MusicBox: creating a musical space with mobile devices

Jan-Torsten Milde
Fulda University of Applied Sciences
milde@hs-fulda.de

ABSTRACT

This paper describes the ongoing development of a system for the creation of a distributed musical space: the MusicBox. The MusicBox has been realized as an open access point for mobile devices. It provides a musical web application enabling the musician to distribute audio events onto the connected mobile devices and control synchronous playback of these events.

In order to locate the mobile devices, a microphone array has been developed, allowing to automatically identify sound direction of the connected mobile devices. This makes it possible to control the position of audio events in the musical space.

The system has been implemented on a Raspberry Pi, making it very cheap and robust. No network access is needed to run the MusicBox, turning it into a versatile tool to setup interactive distributed music installations.

1. INTRODUCTION

Modern web technology can be used to create highly interactive, visually appealing, collaborative creative applications. With the development of the Web Audio API, a promising basis for the creation of distributed audio application has been made available. A number of interesting projects have shown some progress in bringing musical application into the web browser: Flocking ([1]) defines a framework for declarative music making, Gibber allows for live coding in the browser ([2]) and Roberts et.al. ([3]) show how the web browser could be used as basis for developing synthesizers with innovative interfaces.

In our approach, we would like to use the web technology to create musical spaces, thus distributing the sound creation onto a number of small mobile devices placed in a large room. These devices should be synchronized in time and should be controlled by a central musical system: the MusicBox.

2. MUSICBOX: CREATING AN OPEN MUSICAL SPACE

The development and construction of the MusicBox has been driven by the idea to create a music system, which supports the musician to easily define and setup a distributed musical space. The underlying concept is based on a client / server approach, where standard mobile systems (aka smart phones) are used to perform the sound synthesis, or simply the playback of pre recorded sound files. The musician is defining a digital orchestra built from mobile devices. The computing power of standard mobile devices has increased significantly during the last few years, making them feasible for sound reproduction. A limiting factor are the inbuilt speakers, though. In order to perform the synchronized audio playback on the devices, web technologies are used. More specifically a web application using the Web Audio API ([4]) has been implemented.

Figure 1. The MusicBox runs on a Raspberry Pi B. The system is configured to work as an open access point. Mobile devices connect to the MusicBox and start the web application with their browser. Once connected, the timing between the mobile device and the MusicBox is synchronized. The mobile device is then part of the musical space.

3. TECHNICAL SETUP OF THE MUSICBOX

The setup of the MusicBox on the Raspberry PI took 5 steps:

- installation of the standard operating system (Rasperian)
- configuration of access point software
- installation of node.js
- development of the music system as a web application in node.js and express
- integration of the microphone array (via USB connected Arduino).

The first step was to install and setup the standard operating system onto the Raspberry Pi. Rasperian is a Debian based Linux clone compiled for the underlying hardware of the Raspberry. In order to make the system more user friendly, we configured the Raspberry to behave like an

open access point. A pre compiled version of the *hostap*-demon was installed on the system. This version matched the used WLAN-dongle. Once the access point demon was running, connecting to the box was very simple. In order to make things even simpler, we decided to install *dnsmasq*. With this service symbolic names could be used for identifying the MusicBox. In our case we chose *http://musicbox.fun* as the address of the system. This even works without any connection to the internet. The MusicBox thus provides an independent network, which can be used as a basis for musical installations even in very unusual places.

The musical web application has been implemented in node.js. Node.js is targeting the development of modern web applications. It is well suited for the implementation of JSON-based REST services, supports data streaming and the development of real-time web applications. As such it is matching the requirements of distributed musical application very well. On the other hand, it is not powerful enough for heavy real time computation tasks. So real time audio processing should rather be implemented in other frameworks.

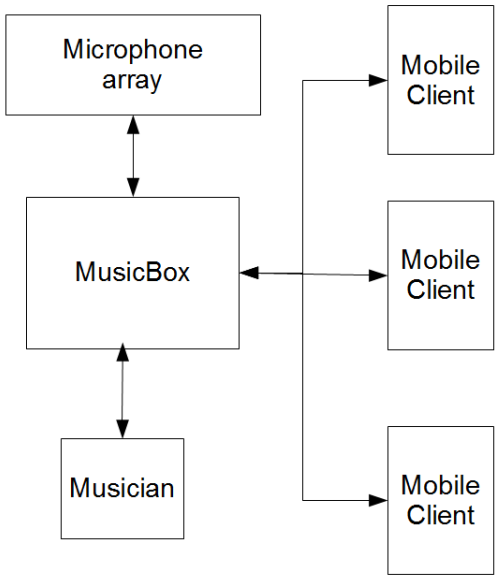

Figure 2. The system architecture of the MusicBox. The web application is synchronizing the mobile devices. It provides a control interface to the musician, that allows to transmit audio data and control data.

The final configuration of the MusicBox prototype has been stored as an iso-image. This makes it very easy to create a running copy of the system, even for beginners with no technical background. Simply create a copy of the iso-image on a standard micro sd card and insert the card into the Raspberry Pi. If the WLAN-hardware matches the standard configuration, the system will be up and running in less then a minute.

4. SYNCHRONIZING DEVICES

An essential pre condition for the implementation of a distributed musical environment is establishing a precice time management for the underlying web application. The basic timing of server (MusicBox) and clients (mobile devices) has to be tightly synchronized. Without this synchronization many musical applications like synthesizers, sequencers, drum machines, loop boxes or audio samplers will not work as expected. Audio events need to be synchronously scheduled in order to create a coordinated playback in an distributed environment.

In order to establish this synchronisation, one could be tempted to use the wireless connection to provide dynamic synchronisation messages. This turns out to be not very reliable. The IP protocol stack is not well suited for real time application communication. It cannot be guaranteed that an IP package is arriving in time.

As a consequence we have chosen to rely on the internal clocks of the mobile devices. This simplifies the synchronisation task and at the same time reduces the networks traffic of the running application.

4.1 Setting the timing offset

Once connected, the mobile devices are synchronizing their system time with the system time of the central MusicBox. The temporal difference between the two timers has to be calculated as precise as possible in order to achieve a high timing correspondence for the nodes of the distributed system.

The synchronization of the system times is a two step process. At first a simple AJAX request is executed upon loading the application (an html page) onto the client. The requests connects to a central web service on the MusicBox and transmits it's system time. The standard resolution is milli seconds.

In a second step a WebSocket connection is established. This connection is used to further refine the adjustment of the time difference between server and client. Our approach follows the solution described in [5]. We adopted the given implementation to the technology used in our setup.

All in all the taken synchronization method results in a quite high precision of the timing adjustments. In our experiments the first phase calculates a temporal difference in the timers of only 30ms. The faster second phase is able to decrease this difference to below 4ms.

As a consequence the deviation of the timing in the complete cluster is approximately 8ms. Starting and stopping audio events on the mobile devices is therefore comparatively synchronous and below the general perceptual threshold.

4.2 Web Audio

The Web Audio API *AudioContext* allows to access the current time (the audio clock) using the *currentTime* property. This property is calculated by accessing the system hardware and returns an increasing double value denoting the number of seconds since the *AudioContext* has been established. The internal resolution of a double value is sufficient to facilitate a very precise timing for the audio events even over a longer period of time. Within the Web Audio API a larger number of functions are controlled by the audio clock. As such it becomes possible to precisely control the timing of the audio events with this property.

For the current version of the MusicBox we rely on this relatively simple form of timing control. It is sufficient to achieve a synchronous start of audio playback across the mobile devices of the musical space.

A clear drawback of this approach is the strict fixation of the timing. Once set, it is not possible to dynamically adjust audio parameters with this simple timing approach. More elaborate timing control for WebAudio applications have been discussed by Wilson ([6]) or Schnell et. al. ([7]).

5. MICROPHONE ARRAY

As soon as the mobile devices have been registered and synchronized, the audio playback can be started. At this point in time no further information about the spacial placement of the mobile devices is available. One could think about using the GPS sensor to request the current position. Unfortunately the precision of this sensor is not very high and quite often the sensor does not work inside buildings.

As a consequence, we developed a simple microphone array to at least achieve a rough estimation of the relative position of the mobile devices. This microphone array is able to detect the relative volume of a mobile device and assigns this to a sector in space. The microphone array consists of 6 identical microphones with a pre-amplifier, that are arranged in a semi circle of 180 degrees with each microphone being responsible for a sector of 30 degrees. It can be mounted on a standard tripod.

The analog to digital conversion is performed by an Arduino, that is connected to MusicBox via USB. The interpretation of the measurements is implemented as part of the web application. Despite of the fact, that the microphone array is definitely a low cost, low tec solution, the resulting sector assignment works surprisingly good.

6. WEB APPLICATION WITH NODE.JS

The musical web application was developed using Node.js. The following services have been implemented in the web application:

- for the initial synchronization a *web service* has been implemented providing the current system time in ms
- a *web socket* based connection to the client is established for sending timing information and control data
- a *visualization* show the attached clients and also displays status information of the clients
- *transmission of audio data* to the clients
- *transmission of control data* as part of the play back control

The web application is split up into two principle application parts: the client part for the mobile devices and the control part for the musician.

The graphical user interface for the client part is kept very simple. It consist of a single colored area displaying the current system (client) status using a traffic light metaphor. The area lights up in green when the client is synchronized and ready to play back audio content. If the area is yellow, audio data is uploaded to the client. The client is in waiting state. And if it lights up in red, then synchronization has failed. The client is not operational.

The control part for the musician is realized by a separate user interface. Once the clients are synchronized, the musician is able to send commands and audio data.

The play back control is based on the visualization. It is comparable to a traditional score notation or the instrumental track display that can be found in most of the current digital audio workstations. The audio elements are assigned to the clients via drag 'n drop (alternatively by a double click). Each audio element has flexible but fixed duration, that is determined by the underlying temporal grid. The musician places the audio element at it's intended temporal position, while the actual play back of an audio event is initiated by sending the starting time to the client.

6.1 Recording sound on the MusicBox

In order to make the MusicBox more flexible and better suited for live performances, we added a USB audio interface to the system. This allows to record audio synchronously with the ongoing performance. The MusicBox thus works like simple loop station. In addition, a set of effects has been implemented (*delay, rever, filter etc.*).

The recorded audio data can then be streamed to mobile devices and then be integrated into the ongoing playback. The musician is therefore able to create audio events on the fly, distribute them to the mobile devices and integrate these audio events into the musical space.

7. CONCLUSIONS

The MusicBox provides a simple, yet robust and flexible environment to easily create a distributed musical space using standard mobile devices. It synchronizes audio playback across the cluster and simplifies the spacial positioning of the connected audio clients.

The current setup is a good basis for further investigations into the creation of musical spaces.

Until now, the clients are passive. While it is possible for multiple musicians to interact with MusicBox simultaneously, no human interaction can be executed by the clients. It would be interesting to extend the functionality of the mobile device into this direction.

While we are currently using web technology, the general approach is not limited to this approach. The MusicBox could quite as well serve native applications on the various devices. These could be more powerful audio applications like software synthesizers that could considerably expand the expressiveness of the musical space.

Acknowledgments

8. REFERENCES

[1] C. Clark and A. Tindale, "Flocking: a framework for declarative music-making on the Web," in *Proceedings of the 2014 International Computer Music Conference*, Athen, 2014.

[2] J. K.-M. C. Roberts, "Gibber: Live Coding Audio in the Browser," in *Proceedings of the 2012 International Computer Music Conference*, 2012.

[3] C. Roberts, G. Wakefield, and M. Wright, "The Web Browser as Synthesizer and Interface," in *New Interfaces for Musical Expression conference*, 2013.

[4] C. W. P. Adenot and C. Rogers, "Web Audio API," in *http://webaudio.github.io/web-audio-api/*, 2016.

[5] N. Sardo, "GoTime," in *https://github.com/nicksardo/GoTime*, 2014.

[6] C. Wilson, "A Tale of Two Clocks - Scheduling Web Audio with Precision," in *http://www.html5rocks.com/en/tutorials/audio/scheduling/*, 2013.

[7] N. Schnell, V. Saiz, K. Barkati, and S. Goldszmidt, "Of Time Engines and Masters: An API for Scheduling and Synchronizing the Generation and Playback of Event Sequences and Media Streams for the Web Audio API," in *Proceedings of the 1st annual Web Audio Conference*, 2015.