ZSCORE: A DISTRIBUTED SYSTEM FOR INTEGRATED MIXED MUSIC COMPOSITION AND PERFORMANCE

Slavko Zagorac
Goldsmiths, University of London
slavko@zagorac.com

Patricia Alessandrini
Goldsmiths, University of London
p.alessandrini@gold.ac.uk

ABSTRACT

This paper proposes a distributed system design for mixed ensemble music composition and performance of stave-based dynamic scores. ZScore is a collection of third-party and newly-developed components which aims to implement described networked notation solutions. The solution scope includes complex notation authoring, reliable score data distribution over a network to heterogeneous clients, precise performance scheduling and dynamic rendering of interactive scores. Taking the specification of optimal system features as a starting point, this paper looks at suitable solutions from other industries where high-throughput, low-latency systems have been successfully implemented. It presents the case for SVG-based notation representation, its distribution over a reliable message-oriented middleware and the innovative alternating pane layout design for dynamic notation rendering. Finally, the paper describes the current state of ZScore development and outcomes from initial user trials. It concludes with future perspectives towards realizing the underlying ambition behind this project: to blur and thereby call into question the traditional boundaries between the roles of a composer, performer, conductor and audience through the effective utilization of cutting-edge technology.

1. INTRODUCTION

During the last two decades, a number of software solutions able to dynamically render music scores distributed over a local or wide area network have been developed. Amongst the available solutions are: InScore [1], Quintet.net [2] and NetCanvas [3] which utilize MaxScore [4] for notation, Bach Composer [5], Decibel ScorePlayer [6] and dfscore [7]. Antescofo [8] offers integrated mixed ensemble composition and performance notation while recent Odot developments also allow for OpenMusic [9] notation integration over a network. Some composers and laptop orchestras in particular develop proprietary composition and performance software in programming environments such as MaxMSP, Processing, SuperColider and ChucK programming language. While all notation applications share common high level functional objectives, their internal data models, score rendering versatility, system dependencies, time synchronization strategy and modes of communication can vary greatly.

Open Sound Control (OSC) messaging protocol has emerged as the leading choice for data and control message encoding and distribution between music notation applications over a computer network. The majority of OSC implementations rely on User Datagram Protocol (UDP) connectionless communication. InScore supports OSC natively, while MaxMSP-based solutions use various third-party OSC implementations. Quintet.net additionally uses TCP protocol where reliable messaging is required. More recently, several solutions which utilize Websocket point-to-point connection technology have emerged, such as NetCanvas which displays notation generated in MaxScore and dfscore which relies on Node.js server event distribution.

Odot framework middleware oriented messaging [9] is a welcome step towards network services abstraction. It wraps OSC protocol and provides transcoding to JSON, SVG and S-Expressions, as well as bindings to Javascript and Lisp. Landini [10] can also be classified as a form of a middleware as it creates an additional layer between music applications communicating over OSC. Landini implements a reliable, ordered message delivery protocol which detects packet loss and attempt recovery. Furthermore, it monitors network latency and applies OSC timing corrections for more accurate event synchronization. Quintet.net also provides proprietary strategies which deal with network jitter and latency.

Most of the existing compositional tools allow for the authoring of traditional symbolic notation. Support for graphical notation, custom symbols, staves or extended performance techniques, is commonly achieved by the layering of raster graphics on top of rendered symbolic notation. The maximum number of parts allowed in a score is either restricted explicitly or by the available application memory. Delivery of larger instrumentation, such as a full-sized orchestra, remains a significant challenge in all networked notation systems. Scores are typically composed off-line in a proprietary data model. If required, they are converted to one of the common notation formats for sharing with other notation applications. Currently, there is no clear winner between competing symbolic notation formats such as GUIDO, JMSL or MusicXML. Real-time notation generation and distribution is well supported, however, communication between heterogeneous applications normally requires transcoding of the native data models to OSC on all participating nodes [3].

For time synchronization between network nodes notation applications typically either rely on the system clocks or regular heartbeats sent from the master node. Network time protocol (NTP) is used by default on most LANs for system clocks synchronization, and by design, can cause inaccuracies of up to 100 ms between computer clocks. The application scheduling resolution which defines a minimum time interval between two scheduled events is normally defined in the milliseconds range.

2. DISTRIBUTED MUSIC COMPOSITION AND PERFORMANCE MODEL

A distributed system consists of a number of components which communicate and coordinate actions by passing messages over a computer network. A collection of independent components appears as a single integrated system to its users. The key goals of a distributed system include transparency, openness, reliability, performance and scalability. ZScore is a distributed notation system which ultimately aims to provide the following:

- Reliable and scalable low latency messaging with guaranteed data and control message delivery where critical events are delivered and executed with humanly imperceptible latency (sub 10ms compound network and application latency)
- Accurate performance synchronization across all networked nodes which includes the effects of network latency and jitter
- Complex symbolic, graphical or algorithmic notation authoring for any instrumentation (e.g. full orchestra) and type (acoustic, digital, algorithmic etc.)
- Efficient score data encoding and segmentation strategies which minimize transcoding and avoid packet loss during network transport
- Dynamic and interactive networked notation views on heterogeneous clients which allow for automated notation update, animation, position tracking, conducting signals and gestures display, event triggering etc.
- Real-time capture of a conductor or a musician's gesture and integration with the score and performance flow

A distributed composition and performance model enables heterogeneous components to interact over a message-oriented middleware (Figure 1).

2.1 Message-Oriented Middleware

High-throughput and low-latency messages are typically delivered over a messaging middleware which isolates application developers from the low level networking implementation detail and provides scalable and reliable message delivery. Message-Oriented Middleware (MOM) is a software or hardware infrastructure that provides message

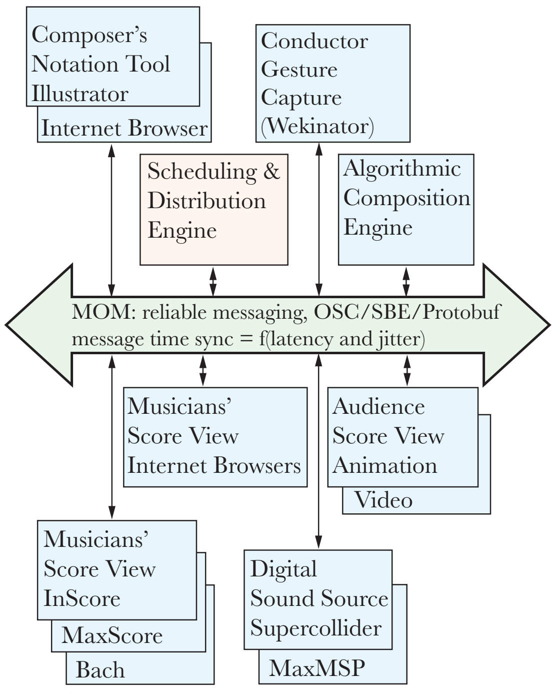


Figure 1: A distributed composition and performance system over message-oriented middleware (MOM).

delivery between distributed system components. Examples of the messaging middleware in music notation systems include Odot [9] and Landini [10]. Both are built on top of OSC protocol but offer different functionality: Odot is conceived as a framework for flexible inter-application communications, while Landini implements a reliable messaging protocol and offers improved time synchronization which takes network latency and jitter into account.

A common approach to make components with disparate data models communicate with each other is to build adapters which sit between the MOM client and each component. The role of an adapter is to translate the network data format (e.g. OSC) to the component's native data model. Language and platform neutral data serialization mechanisms such as Google Protocol Buffers (protobuf) [11] can make this process more flexible and streamlined. Humanly readable protobuf data structure definitions can be relatively easily imported and reused in other participating components. The component message processing should ideally be zero-copy and use preallocated, non-blocking data structures with minimal thread switching.

A music application message data model needs to be designed to allow for efficient data segmentation into chunks of up to 1500 bytes, which is the standard network Maximum Transmission Unit (MTU) size. This avoids data fragmentation during network transport and reduces the chances of packet loss and reordering.

2.2 Reliable UDP Multicast

Multicast is a network data routing method where a message is sent to a group of nodes from one or more sources. It can be described as one-to-many or many-to-many routing. Client nodes have to subscribe to a Multicast group to receive messages. Multicast protocols are almost always UDP and can be implemented at the application level. However, it is preferable to use network assisted delivery with Multicast enabled hardware routers where the application publishes a single message and the network router sends a copy of the message to all Multicast group members. This can significantly reduce network bandwidth usage and the sender's CPU load.

By default, UDP protocol does not guarantee message delivery. To resolve this issue, a number of reliable UDP based Multicast protocols have been proposed and implemented (PGM, TRDP, LBT-RM etc.). Unlike the TCP protocol, which sends an acknowledgment for each received packet, most of the reliable UDP Multicast protocols track ordered messages and only send a negative acknowledgment (NAK) if they detect a missing message, which is a much more efficient solution.

Several high-throughput, low-latency middleware libraries which provide reliable NAK based UDP Multicast have recently been released under the Open Source license. These libraries, such as Aeron [12], are designed to deliver millions of messages per second (at 40 bytes per message benchmark) with microsecond latencies. Aeron operates at OSI layer 4 (Transport) and can be thought of as a TCP replacement. Internally, it uses Simple Binary Encoding (SBE) so it would need transcoding to OSC where required. As OSC operates at OSI Layer 6 (Presentation), it would naturally fit as a layer on top of Aaron.

A fully scalable and reliable distributed music notation system should ideally incorporate the concepts mentioned above and provide a middleware-like network layer abstraction for delivery of OSC (or similarly) encoded messages over a reliable NAK based UDP protocol. The choice of UDP unicast or multicast protocol should depend on the message type and routing mode (one-to-one or one-to-many). The proposed middleware implementation should also synchronize the timing of message execution on each network node based on network latency and jitter.

2.3 Precise Network Time Synchronization

Precision Time Protocol (PTP) is similar to the NTP clock synchronization protocol mentioned above. However, on LAN networks, PTP can achieve sub-microsecond accuracy which is much more acceptable for networked notation software. The drawback is that PTP is not available by default on most computers and needs to be installed and configured on all network nodes. Open Source implementations PTPd and ptpd2is are available for all Unix-like systems which includes OS X. Alternatively, there are numerous PTP enabled hardware routers and switches which can be used on LAN as master clocks.

For Internet wide performance, the most accurate master clock that can be used is GPS time signal which has a theoretical accuracy of 14 nanoseconds. An example of music system synchronization over a GPS signal is The Global Metronome project [13]. It demonstrated that the combination of GPS for the master clock and NTP for LAN synchronization can produce sub-millisecond network node clock offsets. The main issue with The Global Metronome is that it requires access to GPS signal and, therefore, a clear view of the sky. The most convenient solution to this problem is to place The Global Metronome externally, link it via Ethernet cables to a LAN within a performance venue, and use NTP to synchronize nodes on the network.

In many cases it might be more practical to synchronize notation and event execution over a network in a tempo-relative rather than absolute time. In the simplest of scenarios, the master application instance would send synchronization events at regular intervals (e.g. every 96th of a whole note) to all participating nodes in order to set their internal tempo-relative position. However, excessive synchronization events may cause network saturation, so it would be more optimal to send master synchronization events at a lower resolution (e.g. every beat or a bar) and rely on network node system clocks for more granular scheduling and synchronization. To achieve acceptable synchronization accuracy, this approach would need network latency tracking per node and event timing adjustments similar to the Landini [10] implementation.

2.4 SVG-based Score Representation

A composition data format produced by the score authoring tool needs to contain enough information to enable notation rendering tools to reliably reproduce the intended score layout and perform any time related operations. Semantic data models, such as GUIDO or MusicXML, define both spatial and temporal context for notation rendering. The problem with semantic representations is that both the score authoring tool and all participating notation rendering clients need to fully support the composer's intended notational style. Due to a vast variety of contemporary composition styles, extended playing techniques and many contemporary composers' intentional disregard of standardization, it would be very hard to create a generic yet comprehensive semantic representation solution. As a result, composers and performers are increasingly turning towards systems which allow for constraint-free, graphical notation representations.

Computers can process graphical information either in raster or vector form. Raster format defines actual pixel values that need to be displayed on the computer screen. It is therefore fast to render, however, the file size can grow significantly for higher resolution images, which is not ideal for real-time network transport. Raster format also does not scale optimally, for example, downscaling can cause visible quality loss and it cannot be easily modified. Vector graphics on the other hand define relative x and y positions, paths and attributes such as color, thickness, fill, shape and curve which need to be interpreted by the host application. Therefore, it is slower to render but much more flexible to modify and scale. Scalable Vector Graphics (SVG) is an XML-based vector image format with support for interactivity and animations. It is an open standard supported by all major Internet browsers and many other graphical applications.

As demonstrated by Gottfried [14], SVG format can be successfully transcoded to OSC and this work has been integrated into Odot library. Adobe recently announced its support for Node.js which could allow for the network integration of their SVG authoring tool, Adobe Illustrator. SVG can be relatively easily extended with musical context such as the score element hierarchy and temporal information. The addition of time-space mapping allows for programmable synchronization and easier integration with notation rendering software such as InScore. Similar to computer font distribution, composers can create customized SVG symbol libraries which can then be distributed to networked clients and referenced in real-time scores. In this way, the amount of data that needs to be transferred in real-time can be significantly reduced.

Recently proposed music notation markup standards MNX and related GMNX (MNX-generic) [15] incorporate linking and time-space mapping of SVG graphics. Eventual adoption of these standards will allow for rendering and synchronization of SVG notation in standard Internet browsers.

2.5 Dynamic Notation View Design

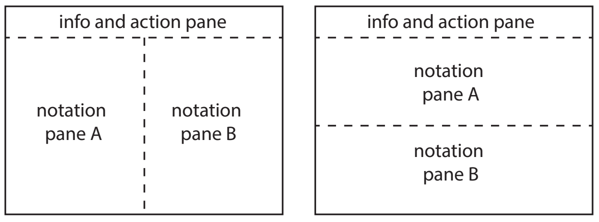
Unlike static notation, the dynamic view requires a carefully thought out refresh strategy which does not interfere with the currently played notation, providing enough time and space for musicians to prepare for the upcoming material. The refresh strategy needs to take into account network and rendering latency and ensure that notation updates do not interfere with the score continuity.

A good dynamic score front end design should fully utilize available screen real estate and provide a clear view of the notation, available actions, and any additional information musicians should be aware of during a performance. If delivery to heterogeneous platforms is required, the notation should be legible when scaled to any of the common screen aspect ratios (4:3, 16:9 and 16:10), screen sizes (10 to 17in) and resolutions (1024x768 to 2880x1800). As most of the common laptop types can only be used in the horizontal screen orientation, it is preferable to optimize notation layout for the horizontal screen viewing.

Dynamic scores with linear stave notation typically either use the full page or stave update as in Richard Hoadleys Calder's Violin where the entire view is replaced with new notation at once, or the continuous scroll as in Cat Hopes Longing and Luciano Azzigottis Spam where the notation moves continually from left to right. These strategies are suitable for particular score types. The full page refresh strategy does not provide much preparation time for musicians, especially where performance continuity is required at fast tempos. The continuous scroll strategy requires musicians to focus on a fixed point on the screen where the notation crosses a vertical synchronization line, thus reducing their capacity to look ahead and prepare for upcoming changes. It also requires continuous notation availability so it is not ideal for generative or free-timing scores.

2.6 Alternating Pane Layout

The alternating pane notation strategy aims to resolve dynamic notation update issues by providing familiar left-to-right and top-to-bottom reading direction and ample preparation time to musicians. The notation is stationary while several animated objects are transposed on top, indicating tempo, current position during performance and conducting gestures. Furthermore, it defines a clear time window for upcoming notation generation and transport. Figure 2 shows the main sections of the alternating pane layout for a full score and an instrumental part. In both cases, the notation view is divided into three main areas (panes). The top pane contains information about the score (title, part name, server status etc.), actionable buttons (for interaction with the server or other peers) and signaling information (tempo and start indicators). The main area, which takes approximately 80% of the screen real estate, is split into two notation panes, A and B. Each of the notation panes display an equivalent of a full score page or a single part stave. The notation is read left to right and top to bottom, the same as with static paper scores.



(a) Full Score (b) Part

Figure 2: Alternating pane layout

At any point of time during a performance, there is always one active and one preparatory pane. At the very start, pane A is active and pane B is preparatory. When the notation content in pane A is completed, pane B becomes active and pane A preparatory. Once the musician's focus is firmly moved to pane B, pane A is updated with the upcoming notation which is scheduled to be performed after pane B notation content is completed. The dynamic update process then continues in a similar fashion, following an ABAB... sequence.

2.6.1 Time restrictions and allowances

There are several time restrictions that should be taken into account when working with alternating notation panes. The notation to be played after the active pane notation is completed needs to be generated, transferred and rendered in the preparation pane by the time the active pane notation is around half way through its duration (T/2). This is to allow for performance continuity and preparation time for musicians. Furthermore, the preparation pane notation should only be refreshed once the active pane notation is played for an appropriate time duration (T1, e.g. longer than one beat) to allow for the musicians' focus switch. This means that the minimum time window for notation preparation (generation, network transport, rendering etc.) is from the active focus switch time (T1) to the active pane half duration time (T/2). If, for example, the composition tempo is 120 bpm and the active notation pane contains 5 bars with 4/4 time signature, then the minimum notation preparation time window is 4.5 seconds (T1 = 0.5s, T/2 = 5s). In most cases this would be sufficient time for the network transfer of graphical stave files or generation of algorithmic notation. This also creates clear timeline rules for the real-time notation generation and display when using alternating pane layout.

3. ZSCORE CURRENT STATE

ZScore is a distributed networked notation system which aims to satisfy requirements and implement the solutions outlined in previous sections. Currently, it is a collection of third-party and custom-made software. Composition authoring is done in Adobe Illustrator extended with the new set of JavaScript plugins. A proprietary Java engine was developed for score distribution and synchronization over a network, while InScore stand-alone clients are used for dynamic score rendering. The video "Composition for Networked Ensembles" [16] explains ZScore's main features and user trials with Moscow Contemporary Music Ensemble in March 2017.

3.1 Time-space mapping and synchronization

ZScore utilizes tempo-relative time synchronization described in section 2.3 which takes advantage of InScore's time-space mapping functionality. In this mode, the master server application sends regular synchronization messages carrying the global tempo-relative position to all clients. Each InScore client runs its internal clock and can synchronize independently if given a tempo and time-space mapping configuration. The master synchronization events effectively override internal client clocks with the global tempo-relative position and therefore ensure common time-space positions across the clients. The synchronization message frequency can be selected per composition and its choice depends on tempos and rhythmical structures used in the score.

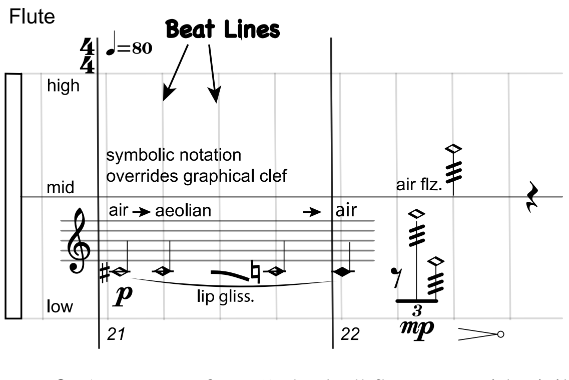


Figure 3: An excerpt from "Ukodus" flute part with visible Beat Lines used for time-space mapping.

The concepts of a Beat Line (BL, Figure 3) and Beat Division Unit (BDU) were introduced for easier time-space mapping and event scheduling workflows. The BDU value can be set to any fraction of a whole note (e.g. 1/8 which is equivalent to a quaver duration) and represents the lowest time resolution available for event scheduling and synchronization. The current minimum BDU value is 1/96. Beat Lines coincide with the bar beat onsets and contain information about their spatial and time position. The BL spatial position is set in terms of x and y coordinates on the score page while their time position is expressed in a number of BDU units from the composition start. Beat Lines are a form of proportional notation, however, there are no restrictions regarding consecutive spatial Beat Line positioning so they can be set individually to suite the score notation density. The time interval between Beat Lines is measured in BDU units. For example, if the BDU value is set to 1/8 then in a 4/4 bar each time interval between Beat Lines is 2 BDU units and in 5/8 bar with the beat division of (3 + 2)/8, the first beat consists of 3 and the second beat of 2 BDU units. Beat Line spatial and time position is exported with the score data and is used in InScore client for space-time synchronization.

3.2 Score Authoring

A vector graphics editor, Adobe Illustrator, is used for composition authoring at present. It allows for the unconstrained positioning of any notation type; export of SVG and multiple raster formats; import and creation of user defined symbol libraries; and is scriptable, which opens a range of opportunities for functional extensions and potential real-time network integration. Illustrator does not provide any musical context by default, therefore, a number of improvements have been implemented for more efficient music composition flows and integration with the networked software.

3.2.1 Hierarchical Layer Structure

Inspired by Gottfried [14], a hierarchical layer structure was created to provide a musical context in Illustrator and enable the automation of score creation and export. The hierarchical layer elements can contain one or more child layers (Figure 4). Currently, a Part layer has a one-to-one

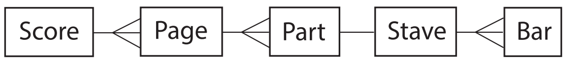


Figure 4: Hierarchical score layer entity relationships.

relationship to a Stave layer which contains all the graphical data required to display an instrument staff. The Bar layer contains all the graphical data required to render the notation and logical data required for synchronization such as tempo, time signature and beat line positions. The notation layer contains all the symbolic or graphical data required to display bar notation and can contain arbitrary notation types. The Illustrator layer structure is displayed in the screen capture in Figure 5.

3.2.2 SVG Symbol library

To accelerate symbolic notation generation, a set of custom symbols based on open-source LilyPond notation font were imported into Illustrator. Due to the flexibility of a vector graphic editor, it was straight forward to extend the library with custom symbols, such as different note head types and sizes and instrument fingering charts etc. (Figure 6). For variable length continuous lines, such as crescendo and decrescendo markings, a set of brushes were created and imported into Illustrator. An example of mixed symbolic and graphical notation created in Adobe Illustrator

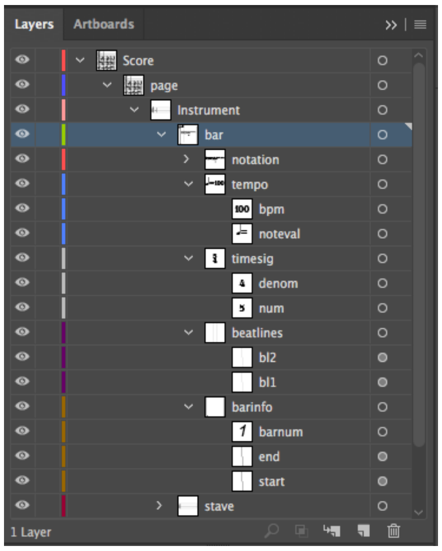


Figure 5: Adobe Illustrator ZScore layer structure.

with the help of ZScore tools plugin and music symbol libraries is displayed in Figure 8.

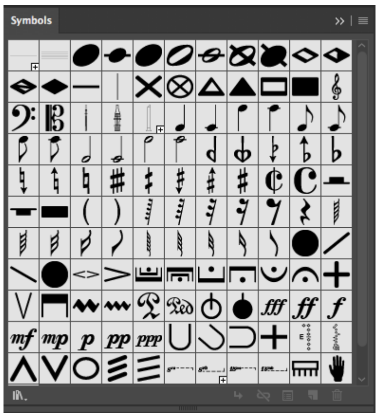


Figure 6: Notation symbol library imported and extended in Adobe Illustrator.

3.2.3 *ZScore Tools JavaScript plugin*

The set of JavaScript plugins were developed to speed up composition workflows and automate score export. Currently ZScore Tools includes: Layer, Page, Bar and Export plugins (Figure 7). The Layer plugin allows for Illustrator layer structure definition, editing, XML import/export and copying between one or more scores. Similarly, Page and Bar plugins help create required pages and bars at specified locations including any meter or tempo changes and Beat Line positioning. Export plugin allows for the export of the full score and parts in SVG or PNG graphical formats. In order to provide accurate and automated space-time mapping, the export process also creates necessary data for InScore in the required format:

$$([X_{start}, X_{end}[[Y_{start}, Y_{end}[)([BDU_{start}, BDU_{end}[)$$

where X and Y are space coordinates and BDU is the number of units since the beginning of the piece. The exported values define two-dimensional rectangles between

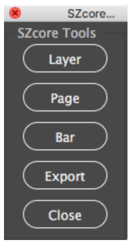


Figure 7: ZScore Tools plugins for Adobe Illustrator .

two Beat Lines and the corresponding start / end tempo-relative time. Additionally, the export process automatically collates score meta data required by the score scheduling engine. This information about time signatures, tempo changes, other score events, BL positions and related BDU values is currently stored in a csv (comma separated values) formatted file. An example of a score page authored in Adobe Illustrator is displayed in Figure 8.

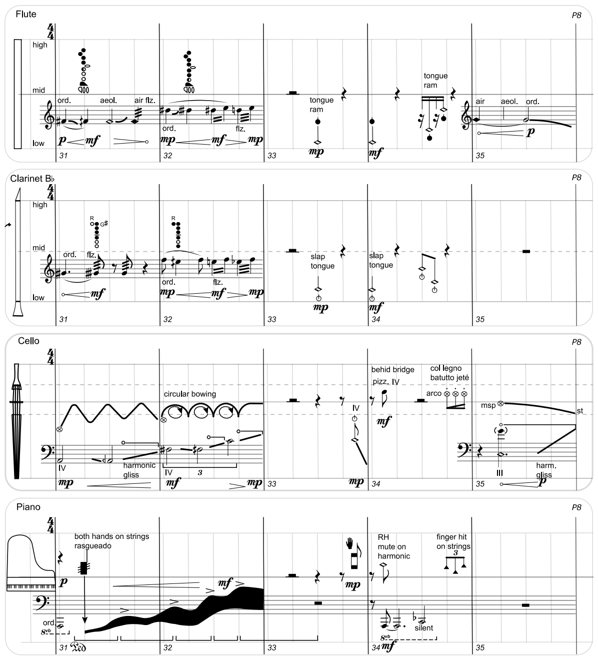


Figure 8: An example of the score page created in Adobe Illustrator demonstrating mixed clef and notation styles.

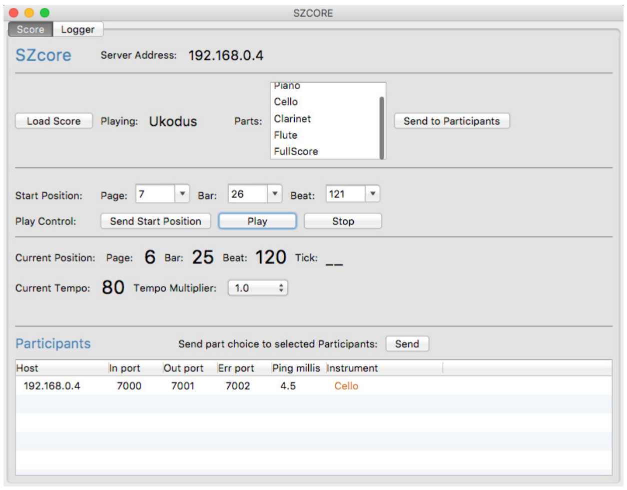
3.3 Distribution and scheduling engine

The central hub in charge of scheduling and distributing the score data over a network in real-time is the ZScore server and management client written in Java programming language. ZScore network management client (Figure 9) can import and parse score definition data exported from the ZScore Tools Illustrator plugin and submit to the server, creating an internal representation of the score metadata. The internal score metadata model mimics layer hierarchy shown in Figure 4.

The server listens to notation client connections and sends information about available parts to all connected clients. When musicians select individual parts on their notation client, the server associates the selected score part with the client's host address. From then on, all messages related to a particular part will be routed to the associated client host.

The server internal scheduling resolution is 1 millisecond with a maximum measured deviation of 0.8 milliseconds. It translates local absolute time to a tempo-relative value expressed in BDU units and evaluates any scheduled score events accordingly. The server supports multiple transports with different meters and tempos which allow for compositions and performances of polyrhythmic and polymetric scores. Events can be preloaded in score data or dynamically added during a performance according to the timing rules discussed in chapter 2.6.1. ZScore's real-time functionality is bounded by the timing rules, so any notation generated during the performance needs to be available in the time window defined by T1 and T/2 boundaries. The current server implementation utilizes LMAX Disruptor [17] which allows for high throughput lock-free data processing with microsecond latencies. At present time, OSC messages are delivered over UDP unicast.

The network management client (Figure 9) can load and start the score from any position defined in terms of the Page, Bar and Beat number on all participating networked clients. It also allows for tempo multiplication in the range from 0.1 to 2.0 for rehearsal purposes. Tempo can also be dynamically modified during the performance.



SZcore Server Address: 192.168.0.4

Playing: Ukodus Parts: Piano, Cello, Clarinet, Flute, FullScore

Start Position: Page: 7 Bar: 26 Beat: 121

Current Position: Page: 6 Bar: 25 Beat: 120 Tick: __

Current Tempo: 80 Tempo Multiplier: 1.0

Participants — Send part choice to selected Participants:

Host	In port	Out port	Err port	Ping millis	Instrument
192.168.0.4	7000	7001	7002	4.5	Cello

Figure 9: ZScore distribution and scheduling engine management client screen shot.

3.4 Dynamic notation rendering

InScore is currently used for dynamic notation rendering due to its networking capabilities, native OSC support, time-space mapping, built-in interactivity and scripting engine. It also supports multiple graphical file formats, although its SVG support depends on the underlying Qt library so some features such as symbol referencing via the xref attribute were not available at the time of the writing. To work around this, all score pages were exported and distributed in PNG raster format.

Once the ZScore startup file is opened, all communication with the server can be done directly from InScore client. The start-up file contains configuration for the alternating pane layout and the set of JavaScript functions which handle all interactive tasks. An example of the dynamic score with alternating pane layout is presented in Figure 10. The active notation pane is highlighted with red borders while the preparatory pane is slightly dimmed. Several features aiming to replace some of the conducting gestures have been added. The signalling traffic light at the top left corner aims to draw the attention of the musicians and provide an indication of the starting tempo. The animated position line, visible as the light green line on the upbeat leading to bar 28 (Figure 10), indicates the current real-time position within the score. It also has the attached tempo indicator ball which is visible as the red circle on top of the stave in Figure 10. For ease of orientation, the actual starting position is marked with a light purple line (on the first beat of bar 28 in Figure 10). The starting position can be set to any Beat Line from the network management client (Figure 9). When the score performance is started in the network man-

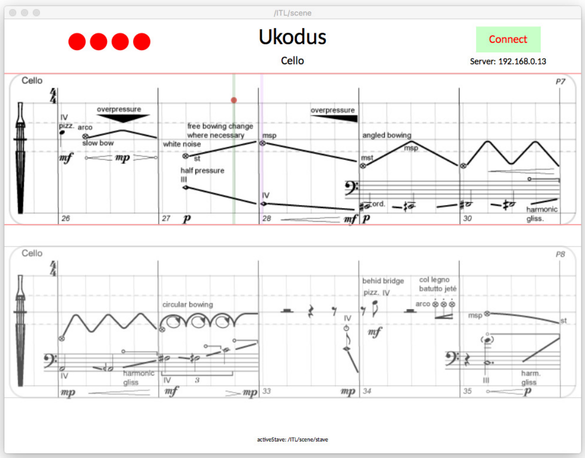


Figure 10: InScore view of the cello part dynamic notation in alternating pane layout.

agement client (Figure 9), the traffic light signal flashes in the starting tempo frequency and the position line starts moving from left to right, indicating the current position on the screen. The attached tempo indicator ball starts simulating conductor signals with vertical movements calculated from the simplified pendulum motion formula where the ictus plane is at the top of the stave. The current position line always starts from the upbeat before the selected starting position in order to mimic the familiar conductor gesture on start. When the position line reaches the penultimate beat of the active pane stave, the preparatory pane current position line will start from the upbeat at the same time to provide notation view continuity.

3.5 User trials

In March 2017, a trial session was held with the Moscow Contemporary Music Ensemble at Goldsmiths, University of London [16] to test ZScore's technical functionality and user experience in a workshop situation. The piece Ukodus for flute, clarinet, cello and piano quartet was composed by Slavko Zagorac for this occasion. A combination of laptops and tablets running both OS X and Windows operating systems were used for the musicians' front ends while the Java scheduling engine and the management client were hosted on OS X laptop. The musicians' devices were networked over the dedicated tri-band wireless hardware router while the scheduling engine laptop was connected directly to the router over the ethernet cable. The maximum round trip latency recorded during the workshop was 12 ms. Apart from some intermittent instability of the notation clients on older Windows OS versions, there were no significant technical issues during the performance.

The system setup was relatively time consuming as all musicians' devices needed software installation and WiFi network configuration as well as the initial functional testing. Due to the familiarity of the notation layout, these highly skilled musicians were able to quickly grasp the technical aspects of the system and perform the entire piece without stopping on their first sight-reading attempt. The mixed clef staves and spatial layout of the notation did not require any additional explanations, nor did they present any particular problems during the performance. The musicians feedback on the alternating pane layout usability and overall system performance was positive and encouraging.

4. CONCLUSIONS AND FUTURE WORK

This successful user trial of the current ZScore implementation has reinforced the case for SVG-based complex notation representation and the alternating pane layout dynamic notation view design. The downsides of the proposed approach are SVG authoring complexity and the considerable development effort required to enable distributed system interoperability. The ZScore Tools Adobe Illustrator JavaScript plugins have significantly accelerated the SVG authoring process while further planned automation should make the composition process even more streamlined. The distributed system complexity can be encapsulated on the server side and within the client API implementation, thereby simplifying the end user experience. Future ZScore development plans include the reliable UDP multicast middleware integration and SVG-based notation authoring and rendering in standard Internet browsers, thus enabling the utilization of any mobile device without any additional software installation. The planned publication of ZScore OSC API for score data scheduling and distribution is expected to encourage collaboration and open the system to third-party software integration. The machine learning software Wekinator [18] integration would enable conductor gesture capture and mapping to score events, thereby allowing for humanized real-time tempo and dynamic changes. The existing beat tracker could be enhanced to provide a much richer representation of the conductor gestures on musicians' screens through animation of its velocity, shape and color. Similarly, direct audience participation and integration with the score decision logic during the performance may be achieved through proprietary audience score views on mobile devices. This would allow not only composers, but also conductors, performers and audience members to significantly impact the composition flow and even create new compositional material through algorithmic functions. The expectation is that these technical innovations may lead to new creative possibilities in music composition and performance.

5. REFERENCES

[1] D. Fober, Y. Orlarey, and S. Letz, "Programming Interactive Music Scores with INScore," in *Proc. SMC Sound and Music Computing conference*, 2013, pp. 185–190.

[2] G. Hajdu, "Quintet.net: An Environment for Composing and Performing Music on the Internet," *Leonardo Music Journal*, vol. 38, pp. 23–30, 2005.

[3] S. James, C. Hope, L. Vickery, A. Wyatt, B. Carey, X. Fu, and G. Hajdu, "Establishing connectivity between the existing networked music notation packages Quintet.net, Decibel ScorePlayer and MaxScore," in *Int. Conf. on New Tools for Music Notation and Representation – TENOR 2017*, 2017, pp. 171–181.

[4] G. Hajdu and N. Didkovsky, "MaxScore - Current State of the Art," in *International Computer Music Conference*, 2012, pp. 156–162.

[5] A. Agostini and D. Ghisi, "Bach: An environment for computer-aided composition in Max," in *International Computer Music Conference*, 2012, pp. 373–378.

[6] C. Hope, L. Vickery, A. Wyatt, and S. James, "The Decibel ScorePlayer – a digital tool for reading graphic notation," in *Int. Conf. on New Tools for Music Notation and Representation – TENOR 2015*, 2015, pp. 59–69.

[7] R. Constanzo, "Dfscore: networked notation software," last accessed: 29 Dec 2017. [Online]. Available: http://www.rodrigoconstanzo.com/dfscore/

[8] G. Burloiu, A. Cont, and C. Poncelet, "A visual framework for dynamic mixed music notation," *Journal of New Music Research*, vol. 46, pp. 54–73, 2017.

[9] J. MacCallum, R. Gottfried, I. Rostovtsev, J. Bresson, and A. Freed, "Dynamic Message-Oriented Middleware with Open Sound Control and Odot," in *International Computer Music Conference*, 2015, pp. 58–65.

[10] J. Narveson and D. Trueman, "Landini: a networking utility for wireless lan-based laptop ensembles," in *SMC Sound and Music Computing conference*, 2013, pp. 309–316.

[11] Google, "Protocol Buffers (protobuf) documentation," last accessed: 29 Dec 2017. [Online]. Available: https://developers.google.com/protocol-buffers

[12] M. Thompson, "[12] Aeron, messaging middleware documentation," last accessed: 29 Dec 2017. [Online]. Available: https://github.com/real-logic/aeron

[13] R. Oda and R. Fiebrink, “The Global Metronome: Absolute Tempo Sync For Networked Musical Performance,” in *Int. Conf. New Interfaces for Musical Expression*, 2016, pp. 26–31.

[14] R. Gottfried, “SVG to OSC Transcoding: Towards A Platform for Notational Praxis and Electronic Performance,” in *Int. Conf. on New Tools for Music Notation and Representation – TENOR 2015*, 2015, pp. 155–162.

[15] W. M. N. C. Group, “MNX Draft Specification,” last accessed: 29 Dec 2017. [Online]. Available: https://w3c.github.io/mnx/

[16] S. Zagorac, “Composition for Networked Ensembles,” last accessed: 29 Dec 2017. [Online]. Available: https://youtu.be/ioqNP4qg6JQ

[17] LMAX, “Disruptor documentation,” last accessed: 29 Dec 2017. [Online]. Available: https://lmax-exchange.github.io/disruptor/

[18] R. Fiebrink, “Wekinator: machine learning software,” last accessed: 29 Dec 2017. [Online]. Available: http://www.wekinator.org/