A Simple Architecture for Server-based (Indoor) Audio Walks

Thomas Resch
Research and Development, University of Music Basel
University of Applied Sciences Northwestern
Switzerland
admin@noteformax.net

Matthias Krebs
Institute of Mobile and Distributed Systems
University of Applied Sciences Northwestern
Switzerland
matthias.krebs@fhnw.ch

ABSTRACT
This paper proposes a simple architecture for creating (indoor) audio walks by using a server running Max/MSP together with the external object fhnw.audiowalk.state and smartphone clients running either under Android or iOS using LibPd. Server and smartphone clients communicate over WLAN by exchanging OSC messages. Server and client have been designed in a way that allows artists with only little programming skills to create position-based audio walks.

Keywords
Max/MSP, audio walk, audio guide, state machine, game engine, LibPd, Pd

1. INTRODUCTION
The server is based on the fhnw.audiowalk.state object, a programmable state machine providing functionality for simple gaming applications. It is configured by using an easy scripting language, which allows the creation of any number of virtual clients, state transitions and messages bound to each transition. The server sends and receives messages to/from smartphone clients via wireless networks using the Open Sound Control (OSC) protocol. The smartphone client passes those messages directly to LibPd, which is responsible for audio playback and optional data processing. The first application implementing the fhnw.audiowalk.state server and client was the indoor audio game/walk *LautLots* [1], which had its debut in September 2013 at the German Railway Station in Basel, Switzerland.

2. RELATED WORK
While the position tracking system is not part of this paper, please refer to [1] and [2] in order to learn about the solutions used in the *LautLots* project. For other publications about position tracking systems, which could be used in combination with the described server and client, please refer to [3] and [4] and references mentioned in those papers. Examples of commercial solutions for creating (mainly) GPS-based (outdoor) audio walks are given in [5], [6] and [7].

3. NETWORK AND COMMUNICATION
An audio walk consists of one or more localities or rooms, where each locality has a unique ID and is identified by its own WLAN SSID. Each client has a unique ID as well.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
NIME'14, June 30 – July 03, 2014, Goldsmiths, University of London, UK.
Copyright remains with the author(s).

All communication between Max/MSP and the mobile clients is done using the OSC protocol. OSC is a very simple messaging protocol that usually relies on UDP. Each OSC message includes an address path, a type tag, a list of arguments of different data types and an optional time tag.
The communication schema we use is shown in Figure 1.

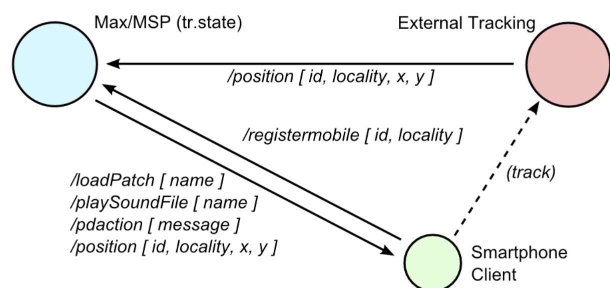

Figure 1. The Communication Schema

The server should be configured with a static IP address, which is known to the clients. After startup, as soon as the client connects to a locality's WLAN, the */registermobile* message is sent to the server. Once a client has successfully registered itself, the server knows its corresponding IP address and OSC port. Subsequently, messages are routed from the virtual client defined in fhnw.audiowalk.state to the actual smartphone client. Whenever the client moves to a different WLAN, it registers itself again, announcing that it has moved on to another locality.
If an external position tracking system is used, it sends */position* messages to the server, notifying each client's position. This enables fhnw.audiowalk.state to deliver events to the client according to its position. If desired, */position* messages can also be forwarded to the client.
Events delivered by the fhnw.audiowalk.state server are usually passed on to Pd. The */loadPatch* message orders the client to load a given Pd patch stored on the smartphone. The */pdaction* message carries a generic Pd symbol whose function depends on how the Pd patch interprets it. The */playSoundFile* message makes Pd play a given audio file, which is stored on the smartphone.
An example from the Pd patch is shown in Figure 2.

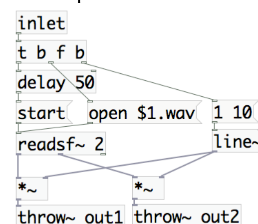

Figure 2. Pd sub-patch for the /playSoundFile message

4. SERVER

4.1 Requirements and setup

The server requires the software Max/MSP and the external object fhnw.audiowalk.state, available at [8]. The minimal setup within Max/MSP is shown in Figure 3.

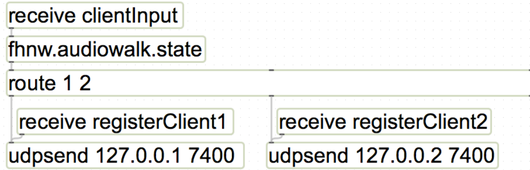

Figure 3. Minimal setup within Max/MSP

4.2 Usage

4.2.1 Creating clients and state transitions

Clients are created by sending the message newClient [clientName] to fhnw.audiowalk.state, where [clientName] can either be a symbol or an integer.

newClient client1

Every client has several instance variables, most important of all *currentState* and *currentPosition*. Clients also keep track of their *lastState*, *lastPosition*, *orientation*, *lastOrientation*, *orientationChanged* and *rotation*.
Transitions are created by sending the message
nextState [conditions][flags][messageReceiver]
[message][wait] to fhnw.audiowalk.state:

nextState 1 currentState 0 position 1. 1. 1. 1. sender message "play X"

In spoken words this means: If any client is currently in state 0 and arrives at position x = 1., y = 1., width = 1. and height = 1, the message "play X" is send to the corresponding client. This is achieved by using the keyword *sender* as message receiver, as an alternative one could use the keyword *all* or specify a client name.

4.2.2 Controlling clients

The message *position [clientName][x][y]* sets the current position. *orientation [clientName][orientation]* sets the current orientation:

position client1 1.5 10.
orientation client1 180.

The message *process [clientName][anything]* allows to send gestures or other kinds of interaction to fhnw.audiowalk.state. By default "yes", "no", "comply" and "noComply" are implemented and are counted in corresponding variables which can be used as a condition:

nextState 6 numberOfNo 5 sender message "5 times no!"
process client1 no

Alternatively it is possible to create a transition condition with an arbitrary string:

nextState 100 anyStringYouLike
process client1 anyStringYouLike

4.2.3 Timing conditions

Using the conditions *maxOveralltime* and *maxTime*, a time limit can be set either for a certain state or a global timeout independent of a state.

nextState 100 currentState 50 maxTime 20. sender message "play Y" wait 10.

In this example a client will automatically move on from state 50 to state 100 after 20 seconds. The *wait* message prohibits a client from moving on to the next state for a certain amount of time, even if the client fulfills the conditions for a transition.

nextState 1000 maxOverallTime 600. sender message "play Y"

Here, all clients will move on to state 1000 after 600 seconds, independent from their current state.

4.2.4 Flags

While the example below could be realized more easily without using any flags, simply by giving a different *nextState* argument for the *yes* and *no* conditions, it demonstrates their usage:

nextState 11 currentState 10 no
nextState 11 currentState 10 yes setFlag 1
nextState 12 currentState 11 flag 1
nextState 13 currentState 11 !flag1
nextState -1 currentState 12 flag 1 unsetFlag 1

The construct *nextState -1* will not result in any state transition and might create a deadlock. It should always be bound to a flag condition, which at the same time is then set or unset so it can happen only once as shown above.

4.3 Implementation Details

fhnw.audiowalk.state is implemented in C as an external object for Max/MSP and compiles for OS X and Windows. While fhnw.audiowalk.state only provides the backend, an individual graphical interface can be added using common Max/MSP objects. A simple position simulation can be found within the fhnw.audiowalk.state help file [8]. File in- and output is not implemented, a script is either created by sending *nextState* messages or rather by using a Max/MSP *coll* object containing the *nextState* messages as shown in the help file.

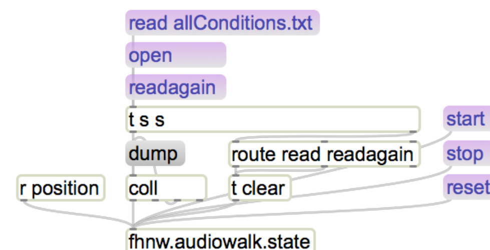

Figure 4. Attaching fhnw.audiowalk.state to a coll object

After sending the *start* message to fhnw.audiowalk.state, both the scheduler, with a default rate of 100 Hz, and the *process* message will force the testing of all client variables against all possible state transitions. This is done by iterating through a linked list, which represents the script in memory. Using a scheduler, rather than testing the position and/or orientation changes every time, makes the server workload independent from rate and accuracy of the used tracking system.

5. CLIENT

A smartphone client prototype for fhnw.audiowalk.state is implemented in the form of an Android app named *FHNW AudioWalk*. We have chosen the Android platform for the prototype because it is the least restrictive in terms of development. There is a large number of devices to choose from and all developer tools are freely available. Despite Android apps being commonly implemented in Java, libraries implemented in native code, such as *LibPd*, can be easily integrated. Background processing and multi-tasking are supported as well, which we take advantage of in the app's design.

5.1 The App Architecture

A minimal Android app consists of at least one *activity*, which basically represents one screen containing user interface elements and their underlying business logic. In addition, Android provides *services*, which can be used for tasks running in the background.

The basic structure of our app is depicted in Figure 5. It features one basic activity for the main user interface, which basically just displays log messages. A second activity is used to configure parameters such as the client ID. All the communication and data processing is performed in a background service, which allows other apps to be shown on top while fhnw.audiowalk.state keeps running in the background. Using a service is the only way, because Android has a very strict application life-cycle management that instantly pauses apps that are no longer visible in the foreground.

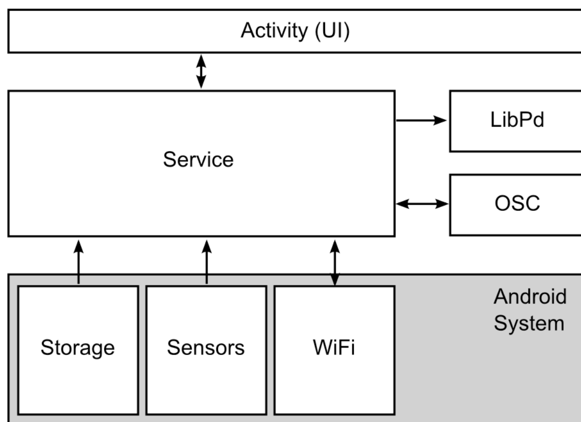

Figure 5. The Android App Architecture

A common problem concerning Android apps is that they cannot actually be closed by the user. When the UI is put into the background by opening a different app or by pushing the Home button, the operating system decides when the app is to be shut down and its memory resources released. This affects our app in particular, because we have a service running in the background which should not stop when the UI is closed. Yet we would like to be able to close the app and stop the service altogether. We work around the problem by providing an "Exit" button in the UI that stops all service tasks and puts the UI in the background. In case the UI is opened again before being destroyed by the operating system, all service tasks are automatically restarted.

A second problem related to application life-cycles is the fact that apps, including services, are generally stopped as soon as the smartphone display is switched off, thus saving battery life. Fortunately, we can deal with this problem by acquiring a *wake lock* when our service is started. A wake lock keeps the CPU awake even when the display is off, allowing us to continue exchanging messages and playing audio.

5.2 Data Management

All the data required for the audio walk, such as configuration parameters, audio files and PD patches, is stored on the external storage partition or an SD card. We have chosen this approach because Android app packages are limited to 50 MB, and also because we would like to be able to add or update data without reinstalling the app. There is even the possibility of downloading updated data directly from the app.

The localities, as well as the app configuration, are stored in JSON format. This allows an easy distribution of the configuration to all clients instead of manually configuring each smartphone. Only client-specific parameters like the client ID can be configured from within the app UI.

Pd patches and audio files are all stored in the same directory, so they can easily be accessed from LibPd.

5.3 Integration of OSC and LibPd

The Android app uses the open-source library *OscP5* [9] for OSC communication. *OscP5* is a library implemented in Java and can therefore be used directly in an Android app. All data types used in OSC are supported. The library provides the *OSCListener* interface that can be implemented by a Java class and attached to an OSC port. Incoming OSC messages are therefore handled in an object-oriented fashion.

The integration of *LibPd* [10] is a more complex task, as it is a C library. Fortunately, Android provides the native development kit (NDK), which uses Java native interfaces (JNI) in order to integrate C libraries with Java applications. Part of *LibPd* for Android consists of Java classes, providing an interface for apps. A service is provided for background processing, in addition to the *PdBase* class, which offers easy-to-use static methods to load patches, play audio files etc. The Java part accesses the native backend through JNI.

We are using a pre-compiled version of LibPd which has already been adapted to Android, which can be downloaded from Github [11].

5.4 Message Processing

The app encapsulates the OSC messages used by fhnw.audiowalk.state in individual classes. Outgoing messages are named *tracking events*, whereas incoming messages are *action events*. Tracking events are passed to an event dispatcher running in a separate thread, thus ensuring that the messages are sent in correct order. The event dispatcher hands over the messages to the OSC library. Action events received by the OSC library are queued in an event receiver and then processed in a separate thread as well.

In general, incoming OSC messages are directly passed on to LibPd. The primary job of LibPd is to interpret these messages and play a new audio file or load a new Pd patch. LibPd can also receive arbitrary symbols, which are processed depending on the Pd patch implementation.

Instead of just interpreting OSC messages, LibPd can also perform local data processing. For example, sensor data gathered by the Android app can be passed on to LibPd and used to trigger events. Different sensors are supported, such as the accelerometer and the gyroscope (if available). The advantage of doing all the processing within LibPd is that no modifications to the app are necessary. Instead, just a new Pd patch has to be copied onto the smartphone.

6. RESULTS

The test bed for implementation was the *LautLots* project [1]. The setup included two different localities with position tracking, five Samsung Galaxy SIII smartphones used as

mobile clients and a configuration of fhnw.audiowalk.state with ca. 700 state transitions.

Using an unsorted linked list as data structure for fhnw.audiowalk.state requires O(n) time for the linear search for a matching state transition. A much more optimized solution would be an array holding each *currentState* at its own index, with a linked list attached in case there is more than one transition with the same *currentState* condition. However, with a maximal CPU usage of roughly 2% on a 2.53GHz Intel Core i5, the workload of fhnw.audiowalk.state is negligible.

Using UDP in combination with OSC is efficient and easy to handle, but has the one disadvantage of lacking any acknowledgment packets that determine whether data transfer has been successful. During our tests, we had one locality with very good WLAN reception, which caused no problems concerning UDP communication. On the other hand, the second locality required multiple WLAN access points and repeaters due to its size. Wireless networks are more vulnerable to packet loss compared to wired networks, because other radio signals can cause disturbance. We encountered occasional packet loss, which resulted in smartphone clients not being registered in some cases. To mitigate this problem, important messages such as */registermobile* are repeated a few times, which drastically reduces the risk of failing to detect events. This causes only little overhead to network communication and workload, because registration messages are not very frequent.

Audio playback and data processing with LibPd on Android worked mostly as expected, we only noticed some audio stuttering in rare cases. This was most likely due to the fact that multiple computing threads were running in addition to handling OSC messages and Pd. While the Android system is multi-tasking-friendly in general, many concurrent tasks may cause difficulties with real-time applications like audio processing. The small delay between triggering an event through gestures or the position and the corresponding audio playback was mostly related to the size of the WLAN and the current Android workload. Since the latency requirements for LautLots where not very strict, this was not much of an issue and was not seriously measured.

7. CONCLUSION

The introduced software combination of server and client is easy to configure and program. The training period for the artists who used the fhnw.audiowalk.state object was only a couple of hours, although, due to the lack of resources, no graphical user interface could be provided for writing the scripts. The artists were still able to create the whole audio walk sequence (almost) without any help. Adding a user-friendly interface would lower the entry level even more. While the language is not computationally universal - only features requested by the artists were implemented - missing functionality, for example loop constructs, could be added to the source code without problems. By using the LibPd library for data processing, we have proven that it is possible to add functionality without actually modifying the application's Java and C source code.

8. ACKNOWLEDGMENTS

Thanks to Sibylle Hauert, Daniel Reichmuth, Prof. Dr. Christoph Stamm and Dr. Michael Kunkel for their patience and support during the development.

9. REFERENCES

[1] FHNW (n.y.), *Large Scale Indoortracking | Eine strategische Initiative der FHNW* [online]. URL: http://blogs.fhnw.ch/indoortracking [accessed 2014, April 23].

[2] Matthias Krebs, Cristoph Stamm, Thomas Resch (2013): "Indoor Tracking – Technik und Kunst in engem Zusammenspiel". In: FHNW (ed): *FOKUS REPORT.* Brugg-Windisch: FHNW, p. 21-30.

[3] Zaafir Barahim, M. Razvi Doomun, Nazrana Joomun (2012). "Low Cost Bluetooth Mobile Positioning for Location-based Application" In: *Proceedings of 3rd IEEE/IFIP International Conference in Central Asia on Internet (ICI2007).* Piscataway: IEEE Press, p. 1-4.

[4] Anthony LaMarca, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian Smith, James Scott, Tim Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, Bill Schilit (2005): "Place Lab: Device Position Using Radio Beacons in the Wild". In: Hans-W. Gellerson, Roy Want, Albrecht Schmidt (Hrsg.): *Pervasive Computing.* Heidelberg: Springer-Verlag GmbH, p. 116-133.

[5] Authentic Tours Limited (n.y.), *myTours* [online]. URL: http://www.mytoursapp.com [accessed 2014, April 23].

[6] Espro Acousticguide Group (n.y.), *acousticguide* [online]. URL: http://www.acoustiguide.com [accessed 2014, April 23].

[7] Toozla (n.y.), *toozla* [online]. URL: http://www.toozla.com/ [accessed 2014, April 23].

[8] Matthias Krebs, Thomas Resch (2014): *Github repository for Large Scale Indoortracking* [online]. URL: https://github.com/fhnw-imvs/fhnw-audiowalk/ [accessed 2014, April 23].

[9] Andreas Schlegel (n.y.), *Andreas Schlegel-oscP5* [online]. URL: http://www.sojamo.de/libraries/oscP5/ [accessed 2014, April 25].

[10] Create Digital Music (n.y.), *libpd* [online]. URL: http://libpd.cc/ [accessed 2014, April 25].

[11] Peter Brinkmann, Naim Falandino, Scott Fitzgerald, Peter Kirn, Hans-Christoph Steiner, (n.y.), *Github repository for LibPd for Android* [online]. URL: https://github.com/libpd/pd-for-android/ [accessed 2014, April 25].

10. Appendices

For detailed information about Max/MSP please refer to www.cycling74.com.