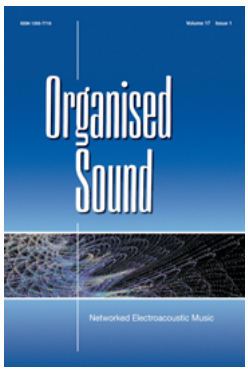

Building on the Foundations of Network Music: Exploring interaction contexts and shared robotic instruments

Owen Vallis, Dimitri Diakopoulos, Jordan Hochenbaum and Ajay Kapur

Organised Sound / Volume 17 / Issue 01 / April 2012, pp 62 - 72
DOI: 10.1017/S1355771811000525, Published online: 14 February 2012

Building on the Foundations of Network Music: Exploring interaction contexts and shared robotic instruments*

OWEN VALLIS[†], DIMITRI DIAKOPOULOS[‡], JORDAN HOCHENBAUM[§] and AJAY KAPUR[¶]

[†]New Zealand School of Music, Victoria University, PO Box 2332, Wellington, New Zealand;
[§,‡]California Institute of the Arts, 24700 McBean Parkway, Valencia, CA 91355, USA
E-mail: [†]vallisowen@myvuw.ac.nz; [‡]ddiakopoulos@calarts.edu; [§]hochenjord@myvuw.ac.nz; [¶]akapur@calarts.edu

Historically, network music has explored the practice and theory of interconnectivity, utilising the network itself as a creative instrument. The Machine Orchestra (TMO) has extended this historical idea by developing the custom software suite *Signal*, and creating a shared, social instrument consisting of musical robotics. *Signal* is a framework for musical synchronisation and data sharing, designed to support the use of musical robotics in an attempt to more fully address ideas of interconnectivity and embodied performance. *Signal*, in combination with musical robotics, also facilitates the exploration of interaction contexts, such as at the note level, score level and sound-processing level. In this way, TMO is simultaneously building upon the historical contributions and developing aesthetics of network music.

1. INTRODUCTION

The Machine Orchestra (TMO) stands on the shoulders of historic networked ensembles such as The League of Automatic Composers and The Hub, as well as contemporary ensembles such as PLork and SLork (Bischoff, Gold and Horton 1978; Gresham-Lancaster 1998; Smallwood, Trueman, Cook and Wang 2008). Informed by the aesthetic and technical contributions of each of these ensembles, we have undergone several iterations of an *ad hoc* cycle, leading to the integration of a new network element: a shared instrument consisting of musical robotics. Through an ongoing process of music composition, music performance, evaluation and software development, we offer several new insights regarding the assimilation of robotic performers into a networked ensemble. These insights are realised in our framework *Signal*, and were partly inspired by two contributions to the field of network music: Gil Weinberg's seminal work on network interactions (Weinberg 2005), and Álvaro Barbosa's concept of a shared social instruments (Barbosa 2003). We offer *Signal* as a case study in overcoming many of the technical challenges of modern network with general emphasis on stabalising network synchronisation, and integrating musical robotics.

We begin by presenting a historical context of networked performance, illustrating how *Signal* is influenced by, and built upon, the successes of many previous systems. We describe how artists and ensembles, both old and new, have dealt with issues regarding communication and embodied performance. Next, we give a detailed technical overview of *Signal*'s implementation. This section discusses the framework's network topology, client/server architecture, general usability, and several implementation decisions made about sync and communication protocols. The final section highlights three compositions that use *Signal* to establish and explain the direct results of our development cycle. Each piece was specifically chosen to demonstrate how an iteratively designed system enables networked music composition and performance within multiple interaction contexts.

2. RELATED WORK

Since the mid-1970s, musical networks have evolved into a rich and diverse field of research and exploration for computer musicians. The idea of a musical network has grown to encompass a wide range of topologies and configurations, from small local networks of performers sharing data with each other (Bischoff et al. 1978; Gresham-Lancaster 1998; Smallwood et al. 2008), to large ensembles of computer musicians half a world apart (Cáceres, Hamilton, Iyer, Chafe and Wang 2008). During this same time, research has also explored systems designed to overcome the challenges inherent in high-latency, low-bandwidth communication (Lazzaro and Wawrzynek 2001; Chafe and Gurevich 2004; Barbosa, Cardoso and Geiger 2005; Cáceres and Chafe 2010; Whalley 2010; Driessen, Darcie and Pillay 2011).

*The authors would like to thank all the computer network ensembles that have laid the foundation on which we have built our research. We would also like to thank Perry Cook, Curtis Bahn, Charlie Burgin, Meason Wiley and Jim Murphy for their exceptional musicianship in realising the pieces described in this article. Additionally, we would like to thank the reviewers at *Organised Sound* and Rory Vallis for their efforts in helping with the realisation of this article.

2.1. Network data sharing

Founding artists were interested in the potential of networks to connect and share data. Early experiments by The League of Automatic Composers (Bischoff et al. 1978) involved three networked microcomputers (KIM-1), each with its own custom software instrument, all sharing control data. This process of allowing the performers to control each other's instruments created an ensemble that had never before been possible, and led to the creation of a shared and social instrument, diffusing the absolute control a musician traditionally had over his or her own instrument. The members of The League named this new style of music 'Network Computer Music', and continued to explore the possibilities of this new style until 1986.

Out of the pioneering efforts of The League, The Hub was formed, and expanded on existing research using the newly developed MIDI protocol. In his 1998 article on the aesthetics and history of The Hub, Scot Gresham-Lancaster explains how 'the advent of both the microprocessor and the affordable, multi-parameter, controllable MIDI synthesizer made possible a new type of network-based performance' (Gresham-Lancaster 1998: 39). He goes on to suggest a link between network music and the process music of composers such as John Cage, David Tudor and Pauline Oliveros. In networked computer music, the processes that Lancaster describes arise out of the rules governing the ways in which performers share data through various network topologies and algorithms.

The potential for performers to share musical data through networks has become one of the central focuses of networked computer music, with Brian Kane going so far as to say 'Any aesthetics of Net music would, correspondingly, imply a set of musical practices that exploit these (and other) specific affordances of networks' (Kane 2007: 2). Additionally, while developing a classification framework for describing the multitude of possible networked ensemble interconnections, Gil Weinberg states that he attempted 'to map the field based on what [he sees] as the central innovative concept of the medium: the level of interconnectivity among players and the role of the computer in enhancing the interdependent social relations' (Weinberg 2005: 26). In developing his framework, Weinberg changed the name of networked computer music to interconnected music networks, reflecting this focus on interconnectivity.

This focus can be found in many different networked music compositions, past and present. The Hub's piece *Waxlips* (Tim Perkis, 1991) is based around a rule regarding the way a client requested notes from the master machine, and in doing so is 'an attempt to find the simplest Hub piece possible, to minimize the amount of musical structure planned in advance, in order to allow any emergent structure out of the group interaction to be revealed clearly' (Gresham-Lancaster 1998:42). While in pieces such as Clix (Ge Wang), The PLOrk Tree (Dan Trueman), and PLahara (Dan Trueman), Princeton's Laptop Orchestra PLOrk (Smallwood et al. 2008) examines the way in which data can be passed around a network using a wireless router and topologies as complex as peer-to-peer interconnectivity.

2.2. Challenges of embodiment

While these new methods of interconnectivity create experimental ways for performers to socially produce and consume musical data, they have also introduced new challenges. Complex networking schemes allow for the sharing of performance data between musicians; however, the connection between the music and the listeners – both ensemble participants and audience members – is potentially diffuse and unfocused. Weinberg describes this as one of 'the field's main drawbacks, in [his] opinion, stem[ing] from the focus that was put on complex interdependent connections which forced participants and audiences to concentrate on low-level analytical elements in order to follow the interaction' (Weinberg 2002: 6). He goes on to say, in a later article, 'these networks posed high entrance barriers for players by requiring specialized musical skills and theoretical knowledge in order to take part in and follow the interaction in a meaningful manner' (Weinberg 2005: 28). One of the original members of The Hub also expresses a similar idea, stating, 'the audience was often mystified by what they heard in relation to what they saw the performers doing' (Gresham-Lancaster 1998: 43).

Weinberg suggests that 'the design of expressive gesture-based interconnected instruments [would provide] participants with an expressive as well as coherent access to complex interdependent network topologies, which will allow them to focus on the artistic aspects of the experiences' (Weinberg 2002: 6). The instruments proposed by Weinberg would provide a strongly embodied link between the performers' actions and the music created. The Machine Orchestra's implementation of this idea was in the conception of a shared, social instrument comprising an array of custom built electromechanical instruments.

2.3. Shared social instruments

Analogous to several performers playing on a single piano, the shared instrument allows for multiple performers to express themselves independently within a social context (Barbosa 2003). The Machine Orchestra has created this kind of shared, social instrument using musical robotics which are accessible by every member of the ensemble, serving to directly embody the

actions of the group through the physical movements of the robotic actuators (Kapur, Darling, Diakopoulos, Murphy, Hochenbaum, Vallis and Bahn 2011). While this is similar to traditional acoustic ensembles, with each musician performing independently of each other but also as a group, it is also acutely different. Networked ensembles afford computer musicians the ability to exist in multiple interaction contexts in a way that is difficult for acoustic instruments to achieve.

2.4. Interaction contexts

Modern computers afford performers the ability to fluidly shift between different interaction contexts, creating a rich and expressive improvisation space. This would be similar to every member of an orchestra playing their instrument, while being able to conduct their own small section of the ensemble, and simultaneously affecting the sound reinforcement within the performance space. Wanderley and Orio describe seven of these interaction contexts, with the first three having relevance to The Machine Orchestra:

1. Note-level control, or musical instrument manipulation (performer-instrument interaction), i.e., the real-time gestural control of sound synthesis parameters, which may affect basic sound features as pitch, loudness and timbre.
2. Score-level control, for instance, a conductor's baton used to control features to be applied to a previously defined – possibly computer generated – sequence.
3. Sound processing control, or post production activities, where digital audio effects or sound spatialization of a live performance are controlled in real time, typically with live-electronics. (Wanderley and Orio 2002: 69)

However, in order for individuals within an ensemble to take advantage of these different contexts, there needs to be a shared time space or use of synchronism to keep the group together. Traditionally, musicians in an ensemble are constantly adjusting their individual performances in order to 'stay in time' and keep the piece musically cohesive (unless explicitly attempting to create a piece that subverts this); however, computers are not able to do this simply, and require some form of global synchronisation in order for ensemble timing to remain cohesive. Without this global synchronisation source, the different internal clocks of the various computers within an ensemble will eventually drift out of sync with each other, causing a perceptible shift in the timing of musical parts. While an ensemble of computer musicians could possibly perform in both the note-level context and sound processing context without needing synchronisation, it would be difficult to move successfully between all three contexts: for example, playing a synthesiser in the note-level interaction context, while having other material running from a section of the score.

Similar to how a shared time-space facilitates movement between interaction contexts, a pre-composed piece provides a musical guide for exploring these interaction contexts – that is, note-level control, score-level control and sound-processing control. For instance, it can be difficult to predict the compositional output of a procedural algorithm, and, as such, challenging to manage score-level control over the composition; however, a pre-composed piece affords the performer the means with which to accomplish score-level control (e.g. knowing the precise arrangement of a composition allows performers to anticipate large shifts in musical material, and coordinate executing score-level musical events with the other interaction contexts). Similarly, it would be difficult to improvise in the note-level or sound-processing contexts while letting the score-level context run in the background if the composition were not pre-composed. Without a pre-composed framework, note-level and sound-processing improvisation is reactionary, primarily responding to previous musical events. With a pre-composed framework, such improvisation can be anticipatory – the musician can perform using prior knowledge of the composition.

An ensemble exploring these interaction contexts would benefit from a pre-composed piece in the ways described above. Álvaro Barbosa describes such an ensemble as nonimprovisational, but refers strictly to the performer's inability to improvise the pre-composed piece. Barbosa acknowledges the performer's ability to improvise elsewhere, and states that, within the context of network systems for music and sonic arts, 'nonimprovisational performance means the process of playing a sequence of music events (a score) while providing some sort of synchronism with other music or visual events. In a performance, there is space left for individual expression and improvisation, but in musical terms the event should be driven to a certain extent by a predetermined and rehearsed composition' (Barbosa 2003: 57).

To functionally achieve these varied interaction contexts in performance, a significant portion of The Machine Orchestra process directly revolves around software development. The process of development has always been informed by specific musical ideas rather than requirements which try to meet all possible use cases. As development began on the first platform in 2008, the only functional requirement was a common note-level framework for sharing data between performers and among robots. In practice, this process has meant continually evaluating musical needs and incrementally adding new features to the software.

With each iteration of our software, we evaluate previous performances and identify areas that require new or innovative solutions, particularly pertaining

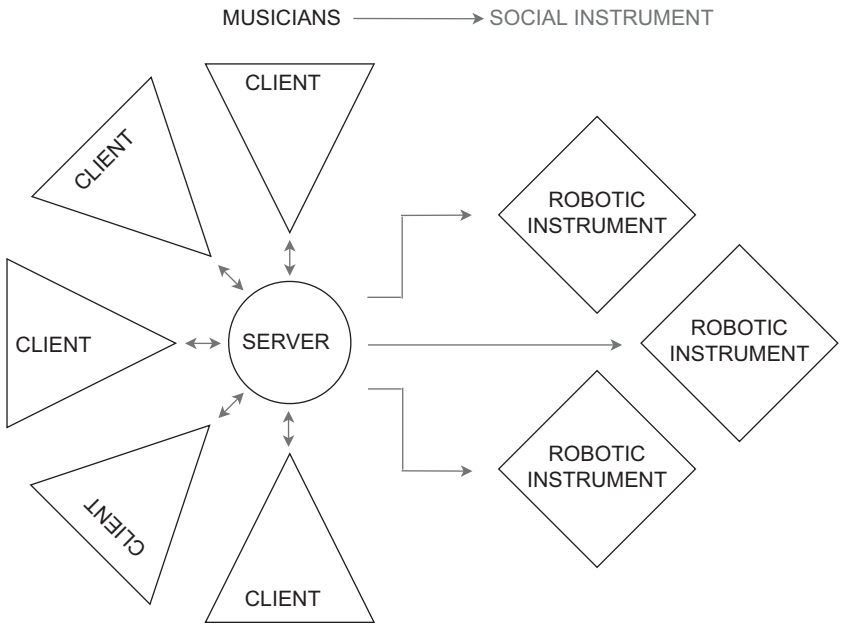

Figure 1. Network topology.

to robotic communication. In instances where information regarding network challenges exist (such as timing problems caused by network jitter), we typically defer to well-known solutions published within the academic community. The following section details the most recent ideas behind *Signal*, offering core implementation details essential to our network. While the needs of our platform are unique, we describe several key problems and solutions that we hope will be of further use to designers and musicians working with musical networks.

3. THE NETWORK FOUNDATION

Signal is the latest iteration of an ongoing client–server platform for TMO. Prior TMO research (Kapur et al. 2011) used the ChucK (Wang and Cook 2003) programming language, and addressed two primary needs of the ensemble: networked performer–performer and networked performer–robot communication. The codebase developed for this initial customised architecture drew inspiration from the framework written by PLork (Smallwood et al. 2008). *Signal* is a complete redesign of this codebase, now being written in C++ using the GNU-licensed Jules' Utility Class Extensions (JUCE) library.[1] Development of *Signal* is motivated by TMO's need for increased low-level control, routing, and monitoring of network data. Addressing these requirements (directly derived from compositional sessions) enabled us to pursue further research in human–robot interaction as a shared, social instrument, as well further research into ensemble performance within multiple interaction contexts. While the analysis of this research remains ongoing, in part because of a quick cycle between musical idea and implementation, the contributions described in this publication target three areas: network synchronisation, protocols, and overall system-usability. This section details the low-level implementation of the network and client–server architecture of *Signal*, further described as *Signal*|*Server* and *Signal*|*Client* respectively. It is critical to note that the problems and solutions presented below are often provided without piece-specific context; in reality, the features presented are the result of many incremental changes resultant of the cycle between musical idea and functional requirement. This decoupling should permit readers to envision similar contexts in their own musical networks that may be influenced by our process.

3.1. Topology and performance

The physical network topology on which *Signal* operates (see figure 1) is a departure from other contemporary laptop ensembles. The design of a low-latency, low-jitter network was necessitated by the inclusion of musical robotics and the need for tight synchronisation between performers. Electromechanical instruments possess an intrinsic layer of mechanical latency that cannot be compensated for in real-time performance. Based on tests done by Mark Cerqueira for PLork (Cerqueira 2010), an early decision was made to avoid wireless due to the added latency and jitter of consumer wireless equipment. In order to maintain temporal musicality across these physically dislocated

[1](2011) www.rawmaterialsoftware.com/juce.php.

instruments, and to minimise the potential need for jitter suppression algorithms, the ensemble is currently wired. At present, all networked data travels through an Extreme Networks 200E unit, an enterprise-grade ethernet switch. This topology compares favourably against work done by Chafe and Gurevich (2004), which maintains that networked ensembles often work best within the 10–20 millisecond (ms) range. The Machine Orchestra's latency metrics between the switch and individual performers indicate a round-trip overhead of no more than 1.20 ms under nominal network conditions (≤ 12 performers). All performers are wired using shielded CAT 5E cable in runs of 3, 7 and 15 metres.

3.2. Server

Signal/*Server* is the central exchange for all data across TMO's network. Similar to Weinberg's idea of 'flower topology' as part of what he calls a 'synchronous centralized network' (Weinberg 2005: 34), the server application does not typically act as the sole source of data, but rather an advanced routing and exchange platform between networked nodes. Since the ensemble has a primary focus on local-area-networked performance, the server application only handles symbolic music data, leaving high-bandwidth digital audio in the analogue realm if it needs to be passed among performers. This decision not to transport audio was done in an effort to reduce possible latency degradation of the network due to congestion. The server communicates to performers and robotic instruments via two standard protocols, OpenSoundControl (OSC) (Wright 2005) and USB-MIDI, respectively. This redesigned *Signal*/*Server* prototype was implemented in C++ for the following reasons: increased stability as it no longer runs in a virtual machine; improved application speeds as C++ is one of the fastest compiled languages; and for graphical user interface (GUI) support through JUCE, which includes advanced facilities for data collection, visualisation and parsing.

3.3. Client

The range of software used for audio synthesis and robotic control in the ensemble is varied: ChucK, Max/MSP, SuperCollider and Ableton Live are all commonly used in performance. However, not all applications can open network sockets and receive or transmit OSC data as required by the server. To mitigate this problem, the *Signal*/*Client* application was developed as a bidirectional MIDI/OSC translator. Early versions of the client were written in ChucK and provided methods to relay incoming sync data and map MIDI notes from host software to robotic actuators. In order for host software such as Live to communicate with this client, a virtual MIDI loopback port was used.[2] The rewritten C++ client includes these methods with additional facility to remove the necessity of a loopback port; in other words, the *Signal*/*Client* application will expose itself to the operating system and any host software as a virtual MIDI device itself. Future work will concentrate on a user interface (UI) tailored for graphically mapping MIDI notes to actuators with user-configurable velocity curves. In addition, the client will expose controls for starting/restarting/stopping sync, and graphical monitors for connection status and network statistics.

3.4. Synchronisation

The first of our three principal areas of investigation was designing a system tailored to our synchronisation requirements. The Machine Orchestra's music is highly dependent on a stable sync source for a number of performance contexts: tight rhythmic sequences between performers, tempo-synchronised audio effects, and note-level improvisatory playing. This aesthetic is derived both from the musical preferences of its members and also from the affordances of musical robotics. While TMO's wired network affords sufficiently low latency, the issue of jitter, however minor, is still problematic when distributing a clock source with dynamic tempo. In real-world use, this jitter translates into minor (1–4 beats per minute) clock drift above and below the desired tempo. Finding the source of jitter can be difficult (Cerqueira 2010), and even though musicians can adapt and perform with a fixed latency up to 40 ms (Chafe and Gurevich 2004; Brown 2010), tempo jitter as low as 1 ms is enough to significantly affect real-time performance (Schmeder and Freed 2008).

An earlier iteration of the ChucK prototype (see figure 2) distributed a clock signal through the server from a conductor client via MIDI at a standard 96 pulse-per-quarter (PPQ) resolution. The jitter from this conductor–server to server–clients relay was far from ideal, though we found it predictable enough to use in performance. Early efforts at combating jitter in the ChucK prototype were focused on implementing basic suppression algorithms such as low-pass and Kalman filters. By design, these filters are not particularly responsive to the rapid tempo changes which are instrumental to many TMO compositions. In an effort to further address the problem, we were motivated to seek alternate solutions not reliant on a relay-based architecture.

The cycle between the ChucK prototype and *Signal*/*Server* and *Signal*/*Client* allowed us to rethink the way we distributed sync messages (see figure 3). Instead of a

[2]In Apple's OSX this is known as an inter-application connection (IAC) bus.

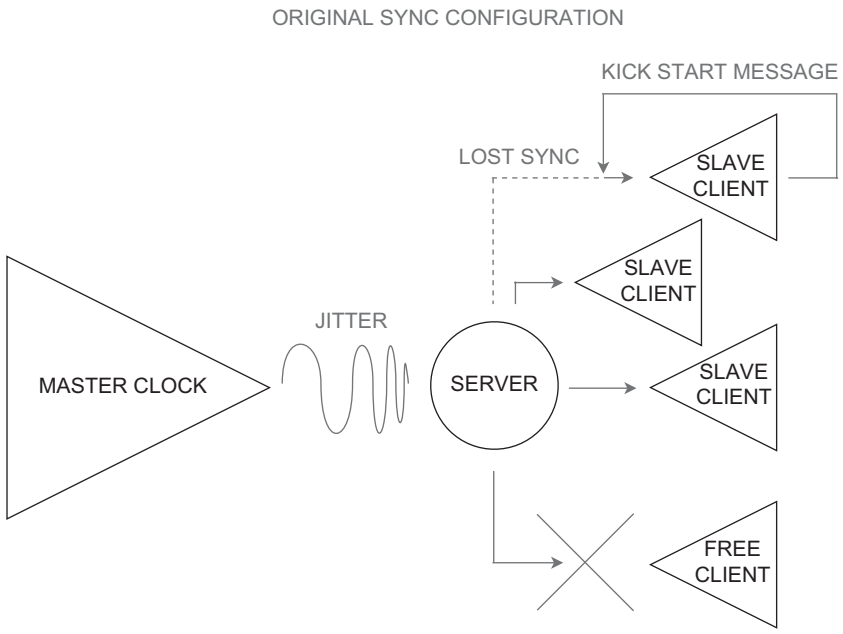

Figure 2. Original sync configuration.

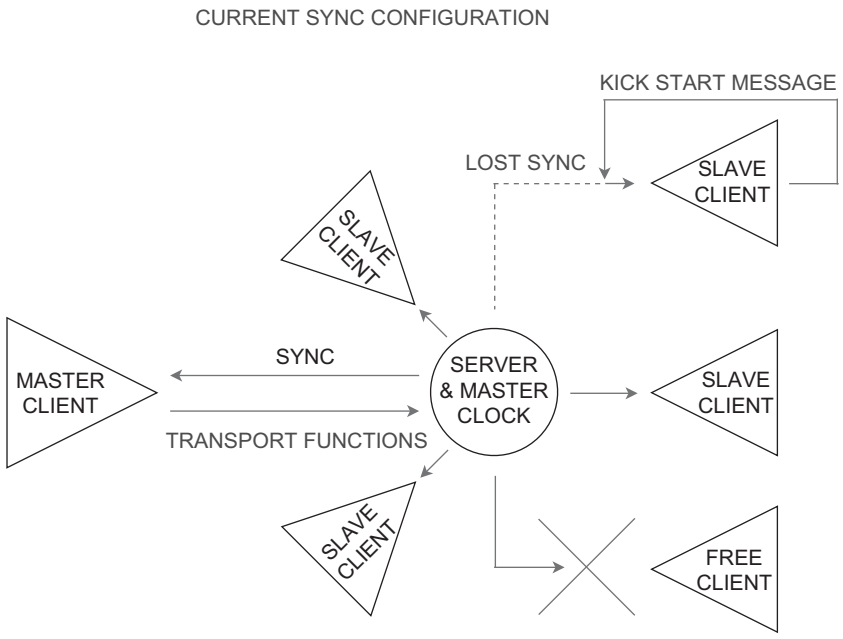

Figure 3. Current sync configuration.

conductor relaying messages through the server, a master *Signal*|*Client* has control over tempo but leaves the physical generation of the clock to a high-priority C++ thread of *Signal*|*Server*. This method minimises the amount of code necessary to distribute the messages and the number of times it passes through the switch. The move to compiled language with a full OSC library also led us to begin actively investigating the effectiveness of OSC's built in time-stamping functionality (Wright 2005; Schmeder and Freed 2008). The use of OSC time-stamps would permit clients to compensate for the jitter present in incoming sync messages, while simultaneously allowing a fast adaptation to global tempo changes from the server. Although using the time-stamps would require a buffer to allow for late timing messages, causing a fixed-latency delay, the delay should be well within human reaction times (Driessen et al. 2011). Current research is exploring this functionality and, while no current quantitative statistics exist to evaluate performance, our empirical observations suggest this approach sufficiently addresses this issue of clock drift.

3.5. Protocols

The second area of focus was standardising and optimising the necessary protocols for a functional musical network. Historically, the client and robots have communicated through a mixture of protocols including OSC, MIDI and serial. Several problems were apparent with the previous ChucK prototype. Firstly, synchronisation support on the client side mandates that we transport MIDI sync over OSC; secondly, the robotic instruments were controlled via serial, a protocol not supported by ChucK. The resultant system was a convoluted mix of protocols and conversions on both client and server, which negatively affected performance and 'debuggability'.

An early goal of the network was to standardise all network communication using OSC, considered by many to be the *de facto* protocol for musical networking. Since a number of musical applications only permit MIDI synchronisation, we began by designing a standardised messaging scheme to pass MIDI sync messages from the server to client over OSC. While this was a trivial feat, the performance penalty incurred by ChucK's virtual machine considerably increased jitter and would sometimes cause a client to fall out of the clock stream entirely. The MIDI specification requires that the master clock send stop/start messages for slave devices to reconnect properly. This would have disastrous results in live performance, so we introduced a kick-start feature into the client. This feature produced a start message, allowing any software applications to be immediately reinserted into the clock stream (see figure 2). The transition to *Signal* alleviated many of the jitter problems; however, we decided to keep the kick-start feature as a matter of convenience in the event of any unforeseen network hiccups.

The musical robotics built for TMO previously communicated via transistor–transistor logic (TTL) serial over USB to an Arduino microcontroller. In order for a performer to communicate with a robot, a note-on message would undergo a complex path to its destination. The server would receive a message as OSC, locally relay it over OSC to a Java application, and then finally re-package and send as serial. This convoluted man-in-the-middle software made pinpointing messaging errors and connection problems difficult. The design of *Signal* coincided with a related project allowing the robots to speak true MIDI over USB, negating the use of serial and this extra application (Diakopoulos and Kapur 2011). This project was instrumental to the architecture of *Signal*|*Server* and our goal of standardising on only two protocols, OSC and USB-MIDI.

3.6. System usability

Our final focus centred on analysing and improving the system in the usability domain. First attempts with the ChucK platform were not concentrated on developmental simplicity or designed with ease of use in mind; the software was more a test of the iterative approach of rapidly prototyping features which arise from current musical compositions. Furthermore, only a simplistic Java-based GUI was used for debugging and manually triggering the robotic instruments directly from the server. Prototyping *Signal* in JUCE allowed us to develop several XML-based APIs to ease general development as well as user-facing client interactions. Compared to previous iterations in ChucK, system transparency was greatly desired for performers to focus on musical tasks at hand rather than tracking down convoluted technical issues.

Two key XML-based APIs were developed for use with *Signal*: one for robotic instruments, and another for clients. These APIs will permit the dynamic handling of XML definition files for specific parameters. For robotic instruments, these definitions enable *Signal*|*Server* to understand basic properties such as name, number of actuators, velocity requirements, base note-to-actuator mappings, and permissions in the case of restricted or broken actuators. For clients, the XML definitions are generated programmatically through the API from special configuration messages sent by *Signal*|*Client*. These properties hold data for a unique identifier, IP address, permissions, and master–client status. Currently, manual definitions are generated in both cases. An expected feature of both client and server will be extended GUI options for graphically editing the data. The server's knowledge of these dynamic configurations enables a number of convenient features such as a GUI for robotic actuation tests and an overview of the network's state.

While still under heavy development, *Signal*|*Client* exposes a GUI for monitoring connection status/performance, configuring certain options related to synchronisation/musical interfaces, and selecting actuator mappings. In the case of synchronisation, the clients can enable or disable the MIDI sync messages or reinsert themselves into a clock stream. Clients will also have the ability to capture data from their musical interfaces and relay to the server for other performers to use. One of the most useful planned features will be the ability for performers to edit arbitrary mappings of MIDI notes to robotic actuators; this is a special use-case for performers who use generative software that may produce a wide range of MIDI notes and want control over a set of actuators.

4. NETWORKED MUSIC

Since 2009, TMO has been exploring a wide variety of networked performance scenarios made possible

by *Signal*. In this section, we focus on three pieces that display the ability for TMO performers to:

1. co-exist in multiple performance contexts;
2. embody and communicate complex synthetic musical interactions to the audience via the use of custom musical controllers; and
3. compose and perform for the shared, social instrument – that is, the musical robotics.

As described earlier, these pieces were created though an iterative process that continually evaluated musical ideas and their functional implementation. While the primary goal of *Signal* was to provide a platform that did not hinder the creative process, and catered to the unique needs of musical robotics, the pieces here illustrate that the resulting system deeply and positively impacted our developing aesthetics of embodied performance and audience understanding. The pieces here also outline the prevailing sonic aesthetics of TMO, which draw ideas from world music, minimalist, and beat-oriented genres; however, the ultimate goal is to emphasise how our iterative process was instrumental in providing a compelling concert.

4.1. Sitka Chant, Moksha and Twilight

In January 2009, TMO held a concert at the Roy and Edna Sisney/CalArts Theater (REDCAT) that premiered the pieces *Sitka Chant*, *Moksha* and *Twilight*. These pre-composed pieces included nine performers and three musical robots, and required the ensemble to practise the arrangement beforehand, learning to follow the direction of a single lead performer who doubled as a conductor. Although all performers were locked to a common tempo, they had the ability to move freely through the arrangement by launching different groupings of loops. This is similar to Terry Riley's *In C* (1964), in that the loops can be thought of as cells that represent different sections of the song. In a similar manner to Barbosa's description of nonimprovisational music, the pre-composed pieces dictated what grouping of loops all performers should be playing at a given point in the piece, but there remained the flexibility to vary or manipulate the arrangement by moving within the small group of currently available loops. Two members of the ensemble were responsible for all the drum and percussion parts within the pieces, and provided these parts while performing in several different contexts. These contexts consisted of a combination of audio loops sent to the speakers, MIDI loops sent to the robots, manipulation of the audio material through custom Reaktor software,[3] and note-for-note performance of samples and/or the robotic instruments. Much of the percussion material composed for the piece was polyrhythmic, with the perceived groove of these parts being highly dependent on the synchronous timing between two of the performers. The remaining musicians in the ensemble were split into two distinct groups: group A performed harmonic and melodic material, while group B added more timbre and gesture-based sound material to the pieces.

[3]Modular Software Synth program made by Native Instruments (2011). www.native-instruments.com.

4.2. Mechanique

Premiering at the January 2009 REDCAT concert, *Mechanique* explored the idea of the shared social instrument by creating a many-to-many relationship between the performers and the robotic instruments. Musicians were not assigned to a single fixed robotic instrument, rather each musician was assigned several actuators across multiple robots; as a result, the network afforded the musicians the ability to perform simultaneously on all of the physically separated robotic instruments. This is a unique feature of working with the shared social instrument and would otherwise be impossible for traditional acoustic ensembles. The ability to divide a single musical instrument/system across multiple performers (e.g. in *Mechanique*, each robot has its actuators controlled by multiple musicians), also enables complex musical interactions that are impossible or difficult to achieve for a single performer. This idea is similar to ideas from Indonesian music, where instruments in the gamelan such as the *reyong* are situated amongst 2–4 players, or other instruments that are gendered into two parts, to achieve extremely fast polyrhythms. Although *Mechanique* was an improvisatory piece that encouraged performer spontaneity (Kapur et al. 2011), there was a higher-level structure dictating when musicians (approximately) entered the piece, as well as the dynamic contour (see figure 4). *Mechanique* began by sparsely introducing the various robotic instruments. As the piece progressed, it grew denser as the ensemble began to play more of the actuators, until gradually the piece crescendoed to an abrupt ending. Additionally, the physically separated, percussive robotic instruments were given a sense of spatial coherency using instrumental drones and synthesised textures provided by several of the performers. These ideas were central in providing the piece with a simple core from which the complex robotic improvisations could emerge and contrast.

As the second piece played in the concert, a key goal of *Mechanique* was to bring in the robotics slowly, thereby introducing the audience to the robotic instruments and the performer's various custom interfaces and controllers. This process allowed the audience to identify individual performers' actions with the physical sound producing actuators of the robots, and, in doing so, reinforced the potential for embodiment represented

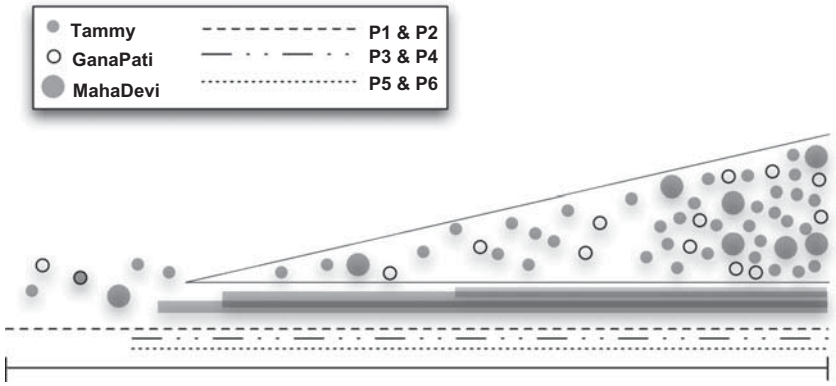

Figure 4. *Mechanique*'s graphical score.

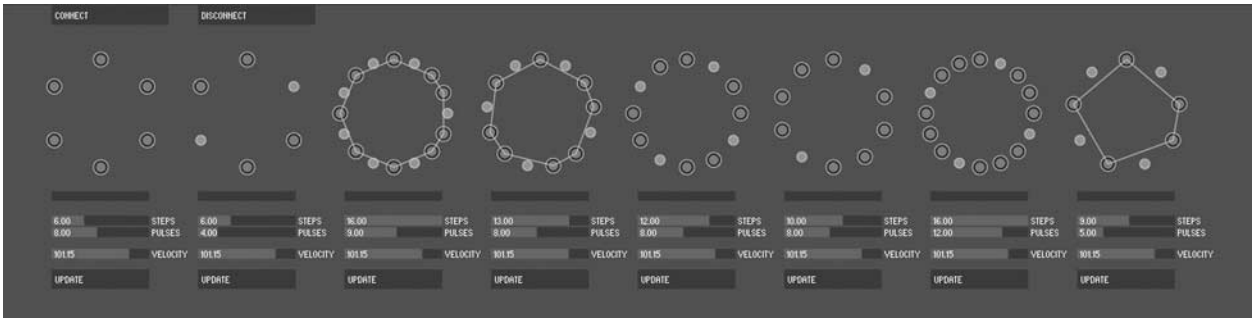

Figure 5. Euclidean sequencer clients.

by the robotic instruments. Additionally, the piece was highly rhythmic and improvisatory in nature, requiring accurate timing and sync between the musicians and robotic instruments. The ability for the *Signal* framework to provide this sync allowed the performers to trigger both individual hits and small phrase-based loops across the multiple robots. This both established rhythms and provided the opportunity for highly spontaneous musical explorations.

4.3. Sequence

Sequence is a structured improvisation involving human players and electromechanical instruments with no synthesised audio. Premiering at the 2011 TMO show, the piece was based on layering polyrhythmic Euclidean rhythms that are further described by Godfried Toussaint in his paper 'The Euclidean Algorithm Generates Traditional Music Rhythms' (Toussaint 2005). Every performer was assigned eight actuators and given a Java-based client that visualised and controlled the circular patterns (see figure 5). Unique to this piece and separate from the sync architecture built for the rest of the ensemble, the performers were only given the illusion of direct control on their client. In reality, the precise timing among the patterns played by each of the 48 actuators was provided sample-synchronously by *Signal*|*Server* (see figure 6). Each client was sent a basic heartbeat message every two seconds, allowing the client GUI to stay approximately in sync with the performer's sounding patterns. In removing distributed synchronisation, *Sequence* examines the notion of a central clock source from a different perspective. While this paradigm may not be appropriate for all pieces, it illustrates the flexibility of *Signal*|*Server* in affording alternate performance scenarios outside of the typical distributed sync system of TMO. Visually, each pattern is projected on a screen visible behind the performers, re-enforcing the audience's ability to connect each rhythm with a specific performer. Sonically, the piece was structured to begin minimally, with few sounding rhythms. As the piece grows, performers are instructed to begin introducing increasingly complex patterns in differing metres, resulting in a dense polyrhythmic texture. Several performers are tasked with keeping several of their actuators playing a simple pattern in 4/4, helping the piece to maintain a cohesive beat in the increasingly maximal polyrhythmic soundspace. In many ways, *Sequence* is a spiritual successor to *Mechanique*, serving many of the same purposes in familiarising an audience with TMO early in a concert.

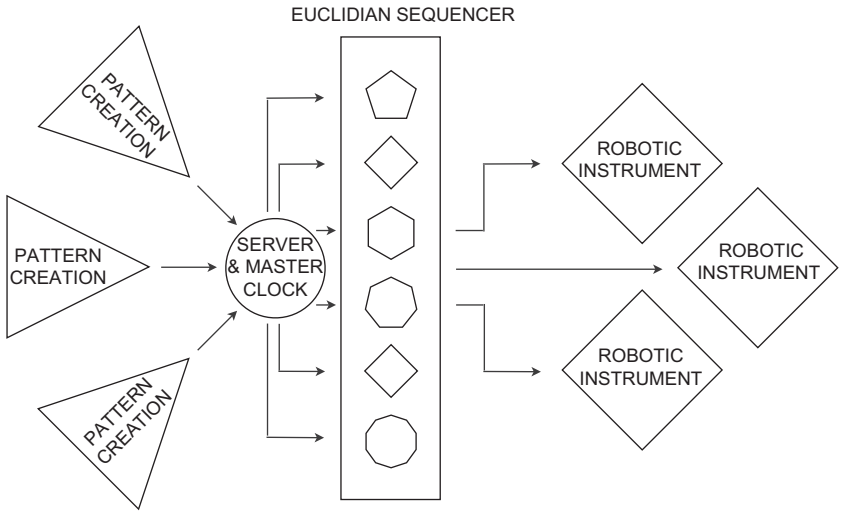

Figure 6. Euclidean sequencer topology.

5. CONCLUSION

Our discussion of interaction contexts represents the unique ability of live computer musicians to extend the performance capability of a single individual through the help of networked software. We have demonstrated how *Signal*, the result of iteratively constructed software designed to accommodate our own musical needs, addresses interconnectivity and musical embodiment without being explicitly designed to do so. In concentrating on the musical demands of the compositions and their relationships to the performers and robots, we have actualised a platform that mediates networked performance without being tied to any one musical idea. One of the main themes in this work is that the success of *Signal* is dependent on the effectiveness of tightly integrated relationships with specific compositions, its own implementation, musical practice, and resultant performance. Through our process of incremental design, dictated entirely by compositional requirements, performers now have access to a musically expressive platform from which they can move through different interaction contexts seamlessly and transparently.

Each time a new TMO piece is performed, new issues arise, while solutions to existing challenges are tested and evaluated. Through the pieces we have presented, we have demonstrated an iterative process that has deeply influenced not only the direction of our software but also our aesthetics. By carefully examining our needs as they arise from our compositions, we hope to offer many technical observations that potentially have great influence on future musical networks: programming languages matter – the lower level, the better; timing is critical: balance the affordances of high-level clocks with their machine-level equivalents; protocols are important – OSC is the *lingua franca* of musical networking; parsing and monitoring network activity is important to quickly and painlessly debug connectivity issues; networking frameworks should be flexible and able to accommodate the development of new interaction contexts; and, lastly, the music should dictate the technology, and not the other way around.

Moving into the future, we hope the ideas presented by our integration of musical robotics into a networked ensemble may influence the direction of new and existing networked ensembles. The lessons derived from the conception and realisation of *Signal* should should be relevant and applicable to a wide variety of interaction contexts outside TMO's own needs. While the novelty of a cycle between musical idea and programmatic implementation may be limited, following such a pattern in a loose but structured way has a great potential to illuminate and reveal many potential avenues for musical exploration that may not have been apparent had the system been fully designed and implemented from the start. We stress that iterative development is key for networked ensembles and hope other ensembles may be able to adopt similar paradigms in the realisation of their own musical ideas.

REFERENCES

Barbosa, A. 2003. Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation. *Leonardo Music Journal* **13**: 53–9.

Barbosa, A., Cardoso, J. and Geiger, G. 2005. Network Latency Adaptive Tempo in the Public Sound Objects System. *Proceedings of the 2005 Conference on New Interfaces for Musical Expression (NIME)*. Vancouver, Canada.

Bischoff, J., Gold, R. and Horton, J. 1978. Music for an Interactive Network of Microcomputers. *Computer Music Journal* **2**(3): 24–9.

Brown, A.R. 2010. Network Jamming: Distributed Performance using Generative Music. *Proceedings of the 2010 New Interfaces for Musical Expression (NIME)*. Sydney.

Cáceres, J.-P. and Chafe, C. 2010. JackTrip: Under the Hood of an Engine for Network Audio. *Journal of New Music Research* **39**(3): 183–7.

Cáceres, J.-P., Hamilton, R., Iyer, D., Chafe, C. and Wang, G. 2008. To the Edge with China: Explorations in Network Performance. *Proceedings of the 2008 4th International Conference on Digital Arts*. Portuguese Catholic University, Porto.

Cerqueira, M. 2010. *Synchronization over Networks for Live Laptop Music Performance*. Thesis, Princeton University.

Chafe, C. and Gurevich, M. 2004. Network Time Delay and Ensemble Accuracy: Effects of Latency, Asymmetry. *Proceedings of the 2004 117th AES Convention*. San Francisco, CA.

Diakopoulos, D. and Kapur, A. 2011. HIDUINO: A Firmware for Building Driverless USB-MIDI Devices using the Arduino Microcontroller. *Proceedings of the 2011 New Interfaces for Musical Expression (NIME)*. Oslo.

Driessen, P.F., Darcie, T.E. and Pillay, B. 2011. The Effects of Network Delay on Tempo in Musical Performance. *Computer Music Journal* **35**(1): 76–89.

Gresham-Lancaster, S. 1998. The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Computer Music. *Leonardo Music Journal* **8**: 39–44.

Kane, B. 2007. Aesthetic Problems of Net Music. *Proceedings of the 2007 Spark Festival*, University of Minnesota.

Kapur, A., Darling, M., Diakopoulos, D., Murphy, J.W., Hochenbaum, J., Vallis, O. and Bahn, C. 2011. The Machine Orchestra: An Ensemble of Human Laptop Performers and Robotic Musical Instruments. *Computer Music Journal* **35**(4): 49–63.

Lazzaro, J. and Wawrzynek, J. 2001. A Case for Network Musical Performance. *Proceedings of the 2001 International Workshop on Network and Operating Systems Support for Digital Audio and Video*. Port Jefferson, NY: ACM.

Schmeder, A. and Freed, A.A. 2008. Implementation and Applications of Open Sound Control Timestamps. *Proceedings of the 2008 International Computer Music Conference (ICMC)*. Belfast: ICMA.

Smallwood, S., Trueman, D., Cook, P.R. and Wang, G. 2008. Composing for Laptop Orchestra. *Computer Music Journal* **32**(1): 9–25.

Toussaint, G. 2005. The Euclidean Algorithm Generates Traditional Musical Rhythms. *Proceedings of the 2005 Bridges Mathematical Connections in Art, Music, and Science*, Alberta, Canada.

Wanderley, M.M. and Orio, N. 2002. Evaluation of Input Devices for Musical Expression: Borrowing Tools from HCI. *Computer Music Journal* **26**(3): 62–76.

Wang, G. and Cook, P.R. 2003. ChucK: A Concurrent, On-the-fly Audio Programming Language. *Proceedings of the 2003 International Computer Music Conference (ICMC)*, Singapore.

Weinberg, G. 2002. The Aesthetics, History, and Future Challenges of Interconnected Music Networks. *Proceedings of the 2002 International Computer Music Conference (ICMC)*. San Francisco, CA.

Weinberg, G. 2005. Interconnected Musical Networks: Toward a Theoretical Framework. *Computer Music Journal* **29**(2): 23–39.

Whalley, I. 2010. Networked Electronics Music, Machines & Internet2: A Conceptual Framework and Some Artistic Issues/Responses. *New Zealand Electroacoustic Music Symposium 2010*, Auckland, New Zealand.

Wright, M. 2005. Open Sound Control: An Enabling Technology for Musical Networking. *Organised Sound* **10**(3): 193–200.