Public Sound Objects: a shared environment for networked music practice on the Web

ALVARO BARBOSA

Music Technology Group (MTG), Pompeu Fabra University, Calle Ocata 1, 08003 Barcelona, Spain
Research Center for Science and Technology of the Arts (CITAR), Portuguese Catholic University, Rua Diogo Botelho 1327, 4169-005 Porto, Portugal
E-mail: abarbosa@{iua.upf.es, porto.ucp.pt}

The *Public Sound Objects* (*PSOs*) project consists of the development of a networked musical system, which is an experimental framework to implement and test new concepts for online music communication. The *PSOs* project approaches the idea of collaborative musical performances over the Internet by aiming to go beyond the concept of using computer networks as a channel to connect performing spaces. This is achieved by exploring the internet's shared nature in order to provide a public musical space where anonymous users can meet and be found performing in collective sonic art pieces.

The system itself is an interface-decoupled musical instrument, in which a remote user interface and a sound processing engine reside with different hosts in an extreme scenario where a user can access the synthesizer from any place in the world using the World Wide Web. Specific software features were implemented in order to reduce the disruptive effects of network latency, such as dynamic adaptation of the musical tempo to communication latency measured in real time and consistent sound panning with the object's behaviour at the graphical user interface.

1. INTRODUCTION

Research work in the networked music field has recently been published in surveys by Álvaro Barbosa (2003), Gill Weinberg (2002) and Dante Tanzi (2001) which describe and categorise several different systems, following diverse architectures and practice contexts. Networked music can extend the boundaries of performance for practising musicians or in community practice performed by general and possibly non-musician users.

Connecting performing spaces for practising musicians to collaborate over long distances is the current popular approach that has facilitated most work performed and generally is implemented over a *peer-to-peer* architecture. Many examples of bilateral transmission of audio for geographically displaced musical events can be found, such as the Sensorband ISDN (Bongers 1998) concerts, the Hub remote concerts (Gresham-Lancaster 1998; Brown and Bischoff 2005) or the *Internet 2* multi-channel music sessions performed in several occasions over the United States and Canada (Woszczyk *et al.* 2005). Some other systems have been custom developed for similar collaborative scenarios, using control data for communication amongst performers instead of encoded digital audio streams. Examples of such systems include the *transMIDI*, which allows musical performers (and listeners) to play together or to organise into multiple session groups, by performing on their MIDI controllers, or a number of electronic music software implementations (MAX/MSP, PD, Reacktor, etc.) that support Mat Wright's Open Sound Control protocol (Wright and Freed 1997).

Commercial solutions addressing the professional recording and rehearsing studio paradigms, such as the early 1999 ResRocket Surfer (Moller *et al.* 1994) or the recent ejamming software (Nelson 2005), tend to incorporate both types of communication data (digital audio and MIDI), but since the intention of such systems is to serve general communities of users, a centralised shared system of community groups for textual communication (chats) is usually incorporated.

In fact, the idea of a shared sonic environment is bounded to public musical practice performed by general and possibly non-musician users on the Net. This is a recent approach to networked music and therefore existing examples are usually preliminary systems based on diverse sonic aesthetics and user interaction models; however, there is a tendency for implementations based on client-server topologies. This derives from the fact that a shared virtual environment is an ongoing instance wish that must be permanently available for users to login and interact with whoever is present at that moment. Examples of projects that follow this approach are Atau Tanaka's *mp3Q Piece* (Tanaka 2000), a shared online sound space on the Web that streams multiple channels of mp3 audio from different servers from which users can manipulate the sources by actuating over a graphical representation of the system behaviour, recent projects based on TransJam Server (Burk 2000), such as Phil Burke's *webdrum* (Burke 2000), Chris Brown's *Eternal Network Music* (Brown and Bischoff 2003), Max Neuhaus' *Oracle* (Neuhaus *et al.* 2005), and the *Public Sound Objects* project, designed since 2002 by the

Organised Sound **10**(3): 233–242 © 2005 Cambridge University Press. Printed in the United Kingdom. doi:10.1017/S135577180500097X

author at the Music Technology Group in Barcelona with the purpose of providing an experimental study framework for shared sonic environments in the context of the Interactive Systems Group research activities.

2. THE *PUBLIC SOUND OBJECTS* PROJECT

The *Public Sound Objects* (*PSOs*) project is web-based collaborative virtual environment focused on music performance, developed at the Music Technology Group of the Pompeu Fabra University. This project has provided an experimental framework in order to implement and test different approaches for online music communication. A preliminary specification of the system was published in Barbosa and Kaltenbrunner (2002), and the first prototype was implemented in December 2002. The *PSOs* system is publicly available online from the URL: http://www.iua.upf.es/~abarbosa/. Conceptually, it explores the notion of a shared Web space for community music creation, and that of an art installation, bringing together physical space and virtual presence in the Internet. The system aims to allow synchronous interaction providing a platform for sonic joint improvisation amongst Web users.

The overall system architecture was designed with respect to the following key factors: (i) it is based on a centralised server topology supporting multiple users connected simultaneously and communicating amongst themselves through sound; (ii) it is a permanent public event with special characteristics appealing both to a 'real world' audience and to an online virtual audience; (iii) the user interface and the sound synthesis engine offer a constrained sonic creation paradigm, which provides some coherence between the individual contributions; and (iv) the system is scalable and modular allowing future expansion and different set-ups.

In this system the raw materials provided to the users for manipulation during a performance are *sound objects*. The definition of a sound object as a relevant element of the music creation process goes back to the early 1960s (Schaeffer 1966). Schaeffer defined a sound object as 'any sound phenomenon or event perceived as a coherent whole (. . .) regardless of its source or meaning' (Chion 1983). From a psychoacoustic and perceptual point of view, Schaefer's definition is extremely useful, since it provides a very powerful paradigm to sculpt the symbolic value conveyed in a sonic piece.

In the *PSOs* system a server-side real-time sound synthesizer triggers a sound object according to the user's action. Since the feedback from other user's performance is strictly auditory, the characteristic which makes a sound object distinguishable from the overall *soundscape* is the key element that permits the awareness of the individual action of a user over his sound object.

3. THE *PSOS* ARCHITECTURE

The *PSOs* system is composed of the *PSOs* server and multiple *PSOs* clients. Clients control a visual interactive interface, while the server controls all computation regarding the sound synthesis and transformation. It is an extreme example of an interface-decoupled application where the synthesis engine is separated from the user interface over a large area network (Barbosa, Kaltenbrunner and Geiger 2003).

Clients communicate with the server through HTTP by sending and receiving packets of data. There are several types of data packets that the clients can send but the most important ones are the *ImpactPacket* – which informs the server that the bouncing ball has hit one of the walls; the *ControlPacket* – which tells the server that the user has changed the value of one of the interface controls; and the *PingPacket* – which is used to measure the network delay between the client and the server.

The server packets are received by a Web application that reroutes them to the interaction server – a module of the *PSOs* server that manages clients, instruments and the events generated by the *PSOs* client. Depending upon the type of data packet received, a sound can be generated by the synthesis and transformation engine and then streamed back to the client by the streaming audio server, or the visual representation of the client can be updated at the installation site by the local visual representation engine, or both.

Server and clients are of different modules:

3.1. HTTP server

Clients connect to the *PSOs* server through standard HTTP connections. Although our initial choice was to implement UDP-based communications – faster than a TCP-based protocol like HTTP – the idea had to be abandoned for two main reasons:

- Most firewalls block all unknown UDP traffic, which would mean that a great number of users would not be able to access our server, also increasing the difficulty of deploying the *PSOs* server for the same reasons: UDP traffic would have to be allowed at a specific port by the firewall.
- Some browsers' security policies for Java applets only allow them to make connections using the HTTP protocol.

In order to overcome these restrictions, a communication system was realised using a 'firewall generally allow' protocol: HTTP. For this a server application

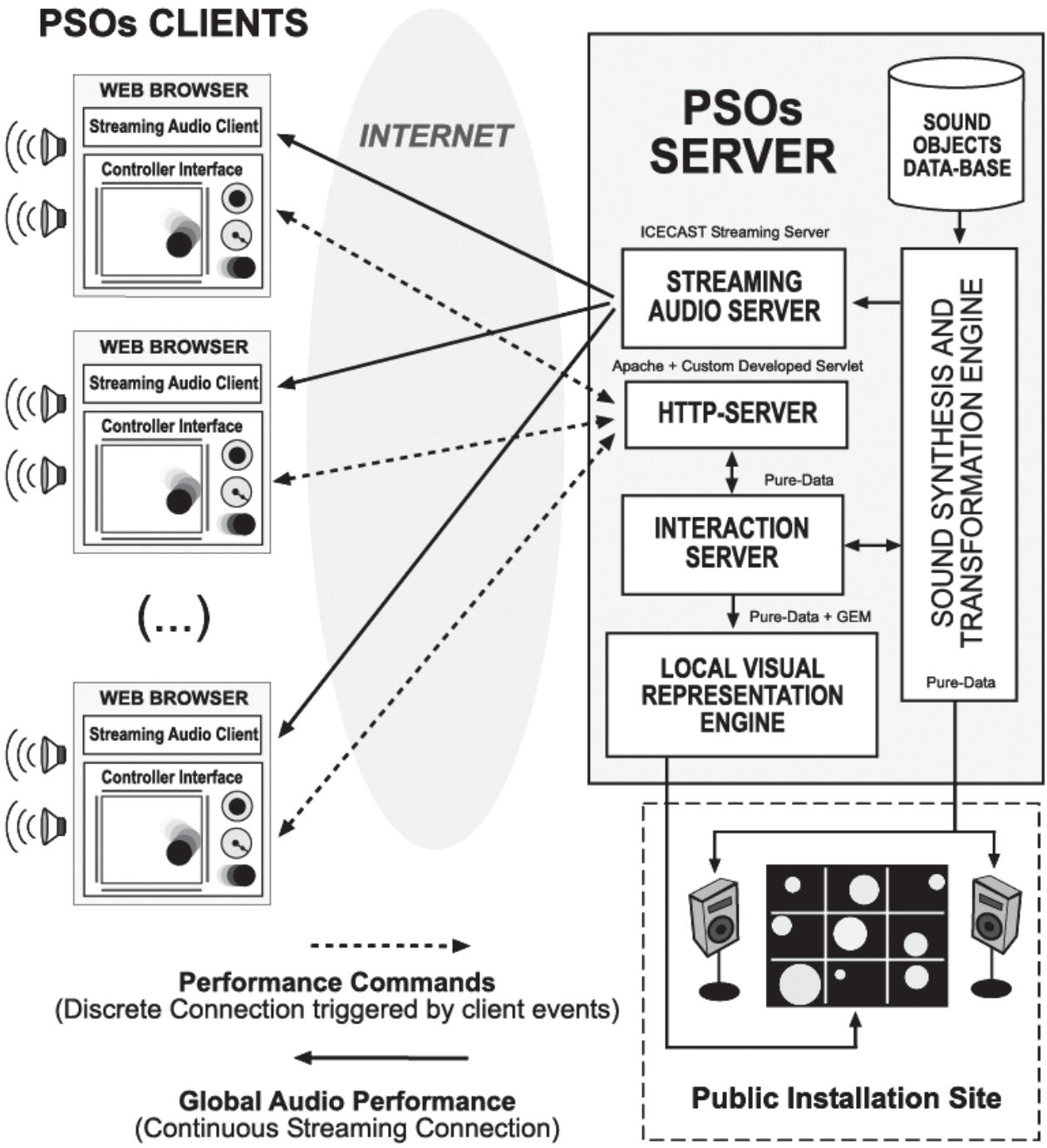

Figure 1. The *PSOs* system architecture.

was implemented, using the Java servlet technology, which acts as a proxy between the *PSOs* client applet and the interaction server. Basically, this servlet just passes data received from the *PSOs* client to the interaction server and vice versa.

3.2. Interaction server

The interaction server is a central piece in the *PSOs* server. It's a *pure data* (PD) module that receives data packets in the form of UDP datagrams from the clients (through the HTTP Server) and acts accordingly to the type of packet received.

A custom PD object had to be implemented for the reception of the UDP datagrams – which was called *extended netreceive* [xnetreceive] – since existing objects for this purpose don't allow PD to acquire the IP address and port number of the client that initiated the communication. The packet types defined so far are as follows:

- *AvailableInstruments*. When the interaction server receives this type of packet it sends as response the numbers of the instruments that are available. Instruments were numbered 1 to 9.
- *LockInstrument*. This type of packet is sent to the server when the user chooses one instrument to play. The instrument number is specified in the packet. The interaction server will check that the instrument is still available and will respond with a true/false result depending on whether the instrument was successfully locked or not. When an instrument is locked it can only be used by the client that locked it.
- *UnlockInstrument*. Informs the server that the user is done with the instrument specified by the instrument number in the data packet. The interaction server will unlock the instrument, which will then become available to other clients.
- *ImpactPacket*. This is the most used packet. It tells the server that the bouncing ball has hit a wall and that a sound should be generated. The interaction server passes these packets along to the synthesis and transformation to the local visual representation engines. Among other information, these packets specify the instrument number, the value of the wall sliders, the speed of the ball, the ball's size, the wall that was hit, what point of the wall was hit, and the size of the ball's trail. This information is then used by the synthesis and transformation engine to generate a sound according to the parameters set by the user in the *PSOs* client interface. It is also used by the local visual representation engine to update the visual representation of that user.
- *ControlPacket*. This type of packet is of interest only to the local visual representation engine. The information that is sent is the same as the *ImpactPacket* but the events that trigger transmission are different. *ControlPackets* are sent whenever the user changes the speed, size or trail size of the bouncing ball. The interaction server passes these packets along to the local visual representation engine so that the installation site can be updated.

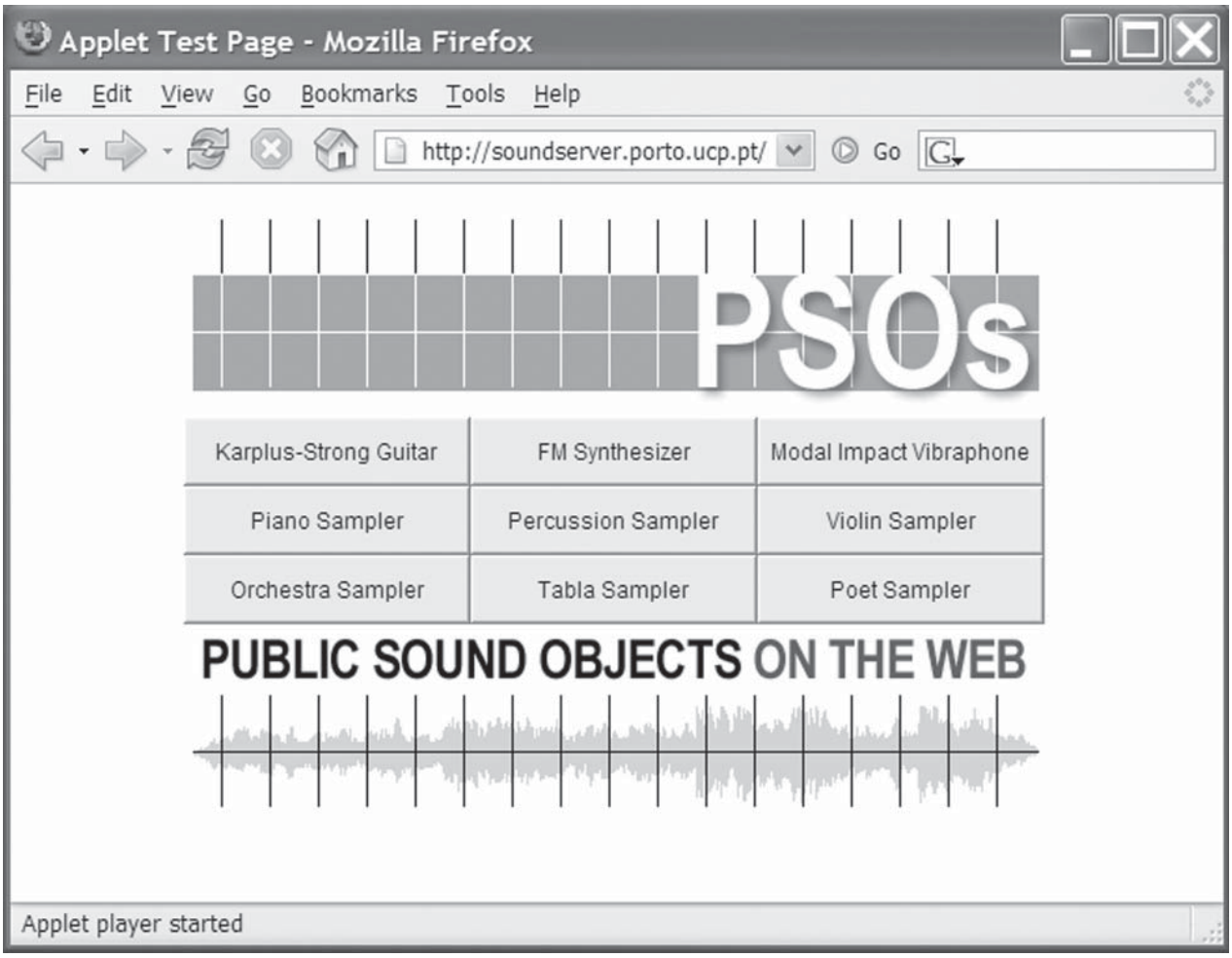

Figure 2. PSOs client entry screen.

- *PingPacket*. These packets hold no direct information; their sole purpose is to allow the *PSOs* client to determine the network delay between the client and the server. The interaction server merely sends back an empty reply to the client.

The other main task of the interaction server is to manage the connected *PSOs* clients. If a client gets disconnected from the network without having sent an *UnlockPacket*, the instrument currently locked by that client would never again be available. It is the job of the interaction server to detect such situations and to automatically free up the instrument. This is done with timeouts, i.e. if a client remains more than a fixed amount of time without contacting the server that client is removed from the list of currently connected clients and its instrument released.

3.3. Synthesis and transformation engine

The synthesis and transformation engine is responsible for the sound generation in response to the *PSOs* clients' generated events. This engine is a PD patch automatically loaded by the interaction server. It receives *ImpactPackets* from the interaction server (PD lists) and generates a sound according to the values specified therein. The parameters taken from these data packets are actually passed on to one of nine synthesis modules.

At this time, the engine has nine modules that correspond to the nine instruments available to users. Since each module is different and independent, the same parameter can have a different meaning for different modules. These modules are:

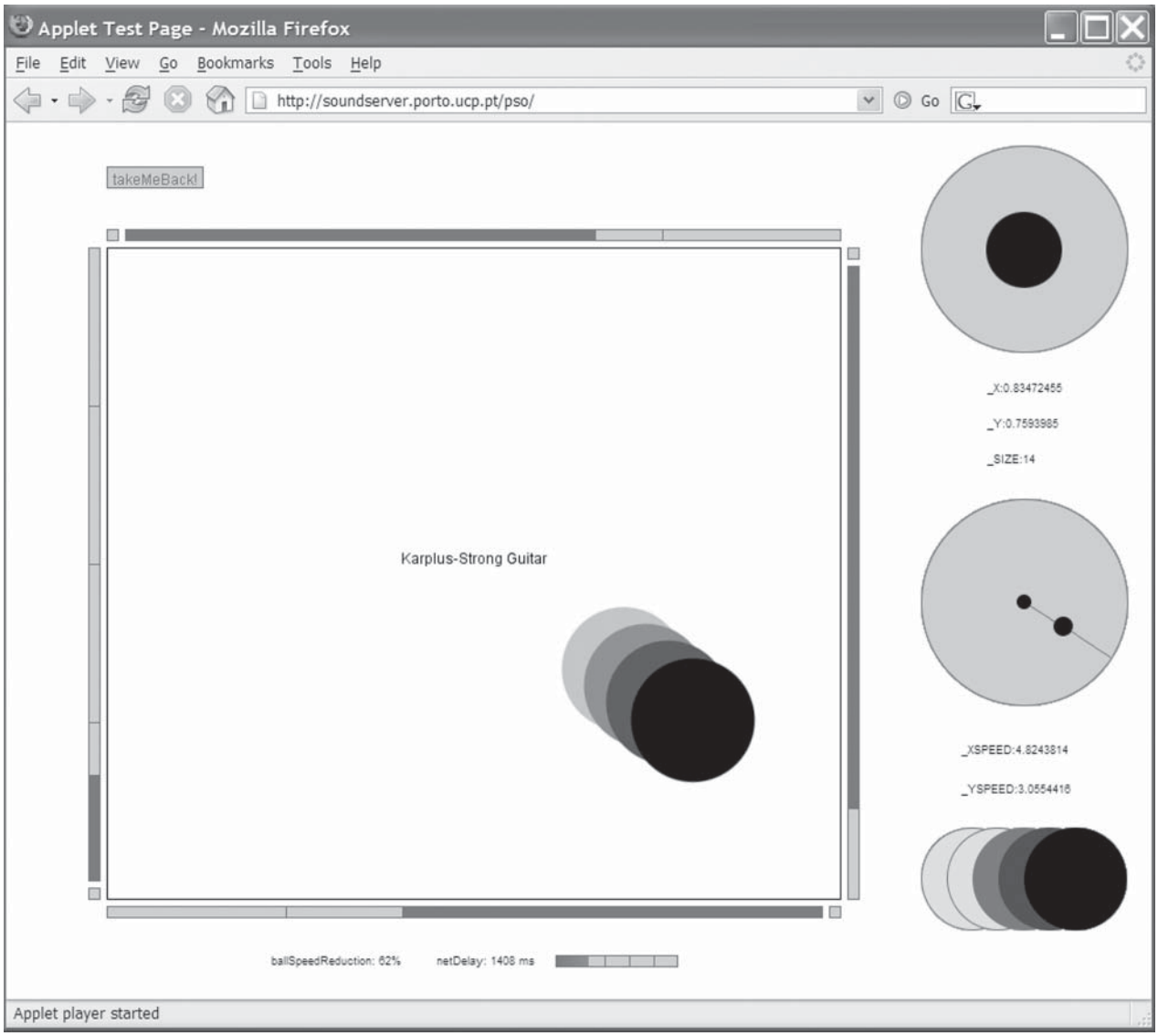

Figure 3. *PSOs* client controller interface.

- *Karplus-Strong Guitar*. As the name suggests, this is an implementation of the Karplus-Strong algorithm for a plucked string sound implemented in PD.
- *FM Synthesizer*. A frequency modulation synthesizer.
- *Modal Impact Vibraphone*. An attempt to produce vibraphone-like sounds using modal impact physical models implemented for PD, available from the *Sounding Objects* project (Rocheso and Fontana 2003).
- *Piano, Percussion, Violin, Orchestra, Tabla and Poet Samplers*. These are in fact only one module, loaded with six different sounds. The sampler was implemented in PD and used six voices, which proved to be enough not to overload the system, for the worst-case scenarios (nine users connected with high-tempo performances).

The sound generated by these modules is streamed in MP3 format, using the [shoutcast~] PD object to an audio streaming server. We use the Icecast2 streaming server for Windows. Each user can choose one of these modules as the sound-generating engine from the *PSOs* entry screen at the client instance. Upon loading the entry Web page, if a sound module is taken by another user, its button on screen will be off and the module will only be available when it is released.

3.4. The client user interface

Once the sound generating engine (instrument) has been selected, the Web browser loads the controller interface applet, which connects to the interaction server, registers and initialises a user session. The graphical user interface (GUI) can differ in future developments.

A *PSOs* GUI implementation is developed to meet the following requirements: (i) it should enable the user to contribute to the ongoing musical performance by transforming the characteristics of a visual sound object representation, sending normalised parameters to the synthesis engine over the network; (ii) the interface application should be able to allow manipulation of each of the modifiers' parameters in the synthesis engine in articulation with the specific installation site set-up; (iii) the GUI itself should be a behaviour-driven metaphorical interface, avoiding a flat mapping of parameters, such as faders or knobs, since providing automatic periodical behaviour for the graphic objects as a sound controller will allow a larger timescale in the user action, which tends to be more appropriate for a system with delayed acoustic feedback.

The current implementation, presented in figure 4, follows a metaphor of a ball that infinitely bounces on the walls of an empty room. When the ball hits one of the walls, a network message is sent to the central

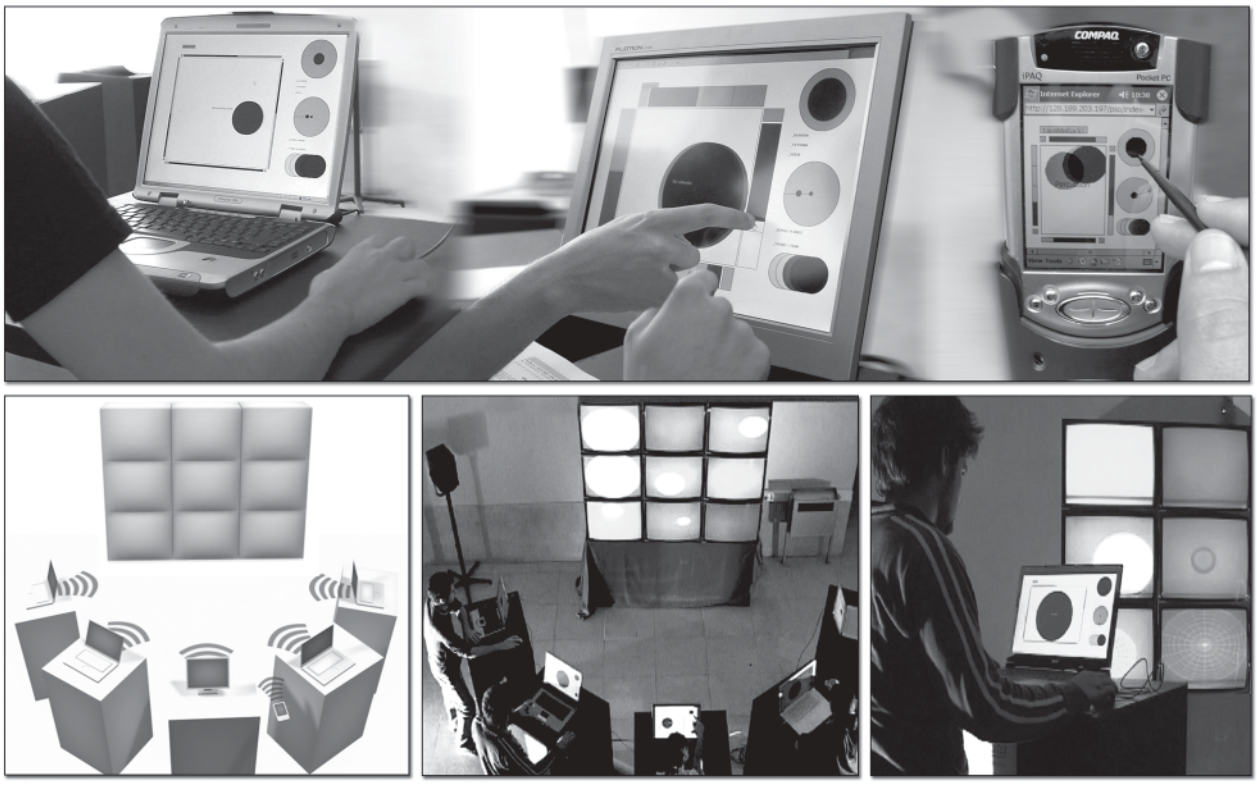

Figure 4. Different interfaces for desktop, touch-screen and PDA hardware clients; The *PSOs* installation site graphic prototype and photos from the installation at Porto School of the Arts in October 2004.

server where the corresponding sound object is triggered, played through a specific source speaker and simultaneously streamed back to the user in a stereo mix of all the sounds being triggered at the moment.

The ball moves continuously and the user can manipulate its size, speed, direction, tail extension and each wall's acoustic texture. Values are then sent to the server and mapped to synthesis parameters. The wall's acoustic texture matches the sound object's pitch (individual pitch values can be assigned to each wall, allowing the creation of melodic and rhythmic sound structures) and the ball's tail extension corresponds to the number of replicas of the delay applied to the sound object.

3.5. Local visual representation engine

The local visual representation engine outputs the visual representation of the bouncing ball model of all the connected *PSOs* clients, at the server's physical location. It consists of a PD patch that uses the graphics environment for multimedia (GEM) external for graphics output, using information from *ImpactPackets* and *ControlPackets* to update the state information for each client.

The visual set-up is composed of a video wall with nine screens arranged in a 3×3 matrix and by local installation of client instances with adapted 'bouncing ball' interfaces for desktop computers, touch screens and mobile PDAs. Each screen from the video wall is assigned to an instrument in the same order that they appear to the user in the *PSOs* client interface. The clients are represented at the installation site as spheres with different colours, sizes and speed. Each client is assigned to a screen in the video wall which also limits the movement of the corresponding sphere, i.e. the limits of each screen are mapped to the limits of the *PSOs* client's window. Whenever a new client connects, a colour is randomly chosen to represent their ball.

Two more parameters were chosen to provide visual feedback: the speed of the ball and the events generated at the client's interface. Although there is an implicit visual feedback on the ball's speed, i.e. the sphere moves faster or slower on the screen, an additional feedback was added by changing the saturation of the sphere's colour. Sometimes when the bouncing ball is set to a large size and occupies almost the whole screen, it is hard to tell its speed because both a slow ball and a fast one will bounce a lot. Mapping the speed to the colour saturation – high saturation for a slow ball, low saturation for a fast one – helps viewers to distinguish these situations. When the local visual representation engine receives a packet, meaning that an event was fired at the *PSOs* client's interface, the client's sphere is temporarily turned into a polygon mesh representation.

The engine only has accurate information when clients send packets to the server. The rest of the time, the position of the bouncing ball has to be interpolated based on the information from the last packet. It is not possible to have a completely accurate representation of the user's bouncing ball due to network latency, different timing mechanisms on the clients and server, and because we cannot predict the user's actions. Despite all this, it is possible to get a fairly good representation of the various clients. The most noticeable representation artefact is the occasional 'jump' of the sphere, e.g. sometimes the representation is changing more rapidly than the client's bouncing ball, so when a packet is received the position is suddenly updated to the correct one causing the sphere to 'jump' back.

3.6. Network latency adapted tempo

When established over long distance, networked music systems have a common problem. Network latency (or Net-delay) is an impediment for real-time musical communication. Using the laws of physics it can be demonstrated that for distances that span continents, current network technology will always introduce higher latency than the minimum acceptable values for real-time acoustic collaboration. A number of experiments have been carried out with the purpose of determining the maximum amount of communication latency which can be tolerated between musicians in order to keep up with a synchronous performance. Concrete results from research carried out in 2002 at Stanford University by Natham Shuett (Schuett 2002) established experimentally an ensemble performance threshold (EPT) for impulsive rhythmic music lying between 20 and 30 ms, which is consistent with the outcome from research carried out by Nelson Lago in 2004 (Lago and Kon 2004) at São Paulo University.

Some results regarding the effects of time delay on ensemble accuracy, which go beyond establishing an EPT for a general scenario of rhythmic synchronisation, were published in 2004 by Chris Chafe and Michael Gurevish (Chafe *et al.* 2004). From the experiment conducted at CCRMA it is clear that by increasing the communication delay between pairs of subjects trying to synchronise a clapping steady rhythm, the subjects tend to slow down the tempo of the rhythm.

Similarly, an experiment carried out by the authors in June 2004 in the Sound and Image Department at the Portuguese Catholic University aimed, amongst other goals, to study the relationship between tempo and latency. In the experiment, simulated network latency conditions were applied to the performance of four different musicians playing jazz standard tunes with four different instruments (bass, percussion, piano and guitar). In a studio set-up, musicians would listen to the feedback from their own instruments

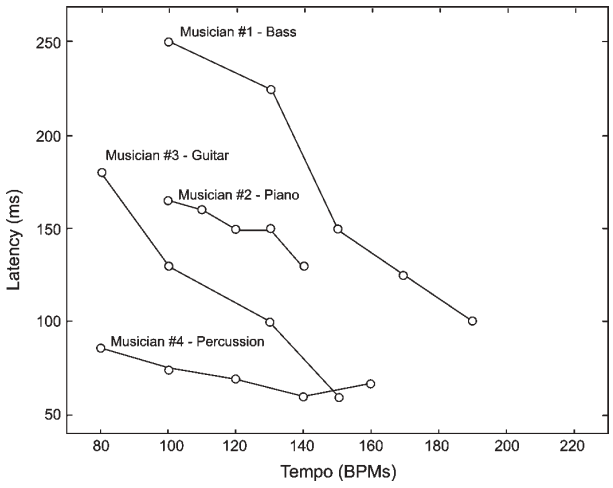

Figure 5. Self-test for latency tolerance in individual performance of four different musicians.

through headphones with delay. Their performance was synchronised with a metronome over several takes with different tempos (beats per minute – BPM). For each take the feedback delay was increased until the musician was not able to keep up a synchronous performance.

The figure 5 graphic shows that, regardless of the instrumental skills or the musical instrument, all musicians were able to tolerate more feedback delay for slower tempos. The only exception to this tendency occurs for the percussionist's curve when raising to 160 BPM, which is related to a synchronous overlap over the rhythmic structure of the music, together with the fact that for percussion instruments it is very hard to totally isolate the performer from the direct instrument sound; therefore, it is clear that there is an inverse relationship between tempo and latency. After obtaining these results the authors proceeded to integrate this concept into the *Public Sound Objects* (*PSOs*) system, aiming to achieve a network-music instrument that incorporates latency as a software feature, by dynamically adapting its tempo to the communication delay measured in real time.

The idea of a network music instrument which dynamically adapts to Internet network latency was implemented recently by Jörg Stelken in the peerSynth software (Stelkens 2003). peerSynth is a peer-to-peer sound synthesizer which supports multiple users displaced over the Internet, measuring the latency between each active connection and dynamically lowering the sound volume of each user's contribution to the incoming *soundscape*, proportionally to the amount of delay measured in his connection. Stelkens followed a real-world metaphor where, in fact, the sound volume of a sound source decreases with the distance to the receiver, which also implies increasing acoustical communication latency. A similar approach was followed in the AALIVENET System (Spicer 2004).

The PSOSs system approaches this same idea, addressing a less immediate, but equally relevant, relation between musical characteristics and communication latency. It implements a network tempo adaptive latency feature, which dynamically reduces the performance tempo according to the latency measured in real time between the client and the server.

In the bouncing ball user interface the musical tempo corresponds to the ball speed. The reduction factor applied to the ball speed is presented in the user interface and it is calculated so that it averagely guarantees that the ball will not hit the walls twice without the sound being triggered by the time the first hit arrives at the client. The main idea is that *the ball will go as fast as your connection allows it to go*. This way the effect of latency is much less confusing, permitting the user to have a much better awareness of the relationship between a hit on the wall and the corresponding triggered sound.

Additionally, sound panning consistent with the object's behaviour at the graphical user interface was introduced so that whenever a ball hits a left or right wall the server triggers the corresponding sound object through the left or right speaker, and when it hits the top or bottom wall the sound object is triggered through both speakers (central pan). This also helps to subconsciously clarify the order of immediate corresponding hits and sounds, especially when the ball bounces on corners.

4. *PSOS* USER STUDY AND EVALUATION

The complete *PSOs* system, including the physical set-up at the server site, was installed at the Portuguese Catholic University Campus in Porto between 7 and 14 October 2004. During this trial period several client instances were installed on campus and 109 subjects tested the system and answered questionnaires. Some of the average results extracted from this opinion pool are presented in figure 6.

Some of these results met our expectations: (i) the interface is effective at establishing a relation between the user action and its effect on the corresponding sound object; (ii) the sound objects available in the current set-up allow acoustic differentiation in the global *soundscape*; and (iii) it is a system accessible to the general public, without requiring previous music formation or previous GUI manipulation skills.

When asked for further comments about the system, most of the users mentioned that the system is clearly about 'experimental music' and the influence of other users' performances, as expressed by the following quotes: 'The system only seems to make sense when used collectively'; 'It is simple to be aware of the other

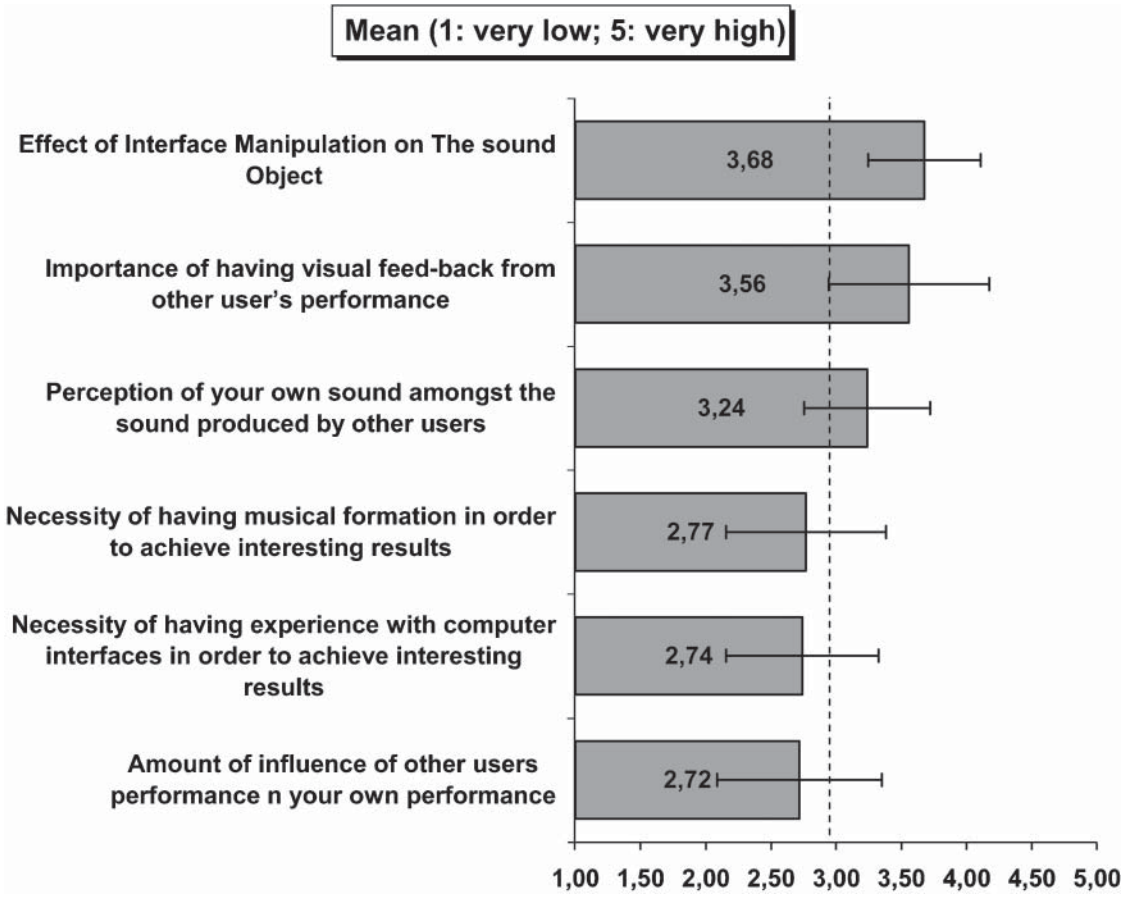

Figure 6. *PSOs* preliminary evaluation results.

users' actions by looking into the video-wall and use it as reference when needed'; 'Interacting with other users makes you achieve different results than you would get by yourself'. From these quotes and from the full statistical corpus it seems that in general users found the visual representation of other users' behaviour useful, to enhance their own acoustic awareness.

5. CONCLUSIONS AND FUTURE WORK

For over two years the *Public Sound Objects* project has successfully functioned as an experimental framework to implement and test different approaches for online music communication. The system is intended to be simple enough so that non-professional musicians can engage in a collaborative sonic performance, and in this sense the recent user study provided confirmation that this goal has been achieved. More than half of the sample users considered that no musical training or experience manipulating computer interactive interfaces was required in order to achieve interesting results.

The System has a fast learning curve, since 90.6 per cent of the sample users learned how to use the system in less then five minutes and 41.1 per cent of this group in less than a minute. This was mainly due to the choice of experimental sound art as the musical aesthetic of the project and the fact that music is generated algorithmically, in the sense that even without a user's interference the 'bouncing ball' will trigger an endless sequence of sounds with random tempo and pitch.

In addition, even though the musical results of the system do not have familiar rhythmic or melodic structures, the control parameters which the 'bouncing ball' interface present are pitch, tempo, dynamics and delay, which are very basic traditional music control parameters, and therefore fit better into what a regular user would expect.

Further improvements are suggested by the evaluations of the sample users regarding visual representation of other users at the client side, and in fact this was the impression one received by performing at the installation site, since it is inevitable to correlate the visual representation of other users and their acoustic contributions to the piece.

The network tempo adaptive latency and the coherent sound panning, implemented in the *PSOs* latest version, represent a significant improvement in

the system usability regarding the disrupting effect of latency.

ACKNOWLEDGEMENTS

The Authors would like to thank Xavier Serra and Sergi Jordà for their guidance and advice in the project, Jorge Cardoso, Gunter Geiger and Martin Kaltenbrunner for their work in the development of the *PSOs* system, and Alexander Carôt for his collaboration in the delay *vs* tempo experiment. This research is supported by the Portuguese institution Fundação para a Ciência e Tecnologia.

REFERENCES

Auracle. 2005. *Akademie Schloss Solitude*. Developed by Max Neuhaus, Phil Burk, Jason Freeman, C. Ramakrishnan and Kristjan Varnik. http://www.auracle.org/, accessed on 20 June 2005.

Barbosa, A. 2003. Displaced Soundscapes: a survey of network systems for music and sonic art creation. *Leonardo Music Journal* **13**: 53–9.

Barbosa, A., and Kaltenbrunner, M. 2002. Public Sound Objects: a shared musical space on the web. In *Proc. of the Int. Conf. on Web Delivering of Music (WEDELMUSIC 2002)*. Darmstadt, Germany: IEEE Computer Society Press.

Barbosa, A., Kaltenbrunner, M., and Geiger, G. 2003. Interface decoupled applications for geographically displaced collaboration in music. In *Proc. of the Int. Computer Music Conf. (ICMC2003)*.

Bongers, B. 1998. An interview with Sensorband. *Computer Music Journal* **22**: 13–24.

Brown, C., and Bischoff, J. 2005. Computer Network Music Bands: A history of the League of Automatic Music Composers and the Hub. In A. Chandler and N. Neumark (eds.) *At a Distance: Precursors to Art and Activism on the Internet*, pp. 372–91. Cambridge, MA: MIT Press.

Burk, P. 2000. Jammin' on the Web: a new client/server architecture for multi-user musical performance. In *Proc. of the Int. Computer Music Conf. (ICMC 2000)*.

Chafe, C., Gurevich, M., Leslie, G., and Tyan, S. 2004. Effect of time delay on ensemble accuracy. In *Proc. of the Int. Symp. on Musical Acoustics (ISMA2004)*. Nara, Japan.

Chion, M. 1983. *Guide des objets sonores. Pierre Schaeffer et la reserche musicale*.

EJamming. 2005. Developed by Tom Nelson. http://www.ejamming.com/, Accessed 27 June 2005.

Eternal Network Music. 2003. Crossfade, developed by Chris Brown and John Bischoff. http://www.transjam.com/eternal/eternal_client.html, accessed on 21 June 2005.

Gresham-Lancaster, S. 1998. The aesthetic and history of the Hub: the effects of changing technology on network computer music. *Leonardo Music Journal* **8**: 39–44.

Lago, N., and Kon, F. 2004. The quest for low latency. *Proc. of the Int. Computer Music Conf. (ICMC2004)*, pp. 33–6.

MP3Q. 2000. Developed by Atau Tanaka. http://fals.ch/Dx/atau/mp3q/, accessed 12 May 2004.

ResRocket Surfer. 1994. Rocket Networks, developed by M. Moller, W. Henshall, T. Bran and C. Becker. http://www.resrocket.com/, accessed 17 April 1999.

Rocheso, D., and Fontana, F. 2003. *The Sounding Object*.

Schaeffer, P. 1966. *Traité des objets musicaux*.

Schuett, N. 2002. *The Effects of Latency on Ensemble Performance*. Stanford University.

Spicer, M. 2004. AALIVENET: an agent based distributed interactive composition environment. In *Proc. of the Int. Computer Music Conf. (ICMC2004)*.

Stelkens, J. 2003. peerSynth: a P2P multi-user software with new techniques for integrating latency in real time collaboration. In *Proc. of the Int. Computer Music Conf. (ICMC2003)*.

Tanzi, D. 2001. Observations about music and decentralized environments. *Leonardo Music Journal* **34**: 431–6.

Webdrum. 2000. SoftSynth, developed by Phil Burke. http://www.transjam.com/webdrum/, accessed on 21 June 2005.

Weinberg, G. 2002. The aesthetics, history, and future challenges of interconnected music networks. *Proc. of the Int. Computer Music Conf.*, pp. 349–56.

Woszczyk, W., Cooperstock, J., Roston, J., and Martens, W. 2005. Shake, Rattle and Roll: getting immersed in multisensory, interactive music via broadband networks. *Journal of the Audio Engineering Society* **53**: 336–44.

Wright, M., and Freed, A. 1997. Open SoundControl: a new protocol for communicating with sound synthesizers. In *Proc. of the Int. Computer Music Conf.*