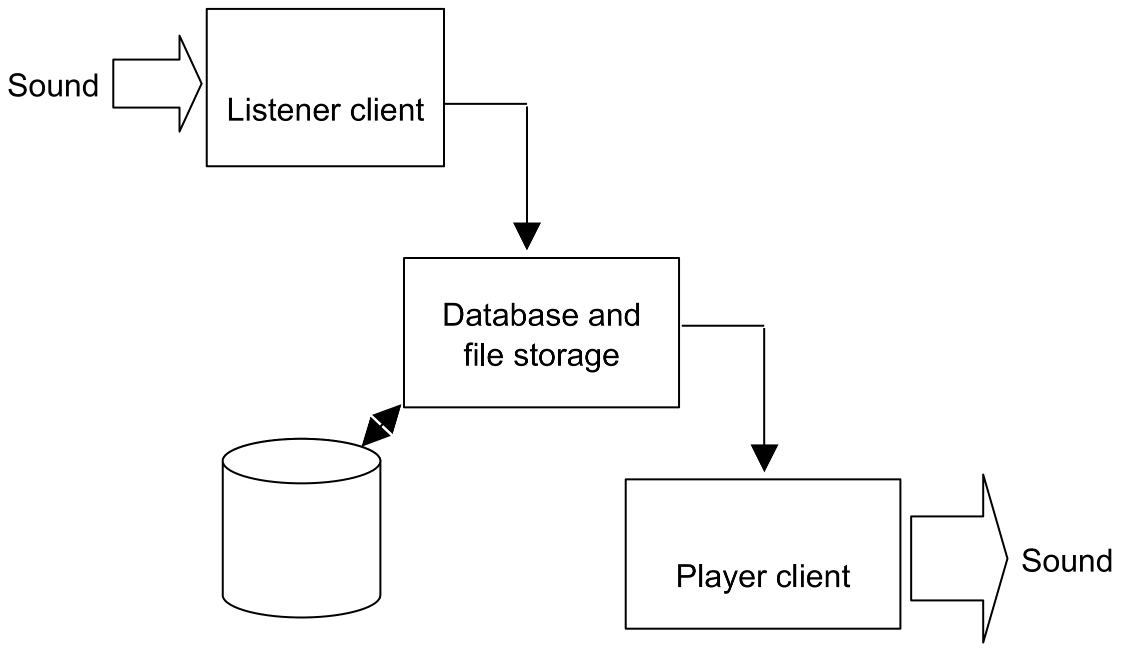
The Curious Listener

Overview

The Curious Listener is a system which automatically monitors the audio activity in an area and records anything interesting! It consists of three types of components which connect with each other over a standard TCP/IP network: -

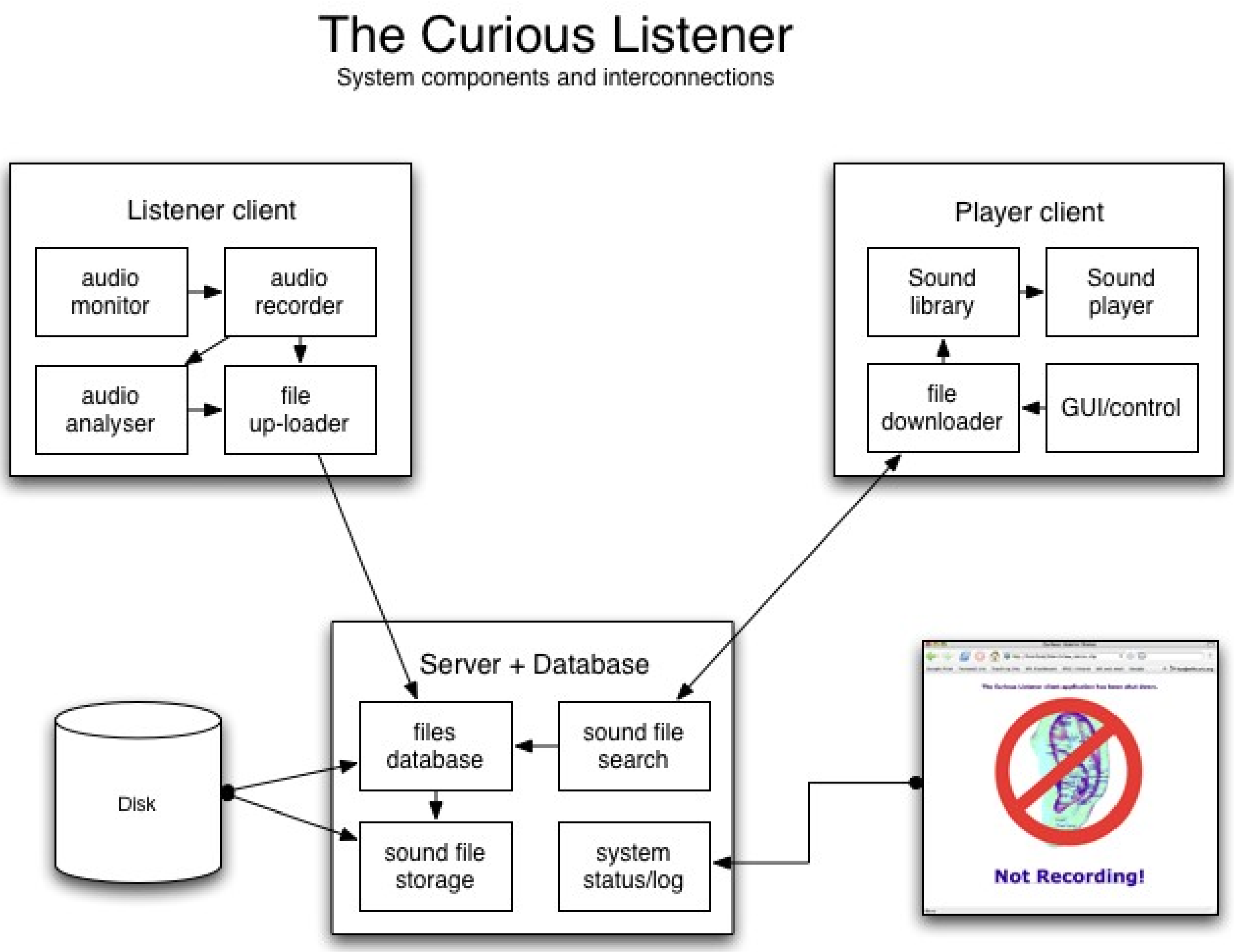


When a listener component has recorded something (speech, music, unusual noises etc.), it uploads the digital sound file and information about the audio to a remote database component. This stores information about the recording, such as when it was made and what sort of sound activity it contains and maintains a searchable catalogue of recordings. Eventually it is envisaged that the server system will attempt to further analyse recordings based on their formant signature, the patterns of unique sonic resonances present in an individual’s speech – from which the system would attempt to identify the who was speaking. Over time a centralised catalogue of sound recordings will be built up, each tagged with its recording date and time and with a guess about who is speaking…

The player client, is able to interrogate the database and retrieve any recordings made within a specific timeframe. These files are then sampled using a random algorithm and a customised dynamic envelope and played back through multiple sound channels (mixed down to stereo in the present system). The effect is of a continuously changing sound collage in which odd, isolated words and phrases are heard fading in and out, but anything longer than a couple of seconds is truncated and fragmented.

Each of the components of the system has a complex internal structure and has been developed using OOP application design principles.

A more detailed picture of the system showing some of the internal detail is: -

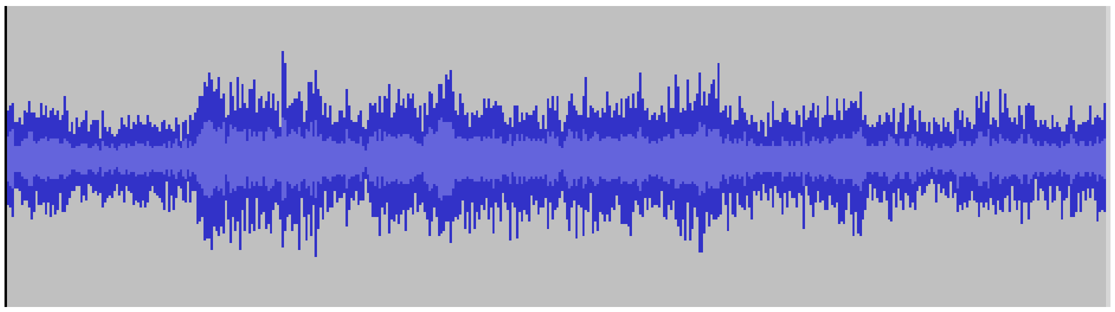


Listener Client

The listener client is largely implemented in Director with Flash used for some interface elements. Director was chosen due to its rapid development times, ability to address the local file system and capability to deal with audio effectively on both Mac and PC platforms. The automatic file uploader was implemented using a local (apache) web server and php-curl components.

The audio monitor continually monitors the audio input level. If it rises rapidly over a set sensitivity threshold, it triggers the recording component which logs the start time and begins recording. The recording continues until a significant period of non-activity has occurred (1.0 seconds in the current configuration). However, the input level is monitored throughout the recording and off-on and on-off transient times are logged: -

Here is a sample recording (note this has been normalised to -0.3 dB so that transients can be identified visually).



The recording shown here is 2.77 seconds long and the listener client identified 4 transient pairs (on + off) within this time:-

0	0.04
0.524	0.566
1.125	1.165
1.73	1.77

As soon as the recording is ended, the transient log is written to a text file and then the analysis component runs a simple rule-based algorithm taking as inputs the transit density, the average length of transients and the duration of the shortest transient. This process returns a probable audio-type for the recording – currently, the system attempts to distinguish between speech, music and ‘unusual noise’.

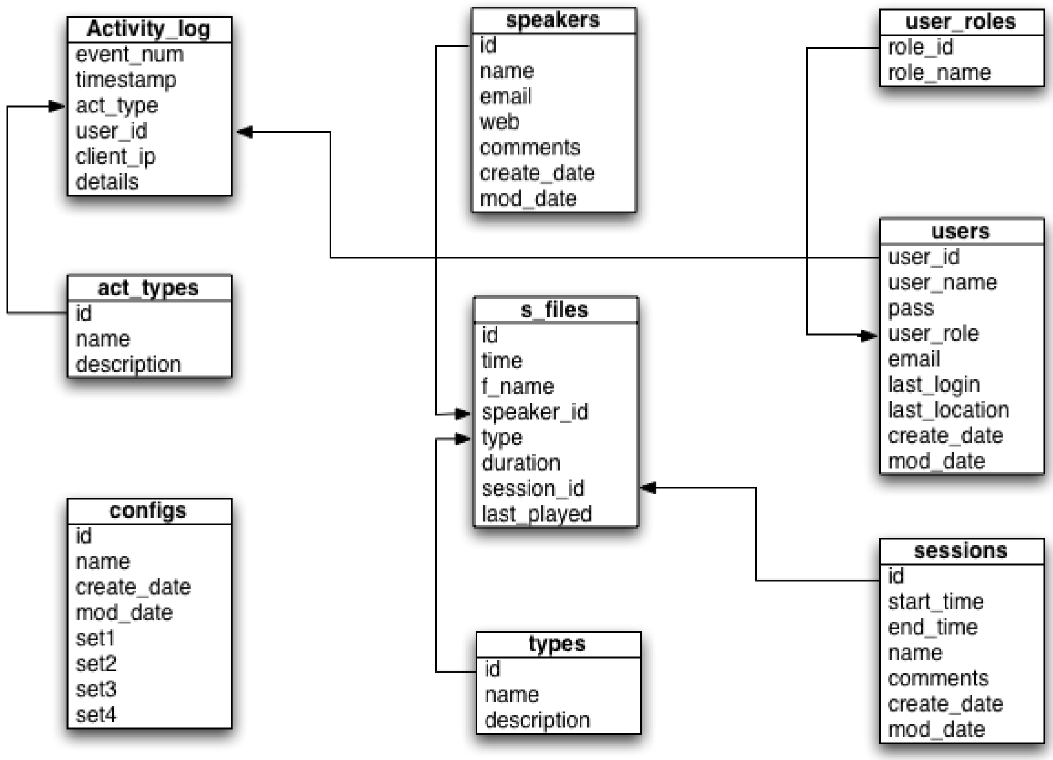
Once the analysis is complete, the sound and data files are uploaded to the server by a call to a php upload script running on a local web server (apache). This rather complex arrangement is necessary since while Director is able to access the local file system, php on a remote server is not without user confirmation[1], although one php-apache system can send files to another and can access files on the machine it is running on.

After a successful upload has been confirmed by the remote file store, an entry is made in the database catalogue (see below) using a Director call to a remote php script and the local files are deleted.

Database and file server

The database is implemented in mySQL with a php/apache layer providing the network interface for recorder and player clients.

The database schema showing the table and field names[2] and data relations is: -



Not all of these tables are used in the current implementation. The most important table is *s_files* which stores details of all the recordings held in the system.

The sound and transient data files are uploaded and downloaded using the same apache web server used for delivering the php communication layer. Principle components are: -

upload_file.php is called locally with a filename. It causes the local web server to upload the specified sound and data files to the remote server.

store_sound.php is called by a remote listener client to add a sound file entry to the database.

get_sound_list.php is called by a remote player client with start and end times and dates. It returns a list of files recorded within the time period supplied. These can then be downloaded by a standard http requests from the player client.

show_status.php can be called from a browser and displays an image and text indicating the current status of the listener client. It automatically refreshes every 5 seconds and is designed for continual monitoring of system status to alert people when recording is active. See http://bartleby.ioct.dmu.ac.uk/~ianw37/listen/show_status.php

list_sounds.php can be called by a web browser and displays a list of the last 100 recordings made by the system formatted for a human reader. It is designed as a diagnostic tool during development and would not be present on a publicly deployed system. Sample output is: -

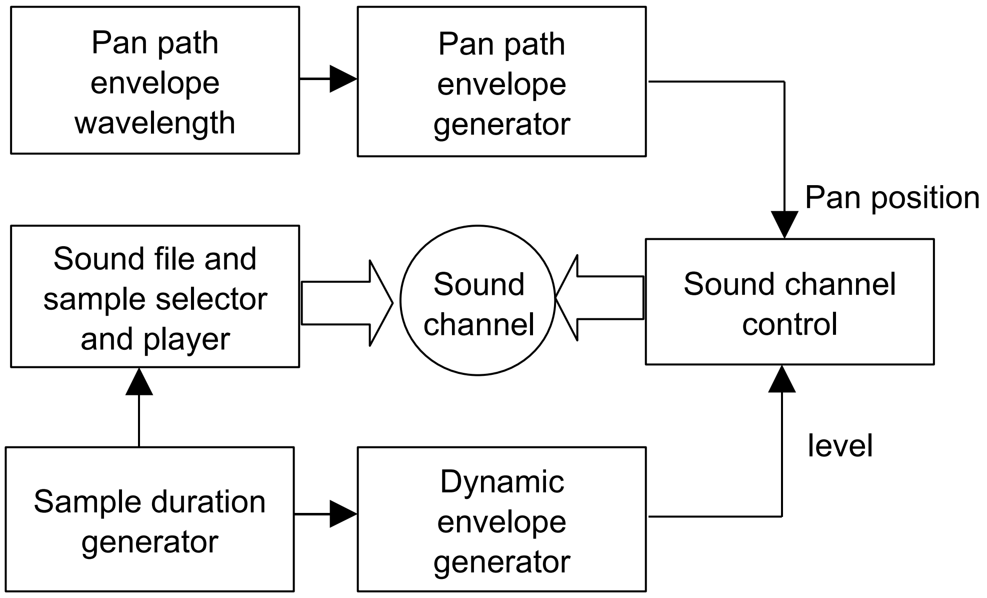
Recording Date	Speaker	Duration (Secs)	Type	Filename (click to listen)	Transit Data
28th February, 2007 at 17:44:20	unknown	9.544	speech	07_02_28_17_44_29726081	data
28th February, 2007 at 17:44:07	unknown	8.014	speech	07_02_28_17_43_29715630	data
28th February, 2007 at 17:43:57	unknown	24.988	speech	07_02_28_17_43_29688023	data
28th February, 2007 at 17:43:28	unknown	0.16	unusual noise	07_02_28_17_43_29684746	data
28th February, 2007 at 17:43:26	unknown	0.64	unusual noise	07_02_28_17_43_29681211	data

Player client

The player client is implemented as Director stand alone application, although a Flash version designed for browser-based operation would also be relatively simple, although less secure.

The client gets a list of sound files within it's time period (which is part of the player configuration settings) and then downloads them to the local web cache. The sound playback part of the client is made up of 4 identical playback engines.

The functional blocks of each are: -


Pan path envelope wavelength
Pan path envelope generator
Pan position
Sound file and sample selector and player
Sound channel
Sound channel control
level
Sample duration generator
Dynamic envelope generator

Each playback engine has access to all the soundfiles that the player client has obtained from the central file server. They are not synchronised (indeed the pan-path generators are deliberately started with different values for wavelength and phase so that sounds are spread across the stereo space.

[1] Flash has the same limitation; it requires a user to select and confirm file uploads. Unfortunately, Director does not have an open source (or cheap) CURL Xtra available which would have made implementation much simpler.

[2] Note that field data types are not shown due to space. Record identifiers are integers, length 5; other fields are either characters or sql date formats accordingly.