, 1 ed., 2009.
6. A. Shionozaki, "Integrating routing and resource reservation mechanisms in real-time multicast protocols," in *ICDCS '96: Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS '96)*, (Washington, DC, USA), p. 141, IEEE Computer Society, 1996.
7. "Red 5." `http://www.red5.org/`, 2009.
8. J. Bross, J. Oppermann, and C. Meinel, "Enabling video-blogging without relying on external service-providers," *Computational Science and Engineering*, vol. 4, pp. 515–522, 2009.
9. S. Sukaridhoto, N. Funabiki, T. Nakanishi, and D. Pramadihanto, "A comparative study of open source softwares for virtualization with streaming server applications," in *Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on*, pp. 577–581, May 2009.
10. P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, (New York, NY, USA), pp. 15–28, ACM, 2007.
11. T. xiph open source community, "Speex." `http://www.speex.org/`, 2009.
12. B. R. Chang, C.-P. Young, H. F. Tsai, and R.-Y. Fang, "Embedded system for inter-vehicle heterogeneous wireless-based real-time multimedia streaming and video/voice over ip," *Fourth International Conference on Innovative Computing, Information and Control*, pp. 365–368, 2009.
13. W3C-WAI, "Introduction to web accessibility." `http://www.w3.org/WAI/intro/accessibility.php`, 2010.
14. M.-L. Leitner, R. Hartjes, and C. Strauss, "Web accessibility issues for the distributed and interworked enterprise portals," in *ICPPW '09: Proceedings of the 2009 International Conference on Parallel Processing Workshops*, (Washington, DC, USA), pp. 270–275, IEEE Computer Society, 2009.
15. A. Inglis, "Video email: a method of speeding up assignment feedback for visual arts subjects in distance education," *British Journal of Educational Technology*, vol. 29, no. 4, pp. 343 – 354, 1998.
16. S. Hase and H. Saenger, "Videomail — a personalised approach to providing feedback on assessment to distance learners," *Distance Education*, vol. 18, no. 2, pp. 361–368, 1997.