It can be shown that this grammar can be parsed by LR(1) without ambiguity ("Reduce-reduce" error or "Shift-shift" error). "Reduce-shift" errors occur but can be solved by "preferring reduce then shift."

6. IMPLEMENTATION

We implemented a Java-based API called MCFC (music control flow compiler) that creates a score compiler from a user-defined lexical definition and grammars at start-up and translates string-format score symbols to a flattened score based on the given grammar. Unlike lex and yacc [11], which are used frequently for generating code frameworks for making compilers, MCFC generates the entire compiler directly from the CFG and attribute grammar, and users can switch among different grammars without closing the application. This feature is especially useful when dealing with notation from different domains.

MCFC does not assume any model of the score and works in the string representation. The first step for any application to have this API embedded is to convert the score to a list of symbols as demonstrated in Section 5.1.

To make the compiler understand the semantics of these symbols, the user needs to provide an additional file declaring all the symbols, which is known as lexical definition, and is accomplished using regular expressions similar to those used in lex [11].

6.1. Grammar Definition

The grammar definition is put in an additional file with extensions "_g". The content of the grammar definition is essentially identical to the attribute grammar shown in Section 5.6.

6.2. The Intermediate Grammar

The score compiler supports a built-in grammar that uses a small set of instructions to make control flow more flexible. The basic instructions are shown in Table 2.

Table 2: Default Intermediate Language Grammar

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( &amp; &lt; N &gt;, &lt; N &gt; )</td>
<td>A section mark, ( A ) in ( (A, A, 2) )</td>
</tr>
<tr>
<td>( (\text{loop}, \text{LC}) \rightarrow )</td>
<td>A loop mark labelled by ( L ), similar to Left Repeat</td>
</tr>
</tbody>
</table>
| \( (\text{rep}, \text{LC} 
\rightarrow \text{T} \rightarrow \text{CN}) \rightarrow \) | Repeat to Label \( L \) for \( T \) times, similar to Right Repeat |
| \( (\text{th}, \text{B} \rightarrow \text{N} \rightarrow \) | A jump mark labelled by \( B \) |
| \( (\text{jmp}, \text{L} \rightarrow \text{C} \rightarrow \text{N} \rightarrow \text{B}) \rightarrow \) | jump to \( B \) at the \( T \)-th repetition for loop \( L \) |

The intermediate grammar can be used to annotate word-defined scores as in Figure 3. It is very useful to compile the original score to this intermediate form and then use the built-in translator to further compile it to a flattened score. The compiler for the intermediate grammar has sophisticated functions such as loop mapping, which helps to relate static score positions back to multiple performance positions \( k \) in \( f(B) \). For example, one might want to select score position as it occurs in the "2nd repeat after the D.S." For example, consider the score

\[
(A_1, 2) = (b, 4, 4) (b, 4, 4) (b, 0, 4) (b, 0, 4) \]

which is translated into intermediate presentation

\[
(A_1, 2) = (b, 4, 4) (b, 4, 4) (b, 0, 4, 4) (b, 4, 4) \]

and finally to the flattened score

\[
(A_1, 2) = (b, 4, 4) (b, 4, 4) (b, 0, 4, 4) (b, 0, 4, 4) \]

Output symbols are marked with labels (below) called flags that show the mapping from the symbol to loop. The format is \([L1;L2] \) count 2, where count is the number of repetitions of loop \( L \). Notice that \( (b, 0, 4) \) appears with four distinct loop labels

\[
([L0,1,1;L0,1,1] [L0,1,1;L0,1,2] [L0,1,2;L0,2]) \]

6.3. Rearrangement

The other feature of this built-in translator is its ability to handle rearrangement. In the example above, the score is separated into subsections by section marks.

\[
A_1:-1 \quad (b, 4, 4) \quad (b, 4, 4) \quad (b, 0, 4, 4) \quad (b, 0, 4, 4) \quad (b, 0, 4, 4)
\]

One can input a rearrangement based on the section marks and the loop mapping.

\[
A_1:-1 \quad A_2:-2;L_2 \quad A_1:-1
\]

Then the flattened score is

\[
A_1:-1 \quad (b, 4, 4) \quad (b, 4, 4) \quad (b, 0, 4, 4) \quad (b, 0, 4, 4) \quad (b, 0, 4, 4)
\]

In this way the rearrangement notation used in Figure 3 can be compiled.

7. APPLICATION

The score compiler is used in a music score display application that, in turn, can be used as a part of a human-computer interface for score following, human computer music performance, music education, multimedia databases, etc., where the correct reading order is essential. As a demonstration, we built a system called Live Score Display (LSD) for short that performs import score images, annotate control flow symbols, re-arrange the score and play the score in real time. With the flattened score representation, the user can browse to any repetition of a repeat, even in heavily nested repeats.
system is JackTrip (CCRMA, Stanford University) where several remote audio streams can be mixed together from a master server [4]. However, several limitations arise when high quality audio streams need to be sent and synchronized over the internet, given the large amount of data involved. An alternative networking procedure can be achieved by using a lighter and more flexible musical data format like OSC, where no real audio is required, but a series of short messages.

This protocol, in combination with the SonicMaps locative audio tool, opens a door to the possibility of linking not just performers, but multiple distant spaces in new imaginative ways. For instance, we will discuss how a locative audio urban experience, in the terms earlier described, can be connected to a live Concert Hall event, establishing meaningful relationships between these two spaces and their audiences.

2. SOFTWARE DESIGN

SonicMaps was developed using the Unity3D Game Engine1 and takes advantage of its 3D spatial audio engine, along with its rendering and physics engine, scripting languages (JavaScript, C#), and multi-platform publishing capability. Currently, SonicMaps is available on iOS and Android devices while user content management and support is provided from a PHP enabled website2.

In order to geolocate sounds, a geographical positioning system was implemented by translating the geographical coordinate system used by GPS mobile devices (WGS84), into Unity's Vector2 Cartesian coordinates (x,y). This was completed using the Mercator projection2.1. The map tile for the current location is obtained from the Google Static Maps3 service using a Wi-Fi or 3G internet connection, and it serves as a texture for a simple plane mesh whose dimensions correspond to the considered scale. On the other hand, user position is displayed on screen as a small blue arrow that reads the user's GPS tracking2.2.

This value is stored on every session, so if we run the application somewhere else and the initial reference point changes, all objects positions can be recalculated while numbers are kept below the 7 digits limitation.

Additionally, we avoided using absolute coordinates from a fixed 3D perspective and opted for a relative positioning system that takes the first GPS coordinates (on application start) as a relative frame of reference. This value is stored on every session, so if we run the application somewhere else and the initial reference point changes, all objects positions can be recalculated while numbers are kept below the 7 digits limitation.

A map tile for the current location is obtained from the Google Static Maps3 service using a Wi-Fi or 3G internet connection, and it serves as a texture for a simple plane mesh whose dimensions correspond to the considered scale. On the other hand, user position is displayed on screen as a small blue arrow that reads the user's GPS tracking2.2.

This value is stored on every session, so if we run the application somewhere else and the initial reference point changes, all objects positions can be recalculated while numbers are kept below the 7 digits limitation.

2.1. User Interface

When starting SonicMaps, it will first try to determine our current position. The message “Searching for Satellites" is displayed on the screen so location Services needs to be enabled on the device before we run the application, as it is necessary to check that GPS signal is good enough; otherwise, after 30s user is asked to check location settings and restart SonicMaps.

Sometimes, when connected to Wi-Fi (without GPS enabled), the application might still be able to get our location, but this will result in much less accuracy.

Once our current location has been set, the main UI layout shows up (Fig.2).

Figure 2. SonicMaps User Interface.

A 2D sound is just a common stereo or mono audio file which is played with no spatial qualities, just as they would be heard from any regular music player.

The 3D property allows the spatial sound feature we discussed previously. However, all 3D sounds in Unity are forced into mono by default, for easier panning and a more effective spatial localization. This limitation can be partially corrected by emulating distance attenuation with a simple custom JavaScript to produce a stereo pseudo-3D audio source, with no panning or directional properties, whose volume increases as we get closer to the center of the defined area.

Audio clips can also be looped, which is useful when we need to maintain a sustained sonic texture within an area and it is not possible to predict how long will the user stay in that particular location.

Another important issue, specially when working with 2D sounds, is that whenever a listener leaves a sound area, the corresponding audio clip would suddenly stop playing and produce undesirable artifacts (clicks, etc.). Nevertheless, a 2-3 sec fade out script (depending on frame rate) is activated on trigger exit, providing a more consistent termination.

2.2. Sensors implementation

2.2.1. GPS tracking

Thanks to GPS Location Services, user position is updated every half a second, although transitions are smoothed over time using linear interpolation. The sensor’s accuracy is graphically displayed using a white error-circle range around the position arrow. In some devices this accuracy may be as good as 5 m or less.

Tall buildings, mountains, and other objects, often interfere with or interrupt the GPS signal from satellites and Google maps are not always 100% accurate so they might present significant discrepancies from the actual terrain [5]. SonicMaps tries to minimize these issues by offering what we have defined as “on-site” editing, that is to say, being able to edit projects from a mobile device, as we walk, and ensuring that sounds are just at the desired location. Therefore, positioning inaccuracies are taken into account while editing so unwanted displacements are not perceived during content playback. This method is also useful to decide if sound materials are aesthetically or functionally suitable for a particular place, by actually being there as we edit.

2.2.2. Solid state compass

The solid state compass sensor is called every 50ms in order to get orientation data. The user position arrow rotates to match the new heading and surrounding sounds are panned accordingly. Noise data from the compass sensor due to random movements, e.g., when handling the mobile device as we walk, is filtered so only substantial rotations are visualized.

2.3. Audio Engine

2.3.1. Spatial 3D sound

The Unity Game Engine uses an implementation of the FMOD audio library for the creation and playback of interactive audio. This library includes an Audio Listener component (virtual microphone) to output spatial audio resulting from any audio source in a scene. SonicMaps attaches the audio listener to the user position arrow so sounds are perceived from this point of view. Likewise, Audio Source components can be found in every sound area we create and will be assigned a transform position (Vector3 ). Therefore, audio signals experiment a customizable amplitude drop-off (volume attenuation) over distance and are panned into the stereo sound field according to their relative angular position. Multichannel configurations3 are also possible but they are not currently supported on mobile devices.

Three playback modes have been devised as a result of combining two fundamental variables:

A. 3Dmono
B. 2D stereo or mono
C. Pseudo-3D stereo

A 2D sound is just a common stereo or mono audio file which is played with no spatial qualities, just as they would be heard from any regular music player.

The 3D property allows the spatial sound feature we discussed previously. However, all 3D sounds in Unity are forced into mono by default, for easier panning and a more effective spatial localization. This limitation can be partially corrected by emulating distance attenuation with a simple custom JavaScript to produce a stereo pseudo-3D audio source, with no panning or directional properties, whose volume increases as we get closer to the center of the defined area.

Audio clips can also be looped, which is useful when we need to maintain a sustained sonic texture within an area and it is not possible to predict how long will the user stay in that particular location.

Another important issue, specially when working with 2D sounds, is that whenever a listener leaves a sound area, the corresponding audio clip would suddenly stop playing and produce undesirable artifacts (clicks, etc.). Nevertheless, a 2-3 sec fade out script (depending on frame rate) is activated on trigger exit, providing a more consistent termination.

2.3.2. Sound Areas and audio buffering

Every sound area can be explained as a combination of two separate components: a buffering area and a playback area (Fig.3).

By default, every sound area includes a link URL to a sample audio clip (http://sonicmaps.org/sample.mp3) that will start downloading whenever we enter the buffering area. This buffering area guarantees that the sound is ready to play by the time we reach the actual playback area. The size of the buffering area is proportional to the size of the playback area, and once the download is complete the latter turns green as a notification.

3.1. Sound Source components 3.2. Pseudo-3D stereo

3 Quad, 5.1 and 7.1 surround.

4 http://www.jsongr.

5 http://sonicmaps.org


7 http://sonicmaps.org
SonicMaps was developed using the Unity3D GameEngine and takes advantage of its 3D spatial audio tool, along with its rendering and physics engine, scripting languages (JavaScript, C#), and multi-platform publishing capability. Currently, SonicMaps is available on iOS and Android devices while user content management and support is provided from a PHP enabled website2.

In order to geolocate sounds, a geospatial positioning system was implemented by translating the geographical coordinate system used by GPS mobile devices (WGS84), into Unity's Vector2 Cartesian coordinates (x,y). This was completed using the Mercator projection (WGS84), into Unity's Vector2 Cartesian coordinates. That means you can only have 7 digits to set a coordinate and anything longer than that will be truncated with a consequent loss of valuable information.

Additionally, we avoided using absolute coordinates from a fixed 3D reference frame and opted for a relative positioning system that takes the first GPS coordinates (on application start) as a relative frame of reference. This value is stored on every session, so if we run the application somewhere else and the initial reference point changes, all objects positions can be recalculated while numbers are kept below the 7 digits limitation.

2.1 User Interface

When starting SonicMaps, it will first try to determine our current position. The message “Searching for Satellites” is displayed on the screen so location Services needs to be enabled on the device before we run the application. It is necessary to check that GPS signal is good enough; otherwise, after 30s user is asked to check location settings and restart SonicMaps. Sometimes, when connected to Wi-Fi (without GPS enabled), the application might still be able to get our location, but this will result in much less accuracy.

Once our current location has been set, the main UI layout shows up (Fig.2).

A map tile for the current location is obtained from the Google Static Maps3 service using a Wi-Fi or 3G internet connection, and it serves as a texture for a buffer area. This buffering area guarantees the spatial relationships between these two spaces and their audiences.

2. SOFTWARE DESIGN

Figure 2. SonicMaps User Interface.

However, audio files are not initially stored in the mobile device but online, using a personal server or file hosting service. Therefore, sound areas only contain URLS to online audio files, avoiding the manual process of having to transfer these files into every device before we can play a project. Every project is saved as a JSON4 text file containing serialized data about the sound areas and their properties. This makes content publishing and sharing extremely easy, with the help of a dedicated database.

2.2. Sensors implementation

2.2.1. GPS tracking

Thanks to GPS Location Services, user position is updated every half a second, although transitions are smoothed over time using linear interpolation. The sensor's accuracy is graphically displayed using a white error-range circle around the position arrow. In some devices this accuracy may be as good as 5 m or less. Tall buildings, mountains, and other objects, often interfere with or interrupt the GPS signal from satellites and Google maps are not always 100% accurate so they might present significant displacements from the actual terrain [5]. SonicMaps tries to minimize these issues by offering what we have defined as “on-site” editing, that is to say, being able to edit projects from a mobile device, as we walk, and ensuring that sounds are just at the desired location. Therefore, positioning inaccuracies are taken into account while editing so unwanted displacements are not perceived during content playback. This method is also useful to decide if sound materials are aesthetically or functionally suitable for a particular place, by actually being there as we edit.

2.2.2. Solid state compass

The solid state compass sensor is called every 50ms in order to get orientation data. The user position arrow rotates to match the new heading and surrounding sounds are panned accordingly. Noise data from the compass sensor due to random movements, e.g., when handling the mobile device as we walk, is filtered so only substantial rotations are visualized.

2.3. Audio Engine

2.3.1. Spatial 3D sound

The Unity Game Engine uses an implementation of the FMOD audio library for the creation and playback of interactive audio. This library includes an Audio Listener component (virtual microphone) to output spatial audio resulting from any audio source in a scene. SonicMaps attaches the audio listener to the user position arrow so sounds are perceived from this point of view. Likewise, Audio Source components can be found in every sound area we create and will be assigned a transform position (Vector3). Therefore, audio signals experiment a customizable amplitude drop-off (volume attenuation) over distance and are projected into the stereo sound field according to their relative angular position. Multichannel configurations5 are also possible but they are not currently supported on mobile devices.

Three playback modes have been devised as a result of combining two fundamental variables:

- 3D/mono
- 2D/stereo or mono
- Pseudo-3D stereo

A 3D sound is just a common stereo or mono audio file which is played with no spatial qualities, just as they would be heard from any regular music player. The 3D property of the spatial sound feature we discussed previously. However, all 3D sounds in Unity are forced into mono by default, for easier panning and a more effective spatial localization. This limitation can be partially corrected by emulating distance attenuation with a simple custom JavaScript to produce a stereo pseudo-3D audio source, with no panning or directional properties, whose volume increases as we get closer to the center of the defined area.

Audio clips can also be looped, which is useful when we need to maintain a sustained sonic texture within an area and it is not possible to predict how long will the user stay in that particular location.

It also helps to keep an audio file size small, which is strongly recommended when relying on a slow and sometimes expensive 3G internet connection.

Another important issue, specially when working with 2D sounds, is that whenever a listener leaves a sound area, the corresponding audio clip would suddenly stop playing and produce undesirable artifacts (clicks, etc.). Nevertheless, a 2-3 sec fade out script (depending on frame rate) is activated on trigger exit, providing a more consistent termination.

2.3.2. Sound Areas and audio buffering

Every sound area can be explained as a combination of two separate components: a buffering area and a playback area (Fig.3). By default, every sound area includes a link URL to a sample audio clip (http://sonicmaps.org/sample.mp3) that will start downloading whenever we enter the buffering area. This buffering area guarantees that the sound is ready to play by the time we reach the actual playback area. The size of the buffering area is proportional to the size of the playback area, and once the download is complete the latter turns green as a notification.

1 http://unity3d.com/
2 http://sonicmaps.org
3 https://developers.google.com/maps/documentation/staticmaps/
4 http://www.json.org/
5 Quad. 5.1 and 7.1 surround.
2.3.3. Audio Playback

Any audio file contained in a sound area will automatically start playing any time we enter its playback area (Fig.3). When we exit the playback area, if the sound is still playing, it will rapidly fade-out and stop playing. The fade-out time is no longer than 1-2 seconds and it has been implemented to avoid sonic artifacts. An external audio editor can be used if we need longer fade-outs or any fade-in. SonicMaps does not apply any fade-in effect to sounds when entering an area. However, a natural fade-in effect is perceived when using the 3D audio option, since amplitude depends on how close we are to the center of the sound area.

2.4. Sound Areas editing

To create a new sound area, users just need to tap the map right at the place where they want to locate it and press a “New Area” button in a resulting window. A gray circular area is then instantiated.

If we touch this newly created area, it turns red (editing mode) and the sound properties windows are displayed (Fig.4).

In this new window user can set the URL for the corresponding audio file, toggle between 3D/2D modes, activate the loop playback, set the volume, etc. The size of the sound area can also be modified with a simple touch and drag gesture.

2.5. Project publishing. A public platform for locative audio content.

Once all sound areas have been created, the project can be saved and published along with some descriptive metadata into a MySQL database which is accessible for other users via a web page.


If we turn our attention to the content itself and its relation with the environment we will soon realize that this living spaces are not static but dynamic; they evolve over time and affect the meaning of our sounds and vice-versa [1]. Consequently, under changing environmental conditions, a ‘placed sound’ might lose part of its meaning and suitability or even suggest something different of what it was intended.

We tried to resolve this problem by using a dynamic content server that provides adapted versions of a sound depending on the current environmental conditions (time, date, temperature, etc.).

This process relies on a Unix cron job, which periodically calls a php script on the server. This script reads the current conditions from the system status (time, date) and other online data feeds (weather conditions) and automatically selects the most adequate sounds for every sound area in a SonicMaps project.

The update is achieved using a folders system to organize the available resources while the php script copies the right files into an active folder that can be accessed by the mobile application (Fig.6).

2.7. OSC Module.

In order to facilitate network communications between the locative audio tool and any external software capable of handling OSC messages, we have implemented a OSC module for SonicMaps.

This module sends the user’s position to a remote IP address via UDP packets so multiple clients are able to connect simultaneously (Fig.7). However this particular approach requires that the specified port number is open in the server machine and port forwarding has been enabled (when behind a router).

To load a published project, users need to provide a link to the corresponding JSON text file (available in the web page) so SonicMaps can re-create all the sound areas in that project.

This system described allows the user to execute all necessary steps from one single mobile device without the need of USB data cables or any external equipment.

The website and database was designed to serve as an open platform for free locative audio content with a focus on usability and accessibility.

The OSC messages being sent are formatted as follows:

```
/userName/GPS_absolute lon lat
/userName/XY_relative transform.position.x transform.position.z
```

where GPS_absolute is the actual longitude and latitude from the GPS sensor and XY_relative are the Cartesian coordinates in Unity’s world units using the frame of reference we mentioned earlier. Notice that the Y coordinate is obtained from transform.position.z given that Unity uses the Y axis as height, so horizontal movements are described on the XZ plane.

Whether we choose one or the other depends on how we intend to make use of this information and what is the software running on the other side of the network.

3. CONNECTING SPACES

SonicMaps might be used as a stand-alone compositional tool and playback system for locative
2.3.3. Audio Playback

Any audio file contained in a sound area will automatically start playing any time we enter its playback area (Fig.3). When we exit the playback area, if the sound is still playing, it will rapidly fade-out and stop playing. The fade-out time is no longer than 1-2 seconds and it has been implemented to avoid sonic artifacts. An external audio editor can be used if we need longer fade-outs or any fade-in. SonicMaps does not apply any fade-in effect to sounds when entering an area. However, a natural fade-in effect is perceived when using the 3D audio option, since amplitude depends on how close we are to the center of the sound area.

2.4. Sound Areas editing

To create a new sound area, users just need to tap the map right at the place where they want to locate it and press a “New Area” button in a resulting window. A gray circular area is then instantiated.

If we touch this newly created area, it turns red (editing mode) and the sound properties windows are displayed (Fig.4).

In this new window user can set the URL for the corresponding audio file, toggle between 3D/2D modes, activate the loop playback, set the volume, etc. The size of the sound area can also be modified with a simple touch and drag gesture.

To load a published project, users need to provide a link to the corresponding JSON text file (available in the web page) so SonicMaps can re-create all the sound areas in that project.

This system described allows the user to execute all necessary steps from one single mobile device without the need of USB data cables or any external equipment.

The website and database was designed to serve as an open platform for free locative audio content with a focus on usability and accessibility.

2.5. Project publishing. A public platform for locative audio content.

Once all sound areas have been created, the project can be saved and published along with some descriptive metadata into a MySQL database which is accessible for other users via a web page.


If we turn our attention to the content itself and its relation with the environment we will soon realize that this living spaces are not static but dynamic; they evolve over time and affect the meaning of our sounds and vice-versa [1]. Consequently, under changing environmental conditions, a ‘placed sound’ might lose part of its meaning and suitability or even suggest something different of what it was intended.

We tried to resolve this problem by using a dynamic content server that provides adapted versions of a sound depending on the current environmental conditions (time, date, temperature, etc.).

This process relies on a Unix cron job, which periodically calls a php script on the server. This script reads the current conditions from the system status (time, date) and other online data feeds (weather conditions) and automatically selects the most adequate sounds for every sound area in a SonicMaps project.

The update is achieved using a folders system to organize the available resources while the php script copies the right files into an active folder that can be accessed by the mobile application (Fig.6).

2.7. OSC Module

In order to facilitate network communications between the locative audio tool and any external software capable of handling OSC messages, we have implemented a OSC module for SonicMaps.

This module sends the user's position to a remote IP address via UDP packets so multiple clients are able to connect simultaneously (Fig.7). However this particular approach requires that the specified port number is open in the server machine and port forwarding has been enabled (when behind a router).

The OSC messages being sent are formatted as follows:

```
/userName/GPS_absolute lat lon
/userName/XY_relative transform.position.x transform.position.z
```

where GPS_absolute is the actual longitude and latitude from the GPS sensor and XY_relative are the Cartesian coordinates in Unity's world units using the frame of reference we mentioned earlier. Notice that the Y coordinate is obtained from transform.position.z given that Unity uses the Y axis as height, so horizontal movements are described on the XZ plane.

Whether we choose one or the other depends on how we intend to make use of this information and what is the software running on the other side of the network.

3. CONNECTING SPACES

SonicMaps might be used as a stand-alone compositional tool and playback system for locative
audio. Nevertheless, the implementation of network capability via OSC suggests a number of more elaborated forms of networked and immersive sound exploration. In particular, we were interested in the possibility of connecting sonically augmented outdoor urban spaces, with the more traditional concert hall or museum experience, proposing a creative framework for composers and sonic artists to construct a dual-interlinked event.

3.1. SonicMaps + MAX

An interesting sound installation, Kinesthesia (Ivica Ico Bukvic, 2012), exploring these ideas, was presented during an Immersive Audio-game Showcase in the University of Manchester. The piece analyzes the human geospatial motion of multiple participants, whose cumulative actions form a subconsciously collaborative data stream devoid of time and space. The ensuing data stream was broadcast from Blacksburg (United States) to Manchester (United Kingdom) where it is reconstructed inside the autonomous meta-instrument and presented to an audience as a persistent spatially-aware installation. Up to eight people with mobile devices running SonicMaps were asked to walk across the Virginia Tech Campus, experiencing a custom soundwalk. Meanwhile, the OSC module was sending their positions to a remote computer in Manchester running a MAX patch that used the incoming geolocated data to feed several sound objects and shape a multichannel musical output.

3.2. SonicMaps + Game Engine (Unity3D)

It is also possible to connect a physical real space and a computer-generated virtual representation of that space for a sonic-centric networked interaction. For example, the piece Alice: Elegy to the Memory of an Unfortunate Lady (Ignacio Pecino, 2012), uses a Unity3D virtual model of Whitworth Park (Manchester) to display the current position of a remote user walking on the real physical park. This virtual environment is then presented to an audience in a concert hall so it is possible to experience, in a synchronized way, the same sounds the remote performer is triggering as he walks outdoor using the SonicMaps mobile application.

4. RELATED WORKS

A relevant work by composer Andy Dolphin in collaboration with programmer Kingsley Ash, Urbiculous Disport (2012), is an audio-visual installation exploring connections between sounds and interactive abstract sound toys generated with the Unity3D game engine. In their own words:

"Urbiculous Disport is an interactive, generative sound-toy installation in which participants capture, manipulate and play with the sounds of the city. Participants use mobile devices to collect sounds from around the city, which are then streamed, reworked and transformed to form the source materials in an original sound-toy application presented in an installation setting. The work therefore exists as a collaboration between the sound recordists and the installation participants, allowing them to explore the sonic environment of the city in different ways and collaboratively influence the resulting audio-visual output."

An example of locative sound using a smart phone application is the National Mall (2011), a Sound-on-Sound looping application exploring connections between sounds. For example, the piece Alice: Elegy to the Memory of an Unfortunate Lady (Ignacio Pecino, 2012), uses a Unity3D virtual model of Whitworth Park (Manchester) to display the current position of a remote user walking on the real physical park. This virtual environment is then presented to an audience in a concert hall so it is possible to experience, in a synchronized way, the same sounds the remote performer is triggering as he walks outdoor using the SonicMaps mobile application.

5. REFERENCES


DEVELOPING A CHROMATIC INTERFACE FOR REAL-TIME DIGITAL HARMONISATION OF SAXOPHONE PERFORMANCE

James Savage
Faculty of VCA and MCM
The University of Melbourne
jsava@student.unimelb.edu.au

ABSTRACT

A new interface for musical expression, the Harmonic Table Pitch Shifter (HTPS) is presented. It is a harmonisation system that allows free selection of intervallic voicings to harmonise live saxophone performance in real-time, resulting in homophonic harmony in up to five parts. Pitch-shifting software developed in Max/MSP reacts to MIDI messages to produce the desired harmonisation features and functions as an envelope-based mixer to give the added voices a dynamic contour over time. A custom MIDI footpedal used on the saxophone arrangement known as the ‘harmonic table’ places the most consonant intervals nearest one another and also exhibits transpositional invariance, allowing chordal ‘shapes’ to be freely moved within the array without affecting their intervallic relationship. Microswitches mounted on the saxophone thumb rest give quick access to a piano pedal-like effect allowing drones or chords to be sustained below melodic playing or improvisation.

1. INTRODUCTION

My experience playing the guitar, piano and as an orchestrator in the jazz idiom deeply ingrained in me the notion that two notes in harmony are somehow greater than the sum of their parts, but the aforementioned musical outlets have never rewarded me with the feeling of expressive freedom I experience when improvising on the saxophone. In my capacity as a single-line instrumentalist, a continuing bias to produce harmony leads me to experiment with technology as an extension of my saxophone playing.

Sound-on-sound looping, for example, allows me to slowly create harmonised passages through a process of layering. Pitch-shifters and diatonic harmonisers deliver a means of producing harmony in real-time, and to a reasonable extent offer real-time control over density, dynamics and timbre. In this case the quality has to be predetermined - a simple pitch-shifter will only produce a fixed interval voicing, and a diatonic harmoniser will only produce consonant harmony within a particular key system.

The aim of this research is to develop a chromatic interface for real-time harmonisation, and this paper will detail the process behind the development of one possible solution, the 'Harmonic Table Pitch Shifter' (HTPS), which consists of a custom MIDI footpedal and pitch-shifting audio plugin. It was important for the control hardware to be reliable in live performance, have a logical and easily navigable layout and allow a wide range of chord voicings to be selected with one or two feet. The processed audio output was to be capable of blending with the live instrument performance or contrasting as a distinct voice. Through a process of experimentation, improvisation and composition I offer conclusions regarding the viability, potential and limitations inherent to this approach.

In the context of this research, the term ‘chromatic’ is used to imply that all possible voicings should allow the performer to produce all possible voicings (limited only by range and number of voices) irrespective of their degree of consonance or dissonance. While extended saxophone techniques such as multiphonics allow a performer to produce multiple pitches at once, this research is limited to digital techniques for producing harmony based on twelve-tone equal temperament. The word ‘harmonisation’ is used throughout the paper to encompass all instances where multiple notes are sounded at once, including unisons and octaves. The harmonisation discussed is usually homophonic in nature, but in certain instances incorporating horizontal displacement or independent movement of parts I describe the effect produced as being polyphonic.

2. PITCH SHIFTERS AND HARMONISERS

2.1. Commercially available products

Digital pitch shifters have been commercially available since 1972, when Lexicon released their speech pathology-focused ‘Varispech’ processor [1]. Another early harmoniser was Eventide’s H910 unit, released in 1975 - possibly the first device marketed specifically for creating “musical harmonies” [2]. Harmonisation products can be grouped into three main categories, with some fitting into more than one category, depending on their feature sets.

The simplest variety is the fixed-voicing pitch shifter. It is capable of producing one or more additional voices shifted at fixed intervals with respect to a live input signal, resulting in a static voicing. These kinds of devices generally limit the performer to utilizing a single interval type or voicing at a time, making it impossible to control harmonised voices on-the-fly while performing. Contemporary examples usually include features from...