Tangle: a Flexible Framework for Performance With Advanced Robotic Musical Instruments

Paul Mathews
Victoria University of Wellington
School of ECS
Wellington, New Zealand
pfcmathews@gmail.com

Ness Morris
California Institute of the Arts
24700 McBean Parkway
Valencia, California
karplusstrong@gmail.com

Jim Murphy
Victoria University of Wellington
New Zealand School of Music
Wellington, New Zealand
jim.w.murphy@gmail.com

Ajay Kapur
California Institute of the Arts
24700 McBean Parkway
Valencia, California
ajay@karmetik.com

Dale A. Carnegie
Victoria University of Wellington
School of ECS
Wellington, New Zealand
dale.carnegie@ecs.vuw.ac.nz

ABSTRACT

Networked musical performance – using networks of computers for live performance of electronic music – has evolved over a number of decades but has tended to rely upon customized and highly specialized software designed specifically for particular artistic goals. This paper presents *Tangle*, a flexible software framework designed to provide a basis for performance on any number of distinct instruments. The network includes features to simplify the control of robotic instruments, such as automated latency compensation and self-testing, while being simple to extend in order to implement device-specific logic and failsafes. Tangle has been tested on two diverse systems incorporating a number of unique and complex mechatronic instruments.

Keywords

network, performance, robotic instruments, shared instruments

1. INTRODUCTION

Tangle builds on the foundation provided by The Machine Orchestra (TMO) [8] and software developed for it [12], attempting to address issues of portability and flexibility in these systems. It is currently being successfully used to control two very different groups of instruments at separate institutions – Victoria University of Wellington, New Zealand and the California Institute of the Arts, Los Angeles.

Tangle differs from the now-classic laptop orchestra configuration [11] in that it focuses on the control of remote instruments and arbitrary performer-instrument links as opposed to the direct 1:1 performer-instrument ratio found in most laptop orchestras. A key focus is enabling the control of arbitrary robotic instruments, for which we provide a latency calibration mechanism and the ability to easily implement device-specific logic and failsafes where necessary.

The primary goal of the network is to provide a robust back-end upon which any kind of interactive system can be built.[1] Its function is to simplify and enable working with groups of musical robots and instruments rather than having an intrinsic artistic purpose *per se*, and it is as valuable to a solitary performer as to a group of many.

Before presenting our network and the details of the various mechanisms it provides to enhance usefulness and adaptability, we will first briefly examine the history of musical networks with a focus on those created to share control of instruments. We will look at special adaptations necessary for the control of sophisticated robotic instruments and then proceed to examine the way in which Tangle provides greatly enhanced facilities for unified control of a range of heterogeneous devices.

2. BACKGROUND

Networked musical performance has evolved considerably since its early appearances in the mid 20th century to suit a range of specific needs. Some of the first examples are performances given by John Bischoff, Rich Gold and Jim Horton in 1978 on three separate "microcomputers" in which each performer controlled part of another's synthesis algorithm [3].

In the 1980s and 1990s, the Hub built on this idea [6] by undertaking a number of innovative performances which included the use of MIDI to send peer-to-peer control data, sonifying poems posted to the internet and some exploratory attempts at geographically separated performance using the internet. Many works for the Hub focused on indeterminacy and emergent behaviors very much in the vein of electronic performances and works by "John Cage, David Tudor, Gordon Mumma, Pauline Oliveros and many others" [6].

The main focus these works has been to use network technology to enable new kinds of performer-performer interaction. Weinberg notes that this is likely an evolution of traditional instrumental performance practice in which musicians are dependent on one another continuously for feedback [15].

Weinberg also considers John Cage to have been among the first to attempt the use of electronics to enable novel interactions in this sphere with pieces such as *Imaginary Landscape No. 4* (1951) and *Cartridge Music* (1960). While the technology has developed significantly, the approach

[1] A comprehensive taxonomy of social interactions within musical networks can be found in [15].

commonly taken by networked music is still to focus on structuring the interaction between performers.

3. OBJECTIVES

Tangle differs from related technologies in that it attempts only to enable, not to limit composers and is agnostic to inter-user interaction. Our focus is instead on providing a building block which can be used for any artistic pursuit, as opposed to previous efforts where the creation of the network was a large part of the endeavour and the network itself provided important constraints guiding the expression.

The goal of Tangle is to enable consistent and predictable control of diverse groups of instruments. As the focus is on enabling any number of artistic pursuits it follows that the system must be capable of controlling arbitrarily complex sets of instruments. Inevitably, complex and expressive instruments require complex control to maximize their capabilities. This is especially noticeable when considering systems such as *MechBass* [9] and *Swivel 2.0* [10] which eschew the relatively simple solenoid actuation of many of their predecessors in favor of the greater degrees of control easily afforded by stepper motor or servo control.

These systems enable fine grained control over the produced sound in exchange for complexity of the mechanism. For example, for Swivel to play a note, first a string must be chosen, then the appropriate position for the pivot servo must be calculated to achieve the correct pitch, the desired pressure to clamp to the string must be chosen, and then the force with which to pick the string needs to be decided upon as well as the time until the string is damped and with what pressure.

The number of decisions required to cause Swivel to play a single note illustrate the complexity that comes with enhancing the expressive power of an instrument. Further, this shows the degree of heterogeneity to which the network must be able to adapt. Hence a primary design goal: maintain extensibility and adaptability while presenting a consistent and predictable interface to users.

4. IMPLEMENTATION

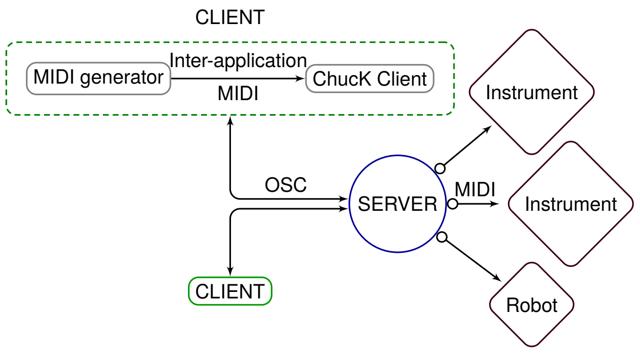

Figure 1: Overview of the network

The framework is implemented in ChucK [14] which provides easy to use implementations of common musical control protocols in an approachable syntax for both experienced programmers and novices. Network communication is performed through Open Sound Control (OSC) messages sent via User Datagram Protocol (UDP).

The system uses a server-client topology similar to The Machine Orchestra [8, 12] (see figure 1). The network addresses just one of Wanderley and Orio's seven interaction contexts: "*note-level control* or *musical instrument manipulation*," [13] as this is all that is necessary to control instruments. Where the note-level control comes from and what is done with the result is left entirely up to the user.

This topology connects an arbitrary number of musicians to an arbitrary number of instruments. It does not limit connectivity; at any time any user may communicate with any instrument. Indeed through the use of OSC's pattern matching in conjunction with the unified interface presented, one user can simultaneously control multiple instruments. The effect of this topology is to turn the set of instruments into one shared meta-instrument and thereby enable synchronous multi-user interaction. This is important in enabling a TMO style of instrument sharing in which all members of the ensemble have equal access [12] but it is amenable to an overlay of stricter structures.

To date the system has only been used to set up local networks, but the benefit of OSC over UDP is that it enables effortless scaling to wide area networks and so provides attendant remote performance opportunities. Using Barbosa's classifications [1] it fits either as a "Local Inter-Connected Network" or a "Remote Music Performance System" depending on precisely how it is used.

4.1 Client

The client software acts as a translator, turning MIDI into OSC and sending the OSC across to the server. For convenience it consists of a single file of ChucK code, configured through arguments using an accessible key=value syntax.

On starting up, the Client notifies the server of its existence and receives, by way of confirmation, a list of the instruments connected to the server, the messages they have available and any information about the instrument. This handshaking ensures that the client software always has an up to date list of instruments connected to the server which enhances the network's adaptability by allowing seamless addition of instruments.

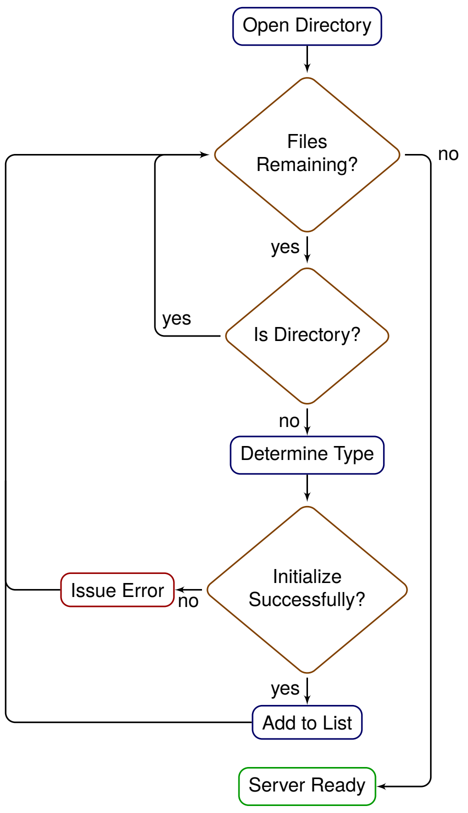

Figure 2: Server process for instrument discovery.

4.2 Server

The server is designed in such a way that it is easy to configure for new instruments and simple to organize for arbitrary groups of existing instruments. Primarily this is done through polymorphism and inheritance; the server keeps a list of top-level Instrument objects which handle the OSC setup and the sending of information about each instrument to the client (name, messages it is listening for and notes about what they do). The output to the physical instrument is handled by specific subclasses. Currently this includes a generic MIDI instrument which reads in information from a configuration file and several specific subclasses of this which contain additional logic. These are described further in section 4.3.2.

On starting up, the server attempts to determine what instruments are connected by a process that combines reading files in a directory and attempting to connect to instruments to determine their availability. This process is illustrated in figure 2.

What this entails is that the server scans through all the files in a specific directory (ignoring subdirectories) and reads the first line. If the line is equivalent to `type=something` it will use the key `something` to look up which instrument to instantiate. Once it has an instance it calls the instrument's initialization method, which operates on the remainder of the file. This enables instruments to be configured in a simple text file. If the instrument successfully parses the configurations and connects to its output, the initialization method returns successfully and the instrument is ready to use. If it encounters a failure, such as if the specified device can not be found, a warning will be emitted but the server will continue searching for instruments and those that do initialize successfully will still be useable.

4.3 Additional Features

The system includes a series of features designed to assist in control of heterogeneous groups of instruments, especially those that may be physically actuated. These include a latency compensation mechanism and the ability to extend the network to introduce arbitrary per-instrument logic. We discuss these further below and provide examples of when this has been indispensable.

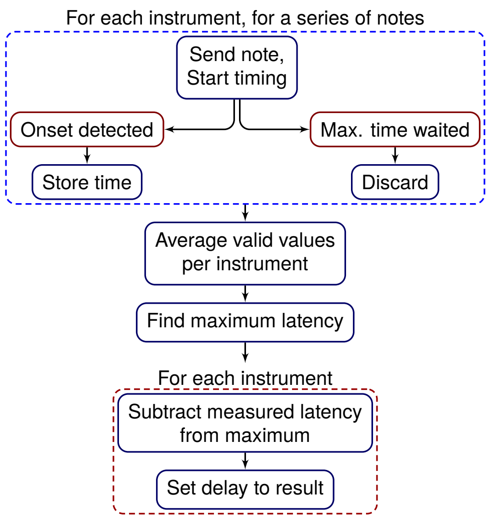

Figure 3: Latency calibration process

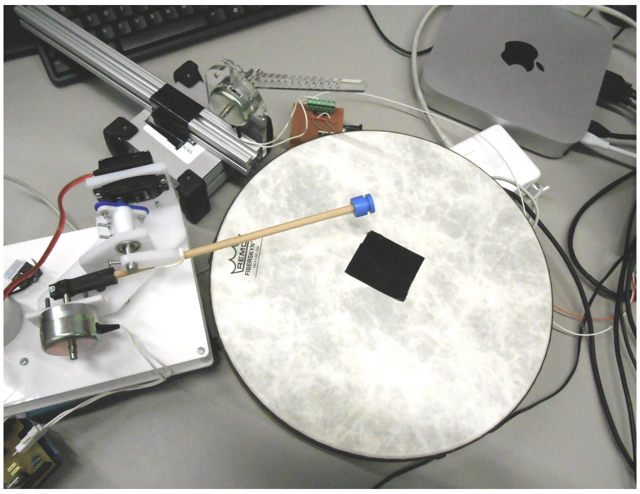

Figure 4: Test setup for latency compensation.

4.3.1 Latency Compensation

At any time, a client can request that the server run in latency-compensated mode. The OSC message to trigger this requires, as an argument, a list of names of instruments. Upon receiving this request the server iterates through the list and attempts to measure the delay between when it sends a note message to the instrument and when it detects an onset in the system default audio stream. When testing multiple instruments, it is currently simplest to mix the various microphone or direct signals to similar levels externally and pass a mix to the server as it processing instruments sequentially.

The server determines the maximum delay time across all measured instruments and inserts the appropriate delay between receiving a message and passing it to the instrument so they will sound at the same time. This is calculated as

$$d(i) = t_{max} - t_i$$

where $d(i)$ is the appropriate delay for a given instrument i and t_{max} and t_i are the maximum and per instrument measurements respectively. If the instrument was not measured, t_i will simply be 0.

The onset detection method uses both the spectral flux and the root mean square (RMS) level of the magnitude spectrum. Spectral flux (or spectral difference) is for our purposes the Euclidean distance between successive magnitude spectra. The combination of these two methods ensures robustness at determining both pitched and non-pitched onsets [2]. The peak picking was performed with thresholds for each value, which are currently set manually. In our testing, using two rotary solenoid drum beater (figure 4), the system overcame 50ms of delay inserted into one beater to cause the two to strike with no perceptible time difference.[2]

While a great number of onset detection algorithms have been published [2, 4, 5] and implementation of a more complex algorithm does have the potential to improve results, our approach is simple and effective. Time domain resolution is limited to the period of the Discrete Fourier Transform used, although precise frequency data is not strictly necessary so a small transform period is acceptable, increasing time-domain resolution and decreasing processing time.

The drawback of adding delay to increase inter-instrument synchronization is that it also delays the sounding result of all user input to any instrument. In a performance context

[2]a video demonstration can be found at: `https://vimeo.com/85770432`

this could be undesirable if the intent is to directly control single instruments. For systems with a less direct mapping between performer and instrument or where synchronicity is paramount, this automated calibration presents a useful tool.

4.3.2 Per-Instrument Extensibility

The class hierarchy within the server is structured such that, if desired, instruments can be added to the system by creating a subclass which overrides the server-instrument communication. In this way it is simple to add specific logic and failsafes or to adapt the system to a whole new communication protocol.

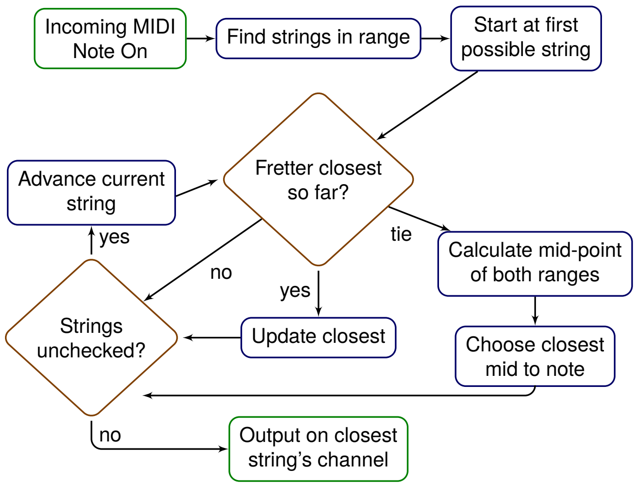

Figure 5: String choosing process for *MechBass*.

An example of when this ability proved useful was demonstrated with *MechBass*, a mechatronic bass guitar [9] which uses a separate MIDI channel for each string and can take a long time to move to a note position. This instrument is challenging to compose for, as it requires the composer to be aware of the range and current position of each string and to sequence each as separate tracks. In order to enable easier use two new classes were added to the hierarchy: an abstract class descending from the basic MIDI Instrument which implements several string choice algorithms and a descendant of this class which specializes its parent for MechBass. The algorithm uses the current state of the four strings and chooses a string for output based on whichever is nearer to the desired note.

Kritaanjli [7] also required additional logic. This robotic harmonium uses a DC motor to pump the bellows to provide airflow and make sound. This can only be done safely when at least one key is depressed or the instrument risks damage. This is an awkward condition to deal with, but was straightforward to solve with a specific subclass on the server side which keeps track of the number of notes depressed, turning off the bellows motor if no keys are pressed.

5. CONCLUSIONS

Tangle provides an adaptable and extensible framework for networked control of diverse and arbitrarily complex instruments and robots. It incorporates useful tools for composition and performance without limiting users to a given paradigm. Adding and removing instruments is simple and easy, as is extending the system to safely and effectively control sophisticated and expressive instruments. Future goals include expanding the output capabilities to encompass non-MIDI devices as well as exploring grid configurations where the instruments are distributed amongst peers.

Tangle is an essential step towards moving networked music forward from the device-specific, project-specific networks that have previously been the norm towards a flexible and general tool that can be used in any number of situations and to enable any number of novel artistic practices.

References

[1] Á. Barbosa. “Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation”. In: *Leonardo Music Journal* 13 (2003), pp. 53–59.

[2] J. P. Bello et al. “A Tutorial on Onset Detection in Music Signals”. In: *Speech and Audio Processing, IEEE transactions on* 13.5 (2005), pp. 1035–1047.

[3] J. Bischoff, R. Gold, and J. Horton. “Music for an Interactive Network of Microcomputers”. In: *Computer Music Journal* 2.3 (Dec. 1, 1978), pp. 24–29. ISSN: 0148-9267. DOI: 10.2307/3679453.

[4] N. Collins. “A Comparison of Sound Onset Detection Algorithms with Emphasis on Psychoacoustically Motivated Detection Functions”. In: Audio Engineering Society Convention 118. 2005.

[5] S. Dixon. “Onset detection revisited”. In: *Proc. of the Int. Conf. on Digital Audio Effects (DAFx-06)*. 2006, pp. 133–137.

[6] S. Gresham-Lancaster. “The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Computer Music”. In: *Leonardo Music Journal* 8 (1998), pp. 39–44. ISSN: 0961-1215. DOI: 10.2307/1513398.

[7] A. Kapur, J. Murphy, and D. Carnegie. “Kritaanjli: A Robotic Harmonium for Performance, Pedogogy and Research”. In: *Proceedings of the International Conference on New interfaces for Musical Expression*. Michigan, May 21, 2012.

[8] A. Kapur et al. “The Machine Orchestra: An Ensemble of Human Laptop Performers and Robotic Musical Instruments”. In: *Computer Music Journal* 35.4 (2011), pp. 49–63. ISSN: 1531-5169.

[9] J. McVay et al. “MechBass: A Systems Overview of a New Four-Stringed Robotic Bass Guitar”. In: *Proceedings of the 2012 Electronics New Zealand Conference*. Dunedin, New Zealand, 2012.

[10] J. Murphy et al. “Designing and Building Expressive Robotic Guitars”. In: *Proceedings of the 2013 Conference on New Interfaces for Musical Expression (NIME), Daejeon, Korea*. 2013.

[11] D. Trueman et al. “PLOrk: the Princeton laptop orchestra, year 1”. In: *Proceedings of the international computer music conference*. 2006, pp. 443–450.

[12] O. Vallis et al. “Building on the Foundations of Network Music: Exploring interaction contexts and shared robotic instruments”. In: *Organised Sound* 17.1 (2012), pp. 62–72. DOI: 10.1017/S1355771811000525.

[13] M. M. Wanderley and N. Orio. “Evaluation of input devices for musical expression: Borrowing tools from hci”. In: *Computer Music Journal* 26.3 (2002), pp. 62–76.

[14] G. Wang. “The Chuck Audio Programming Language. ”a Strongly-timed and On-the-fly Environ/Mentality””. Princeton, NJ, USA: Princeton University, 2008.

[15] G. Weinberg. “Interconnected Musical Networks: Toward a Theoretical Framework”. In: *Computer Music Journal* 29.2 (July 1, 2005), pp. 23–39. ISSN: 0148-9267.