MAXSCORE: RECENT DEVELOPMENTS

Georg Hajdu
Hamburg University of Music and Drama
georg.hajdu@hfmt-hamburg.de

Nick Didkovsky
Algomusic.com
nick@didkovsky.com

ABSTRACT

This paper presents recent development in MaxScore and its peripheral applications. These developments include:

- Adding new functionality to the core mxj object including details on our implementation of an undo/redo stack, new licensing models, custom beam groups, and other new features.
- Strategies to achieve proportional notation, with a look to the future.
- Expanding the feature set of the MaxScore and LiveScore Editors which include new style editors for the design of non-standard clefs, tablature notation and Bohlen-Pierce microtonality.
- Providing tools for greater compatibility with other third-party developments such as bach, Mira, the Scala Archive as well as the conTimbre sample library and its ePlayer.
- New peripheral components for guided improvisation and situated scores.
- Strategies to achieve proportional notation, with a look to the future.

1. INTRODUCTION

MaxScore is a notation package for Max consisting of a core mxj object referred to as "MaxScore object" implementing the Java Music Specification Language, and a number of peripheral abstractions and devices [1]. A complete music editor with menus and floating palettes exists in form of the MaxScore Editor. Some of MaxScore's functionality has been integrated under the moniker LiveScore into Ableton Live via the Max for Live API. MaxScore shares some features with the bach and cage computer-aided composition packages for Max [2] and to a lesser extent with Inscore [3], but is set apart from them by its capability to render to arbitrary contexts as the engines for data handling and graphics rendering are separate entities.

2. RECENT ADDITIONS TO THE MAXSCORE OBJECT

A number of new features have been added to MaxScore. Some of these, like the Undo/Redo stack, were implemented in the core JMSL engine (Java Music Specification Language) that powers MaxScore, while others like new low-cost licensing options, primarily affect MaxScore.

2.1 Undo/Redo

JMSL's Score package originally implemented a fine-grained undo/redo mechanism using a Command Pattern [4]. With this scheme, a user action that affected a *score*, such as doubling the duration of a *note*, was encapsulated in a *command*. The *command* included an undo operation, in this case, halving the duration of the note. *Commands* were added to a stack, and a user's Undo action would pop the top *command* off the stack and execute its custom *undo* operation.

The Command Pattern implementation of an Undo stack worked well for the subset of actions that had *commands* implemented for them. However, as JMSL and MaxScore expose a general API to the user as well as a user interface (indeed Max itself is a GUI), it became difficult to decide at what level undo/redo should be implemented. If the user mouse-clicks a staff, a new *note* is inserted and was undoable because the UI action was wrapped into the Command Pattern. However, the same user may use the API to insert a *note* using the *addNote* message in Max. The *addnote* message is an elemental API call that does not trigger an undoable *command*. Furthermore, the Max user may patch together an arbitrarily complex network of similarly fundamental API procedures which insert, delete, and transform existing notes. We wanted to give that user a functioning *undo/redo* stack and decided something closer to the "Memento Pattern" would be appropriate.

The current *undo/redo* scheme in JMSL's Score package addressed these issues by building a stack of *score* clones instead of a stack of undoable commands. Actions that altered the contents of a *score* trigger the saving of the entire Score to a cache. Undo replaces the current score with the clone at the top of the stack. At first, we were concerned that the user would experience unacceptable pauses while editing as the score was being written to the cache, but in practice we discovered that writing a *score* to the disk cache is almost unnoticeable, even with large scores. The MaxScore user has also been given more control over the *undo/redo* stacks, with new saveToUndoStack, undo, and redo messages. The saveToUndoStack message takes a snapshot of the score's current state and saves it to the undo stack. This allows the user to make arbitrary programmatic changes to the score, i.e. non-UI commands that do not trigger UNDO stack snapshots, save to the undo stack and undo the activity if desired.

This scheme required a layer of programming to provide the user with the sense that the score displayed by an undo/redo event felt like the same score, even though it was actually replaced with a clone. The starting measure of the layout, for example, had to be cached along with the score to restore the current layout. We may add other such features such as restoring the current note selection.

2.2 Proportional Notation

User demand for proportional notation is currently addressed with two different strategies. One method is using linear measure widths and adjusting the base of Blostein/Haken justification algorithm, which is responsible for the influence of duration on horizontal note placement [5]. Another technique is to use invisible rests to fill in time space. The latter is more accurate but a more cumbersome solution.

Figure 1-3 show various layout schemes in JMSL. The default layout uses flexible measure widths, where measures are algorithmically widened to accommodate denser note layout. Changing the layout to linear measure widths ensures that all measures have the same width, a prerequisite to proportional notation. Changing the Blostein/Haken justifier algorithm's base from a default of 0.7 to 0.4 results in a layout that comes close to proportional notation.

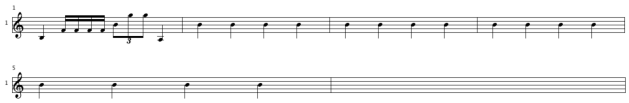

Figure 1. JMSL Score's default layout uses flexible measure widths and a Blostein/Haken justification base 0.7.

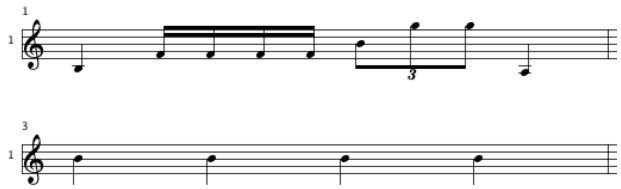

Figure 2. Changing JMSL's measure width to Linear results in all measures being the same width. Notice that the quarter notes in the measure 3 do not align with the notes in measure 1 due to justification algorithm JMSL uses.

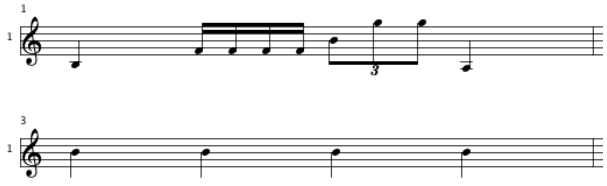

Figure 3. Linear measure widths and a Blostein/Haken justification base of 0.4 comes close to proportional notation. Notice that quarter notes in measure 3 align closely with the notes in measure 1.

An alternative technique to achieve proportional notation is to choose a fine time granularity, say 64th notes, and filling a measure with Linear width with these notes. These notes will all be spaced evenly, and the user may change some of them to invisible rests, either by hand or preferably using a straightforward algorithm. Figure 4 shows this technique, which additionally made stems and beams invisible.

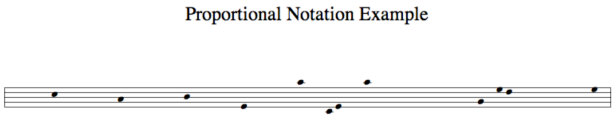

Figure 4. This example ensures accurate time-based note placement required by proportional notation. Horizontal space between note heads is occupied by evenly spaced invisible rests.

A useful plug-in to generate proportional notation using this technique could be created in a straightforward way. With traditionally notated durations as input, the plugin ought simply to quantize their durations to the nearest 64th note to fit into this scheme. JMSL and MaxScore's "Unary Copy Buffer Transform" API (reference to JMSL paper) would serve well as the plug-in platform.

We are developing a new strategy to achieve proportional notation using an underlying data representation and a layout manager that is robust and flexible. This approach will address the shortcomings of the two approaches discussed above, and will be visually precise and free of an underlying quantization grid.

2.3 New MaxScore licenses

We have developed new license levels to accommodate users' needs. The latest is the low-cost ($9.90) LIVE_LITE license, used by composers who wish to use MaxScore in the Ableton Live environment without the ability to edit in MaxScore or develop using JMSL's Java API. JMSL's license scheme accommodates new license types transparently, whose semantics are interpreted programmatically. We have found that a fair number of new users have been very satisfied with the limited but focused functionality of using the new LIVE_LITE license to bring traditional notation into Ableton Live.

2.4 Beam Grouping

Beam grouping is a new MaxScore feature, delivered by JMSL's "BeamGroupTransform", a NotePropertiesTransform which is addressed from Max via a few simple messages sent to MaxScore. A BeamGroup is a specification of how to group notes in a particular time signature. Notes in a measure of 7/8, for example, may be beamed as groups of 2+3+2 or as 3+2+2 or other combinations. The MaxScore user specifies a BeamGroup with the message addBeamGroup <timeSigUpper timeSigLower g1 g2 g3 g4...> , where g1+g2+...+gn add up to the number of beats in the measure as specified by the upper number in the time signature. For example, the following message:

addBeamGroup 7 8 2 3 2

…specifies that a measure of 7/8 should be grouped as 2+3+2 while the following message:

addBeamGroup 6 8 3 3

…specifies that a measure of 6/8 be grouped as two groups of three. Once the user specifies all such beam grouping preferences, the beamGroupTransform message executes this custom beaming on all selected notes, as Figure 5 illustrates.

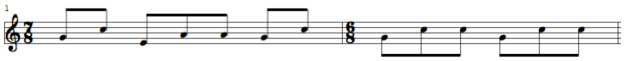

Figure 5. Results of beam grouping, where a measure of 7/8 is grouped as 2+3+2 and a measure of 6/8 is grouped as 3+3.

3. MAXSCORE EDITOR: NEW FEATURES AND TOOLS

3.1 Staff Styles

Staff Styles have been implemented in the MaxScore Editor to enable different representations of musical content, primarily for non-standard notation. Staff Styles rely on a plugin structure which has been described in [6]. The plugins talk to the MaxScore object via a JavaScript object mapping pitch to an arbitrary position of on a staff irrespective of its actual frequency and keeping track of the latter by using a MaxScore *note dimension* called originalPitch. Plugins for notation in the context of the Bohlen-Pierce scale and other microtonal scales have already been created, yet, recently, three new Staff Styles editors (which allow greater variability and flexibility) have been added to the repertoire.

3.1.1 Clef Designer

The JMSL API features a limited number of clefs, namely treble, alto, tenor, bass and percussion clef. The Clef Designer (Figure 6) was created to overcome this impasse by adding another 15 clefs or multi-clef staves (such as the OpenMusic-style FFGG staff [7], Figure 7) as well as providing an interface for the creation of non-standard, user-defined clefs (see Figure 8).

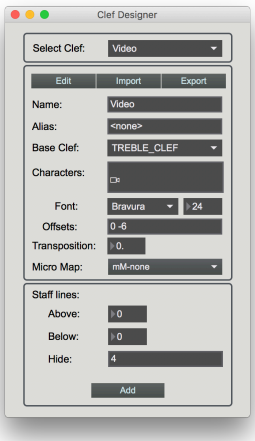

Figure 6. Screen shot of the Clef Designer GUI.

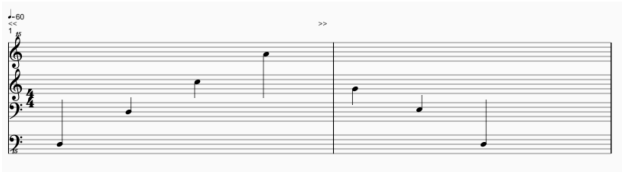

Figure 7. The FFGG staff settable in the Clef Designer.

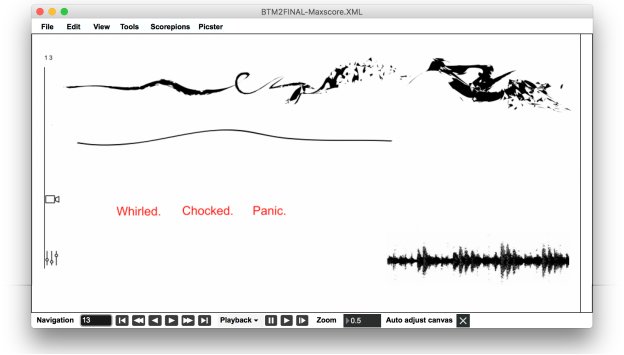

Figure 8. A score by Vietnamese composer Luong Hue Trinh using a non-standard clef for text display on the third staff.

3.1.2 Tablature

Tablature is supported by another editor allowing users to define an arbitrary number of strings as well as fret intervals, both set to pitches in floating-point precision. This, for instance, permits tablature notation of the 10-string 41-tone guitar used in Hajdu's piece *Burning Petrol* [8] (Figure 9).

The editor features 21 presets from monochord to 19-course theorbo which can be used as templates for user-defined tablatures. As with the Clef Designer, user-defined tablatures can be saved into scores they been created for, from where they can be exported as files and imported to other scores. Notes can be dragged to other strings for alternate fingerings and shifted up and down by using arrow keys.

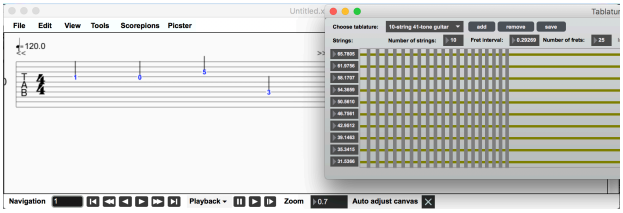

Figure 9. The GUI to the Tablature editor featuring the preset for the 10-string 41-tone guitar (foreground). A short score in the corresponding tablature is seen in the background.

Two things are still on our agenda:

1. The implementation of an intelligent algorithm for fingering, both vertically (chords) and horizontally (melody and chord progression) using constraints and/or neural nets [8].
2. Adaptation of the editor to just-intonation instruments with individual, unequal fret positions. This

poses a particular challenge as two frets can be close together representing tuning alternatives for the same scale degree (e.g. 16/9 and 9/5) or spaced widely apart, possibly even skipping a scale degree. Personal communication with guitarist John Schneider emphasized that just-intonation guitar community is still far from defining a common standard for such scenarios.

3.1.3 Bohlen-Pierce microtonal notation

The Bohlen-Pierce scale is a *macrotonal* tuning dividing the just twelfth ("tritave") into 13 steps. It exists in just and equal-tempered versions, the latter with a step size of 146.3 cents. The chromatic Müller-Hajdu notation has been described in [10] and implemented in the MaxScore Editors as Staff Style. Two subdivisions of the BP scale deserve particular attention:

1. BP triple scale also known as 39ED3
2. BP quintuple scale (65ED3) (whose step size deviates just 0.03 cents from 41-tone equal temperament)

We created an editor which accommodates the aforementioned microtonal BP scales using accidentals from the Bravura font set (Table 1), as partially suggested by clarinetist Nora-Louise Müller.

Steps (195ED3)	Glyph	Reference (see SMUFL [9])
0		natural
5	𝄲	accidentalXenakisOneThirdToneSharp
10	𝄳	accidentalXenakisTwoThirdTonesSharp
-5	𝄬	accidentalWyschnegradsky3TwelfthsFlat
-10	𝄫	accidentalWyschnegradsky9TwelfthsFlat
3	𝅥	accSagittal11MediumDiesisUp
6	𝅥𝅮	accSagittalSharp
9	𝅥𝅯	accSagittalSharp11MUp
12	𝅥𝅰	accSagittalDoubleSharp
-3	𝅦	accSagittal11MediumDiesisDown
-6	𝅧	accSagittalFlat
-9	𝅨	accSagittalSharp11MDown
-12	𝅩	accSagittalDoubleFlat

Table 1. The accidental set for the 39ED3 and 65ED3 Bohlen-Pierce microtonal scales. The indices in the left column refer to the LCM of both scales.

3.2 Expressions

The MaxScore object offers a variety of options to expand its feature set via *note dimensions* and *rendered messages* [1]. Note dimensions are referred to by a numeric or symbolic index and a floating-point value and need to be defined before notes are added to a score. These values are being added to a note event in the order of their index and sent out of the object during playback. In turn, rendered messages consists of single strings (or symbols in Max lingo) applied to notes, intervals, staves as well as measures and are sent out when the object renders to its drawing context. Expressions offer a way to combine the two, so that the messages to be sequenced (an action) are associated with a graphical element symbolizing the action to be performed. As an action can be more than just a single float (e.g. an OSC message with a number of values) the built-in limitation of the MaxScore object was overcome by writing messages into a buffer (a Max coll object) and referring to them by an index sent out during playback (the floating-point value). The buffer is created and updated whenever a score is loaded or events or Expressions are added to it. Expressions are created via the addRenderedMessageTo… family of messages, e.g. addRenderedMessageToSelectedNotes 0 0 "expression Coda[0] 153.3ocSN1kCBBCDD1ixDN.jVPdw27B3c.jARSvVrs.IR3EuQdx7J31HlXxlM67s+MdZaoms3DVgRxc99Fnx2ihppbnSXmMhryCNSfVbgKYn3HJjNKdSjQWZ5RcNd+7EJE7HsAyCJKqwFB79DsW+6OqfslnyKkMic3FCg5dJpHCQLWOLkDZk5qQz+bjZmk.XvXYsWt+1gOvm.fiM". The two zeros in the message refer to the initial coordinates of the part to be rendered (i.e. graphics), which can later be adjusted by dragging the graphics to another position, while the long string after Coda(0) is a Base64-compressed Max dictionary (Example 1).

```
{
        "rendered" :        {
                "0" : [ "frgb", 0.0, 0.0, 0.0, 255.0 ],
                "1" : [ "font", " Times Italic", 18 ],
                "2" : [ "writeto", 0.0, 31.0, "dal niente" ]
        }
,
        "sequenced" :       {
                "0" :               {
                        "editor" : "bpf",
                        "message" : "/amplitude",
                        "value" : [ 1000.0, 0.0, 1.0, 0.0,
0.0, 0, 1000.0, 1.0, 0, "linear" ],
                        "autorender" : "false"
                }
        }
}
```

Example 1. An Expression consists of "rendered" and "sequenced" messages.

Example 1 shows an example for an Expression in JSON format. It consists of the two keys "rendered" and "sequenced", each holding an arbitrary number of entries. The sequenced dictionaries contain the keys "editor", "message", "value" and "autorender", with the latter denoting whether or not the MaxScore editor should try to render the values irrespective of the drawing instructions given by the "rendered" dictionary.

The MaxScore Editor currently offers three editors (Figure 10) for the creation of sequenced messages:

- a generic editor with a text field for the message name and another for its values;
- an editor for breakpoint functions;
- an editor for DJster [11] parameter settings.

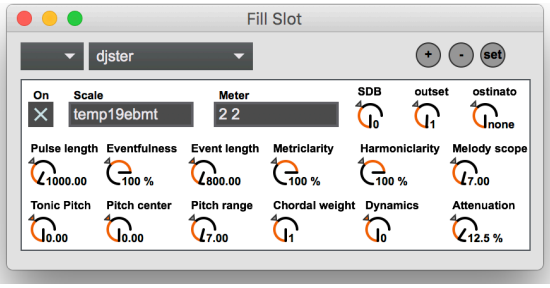

Figure 10. Example of an editor for sequenced messages.

In a MaxScore XML file, Expressions are stored as the Message attribute of a <userBean> element (of a <note> or <interval> parent element). It consists of a string containing a compressed Max dictionary preceded by "expression" and a symbolic reference (Example 2).

```
<note NOTEDUR="2" TUPLET="0" DOTS="0" ACCINFO="0"
DURATION="1.0" PITCH="71.0" VELOCITY="0.5"
HOLD="0.800000011920929" BEAMEDOUT="false" GLIS-
SOUT="false" TIEDOUT="false" ACCPREF="0" ACCVISPOLI-
CY="0" ALTENHARMONIC="false" DYN="0" SLU-
ROUT="false" ISGRACENOTE="false" GRACENOTESEPARA-
TIONSCALER="2.0" LEDGERLINESVISIBLE="true"
WEDGE="none" OTTAVA="none" MARK="0" TEX-
TOFFSETX="0" TEXTOFFSETY="0" NOTEHEAD="0" NOTE-
HEADSCALE="1.0" VISIBLE="true" NOTEHEADVISI-
BLE="true" STEMVISIBLE="true" OVERRIDELEVEL="-1"
ISOVERRIDELEVEL="false" STEMINFOOVERRIDE="false"
STEMINFO="2" TEXT="" >
<dim index="4" value="0.0" name="EventFlag" />
<dim index="5" value="71.0" name="originalPitch" />
<dim index="6" value="1.0" name="index" />
<userBean CLASS-
NAME="com.softsynth.jmsl.score.util.RenderedMessageBean"
Name="RenderedMessageBean_note-sel" Message="expression
Coda[0]
159.3ocSO9lBBCCCE2ixid.lsSFHdC76dAxVyjBccy9G8Cic2MU
mfPHj7KgWdIxAKGYKtfUnk7X7dOzM6QaWWCLU7bHC0M
2Dmv0L4cCJXNiVYzqnKy4455mLMPYIOBNNjYE1PheT3vve
WXEr0kmiRY+xHDESzcV5NRSKdWtXY7j7kJxn0eMh4miz6rJ
.dWfoHnhH2mGo5TxmX4vaGdyE.8yM" Xoffset="0.0" Yoff-
set="0.0" >
</userBean>
</note>
```

Example 2. Expression are stored as compressed Max dictionaries. The <dim> element with index="6" attribute contains the reference (value="1.0") to the message contained in the buffer.

3.2.1 Button Mode

MaxScore has two modes for mouse interaction:

- one for editing notes and other score elements;
- one for repositioning and deleting Expressions and Picster [1] elements.

These modes can be toggled by using caps lock. When clicking on a graphics element in Picster mode, it is highlighted by a red bounding rectangle. We are planning to implement a *button mode* which would allow a Mira user (see section 4.2) to use Expressions as interactive score elements, sending out "sequenced" messages upon clicking on them—thus bridging the gap between score and interface[1]. Jacob Sello's Hexenkessel project [12] is an excellent example for such an approach developed at the HfMT Hamburg.

3.3 Searchable Scala database

The Scala Archive is the world's largest collection of microtonal scales maintained by Manuel op de Coul[2]. It is supported by an increasing number of third-party applications such as the Kontakt sampler and the MuseScore notation editor, among many others.

The Scala Archive currently contains more than 4500 scales and tunings—a number that makes informed choices staggeringly difficult. We have therefore created a searchable database via the Max SQLite JavaScript implementation. Searches can be performed according to

- number of *steps*,
- *pitch content* in terms of both floating-point and rational numbers
- *strings* contained in the comment section of a Scala file.

The database is integrated into the Scala Browser, a Max patch we refer to as a "virtual keyboard". The interface of the Scala Browser displays notes of the scales which can be clicked on to add notes to a score or change their pitch (Figure 11). A MIDI keyboard can be used instead of or in addition to mouse clicks.

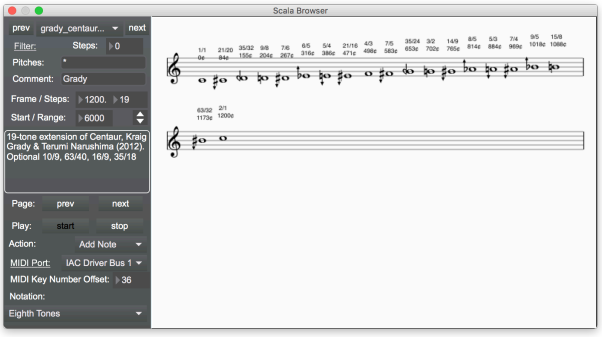

Figure 11. The Scala Browser virtual keyboard containing the Scala Archive as searchable database. The browser filters all scales containing the "Grady" search string in their comment section.

[1] At the HfMT, we have dedicated the UMIS research project (Unified Musical Instrument Surfaces) to the idea that an instrument can also act as a controller and score display.

[2] http://www.huygens-fokker.org/docs/scales.zip

3.4 MaxScore.NetCanvas

MaxScore.NetCanvas is a Java-based peripheral component of MaxScore developed by Benedict Carey, designed to render scores in web browsers via web socket connections [15].

In the latest update the communication between MaxScore and MaxScore.NetCanvas now occurs entirely within the Max environment, doing away with the previous reliance on inter-application messaging. This has the effect of speeding up communication between Max and remote clients, and simplifies the setup procedure for MaxScore users. The set of messages accepted by the MaxScore.NetCanvas object has expanded to include messages specific to part rendering, the behavior of cursors and control of the server (configuration, starting and stopping the websocket server and the new fileserver for serving the html client files). Max users can run multiple instances of MaxScore.NetCanvas concurrently. The new helpfile (Figure 12) contains information about how to use the new MaxScore.NetCanvas abstraction and accompanying mxj object; the source is available on Github.

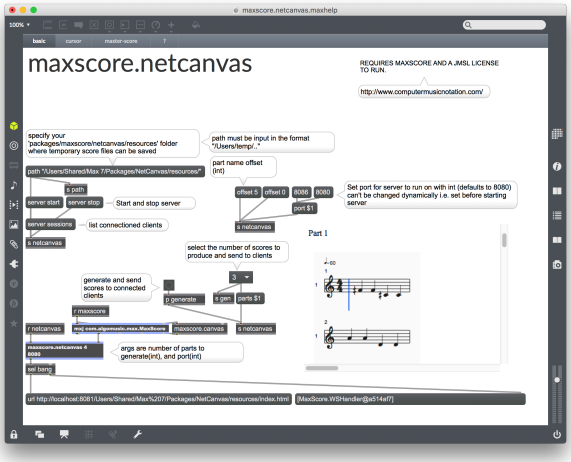

Figure 12. The helpfile to MaxScore.NetCanvas.

4. INTERFACING WITH OTHER THIRD-PARTY DEVELOPMENTS

4.1 bach compatibility

bach is a Max package developed by Andrea Agostini and Daniele Ghisi which has become the de facto standard for computer-aided composition in Max for its thorough integration and plethora of tools such as the bach.roll and bach.score notation objects [2]. Being modelled after IRCAM's OpenMusic environment its externals implement a data format which due to its similarity to the LISP syntax is called llll (lisp-like linked lists). Despite some overlap, bach.score and MaxScore occupy different niches of real-time notation ecosystem. While bach.score excels at continuous tempo changes, polymeter, nested tuplets and some GUI operations, MaxScore shows more flexibility in how it performs graphics rendering. By separating the mxj object designated for data handling from its drawing context, MaxScore can be used for generating PDFs, for data mapping as well as rendering to various 2D and 3D contexts. We therefore have created the maxscore.bachScoreToMaxScore abstraction capable of bridging bach.score with the MaxScore Editor with the aim to preserve as much information as possible (i.e. by translating bach.score's slot and pitch-bend data into corresponding note attributes and userBeans, see Figure 13).

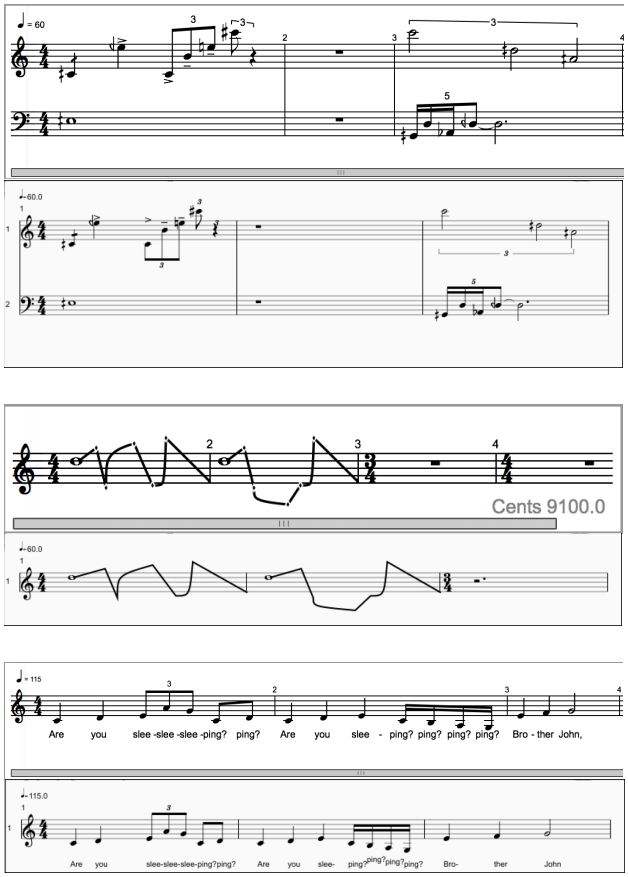

Figure 13. Translations between bach.score and MaxScore, featuring microtones (top), break-point functions (center) and text slots (bottom) performed by the maxscore.bachScoreToMaxScore abstraction.

4.2 Mira and MiraWeb

Mira and MiraWeb are technologies (the latter based on xebra.js) developed by Florian Demmer for Cycling '74, capable of mirroring Max GUI objects such as sliders, buttons, messages and comments in a dedicated iOS application and/or web browsers[3]. It therefore constitutes a perfect companion to Max by harnessing the multi-touch power of iPads, tablet computers or smartphones. The built-in *zeroconf* technology and automatic mirroring (Max objects simply need to be dragged onto an object called mira.frame) make Mira and MiraWeb a superior choice in comparison to alternative applications (e.g. TouchOSC, Lemur or C74).

[3] https://cycling74.com/articles/content-you-need-miraweb

Yet, dedicated notation objects mirroring objects such as nslider or bach.roll are currently out of reach as this would require a major development effort on the side of Cycling '74 or third-party developers. However, Mira and MiraWeb support the fpic object capable of dynamically loading and displaying images. We have taken advantage of this by creating an abstraction called MaxScore.toMira. In this scenario, MaxScore renders to a Jitter matrix object via the embedded jit.render2MaxScore abstraction (Figure 14). Upon exporting the matrix as an image to a temporary location, the fpic object is prompted to load and display this image after a short delay. This image is then transferred over the network to the Mira client. MaxScore.toMira performs adjusts automatically to score dimension and dynamically scales the multi-touch information it receives from the mira.touch object to support user interaction with a MaxScore object.

This approach is efficient enough to create the illusion of a dedicated notation object and thus offers the only seamless solution to date enabling users to interface with Max through music notation also supporting bach.score via the maxscore.bachScoreToMaxScore abstraction (see Figure 15.

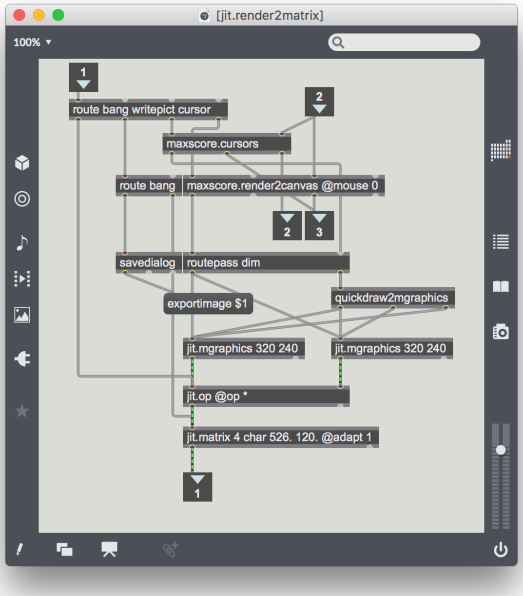

Figure 14. Rendering to Jitter enables MaxScore to save a score as an image.

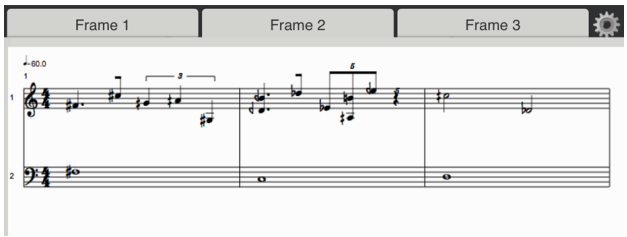

Figure 15. The MaxScore.toMira abstraction allows users to display scores on multi-touch devices. This example displays the content of a bach.score object via translation to MaxScore.

4.3 conTimbre playback

conTimbre is a sound bank of orchestral instruments created by Thomas Hummel. With more than 80000 individual samples—many of them performed with extended techniques—it is a tool becoming increasingly popular amongst new music composers and electronic musicians alike[4]. Using copy protection and a proprietary file format, it requires dedicated software to play back these samples. However, Hummel has implemented a suite of OSC message for interaction with its Max-based ePlayer. We have thus created new abstractions for Max and Max for Live exploiting the power of the conTimbre library and enabling multi-timbral microtonal playback. The MaxScore.2conTimbre module, complementing the MaxScore.Sampler and MaxScore.Fluidsynth2 playback devices, reads ePlayer settings files, which fills the menus with (and sets them to) the current instrument names.

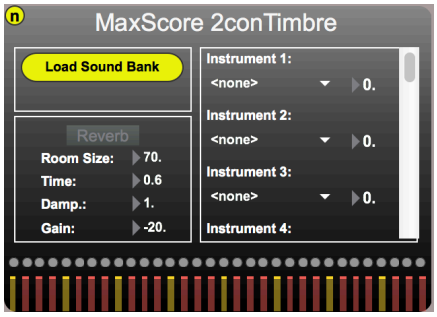

Figure 16. Screenshot of the MaxScore.2conTimbre module, a playback device for MaxScore.

5. NEW TOOLS FOR GUIDED IMPROVISATION AND SITUATED PERFORMANCES

5.1 Cursors

In 2006, Marlon Schumacher, then-member of the European Bridges Ensemble, asked Hajdu to implement cursors in the Quintet.net Client (a software for networked multimedia performance) for his piece *Fire* [14] which were to travel independently of each other at different speeds across its notation display to guide the performance of electronic musicians. In 2016, cursors were also added to the MaxScore Editor and more recently to MaxScore.toMira through the MaxScore.Cursors abstraction. The behavior of those cursors (a maximum of 20 per score) can be controlled with a variety of messages for which the Max @ attribute notation was adopted.

Each message contains the message name cursor, an instance number and any of the following @ attributes:

cursor 0 @begin 0 0 @end 0 1 @runs 1 @countdowncolor 1. 0. 0. 1. @countdown line @color 0. 0. 1. 0.7 @interval 2000 @ timestretch 2. @shape line

[4] https://www.contimbre.com

Depending on the @begin and @end attributes, the length of a cursor will be adjusted to the span of the specified staves. For this, the *getStaffBoundingInfo* query is performed to yield the bounding rectangle around a particular measure/staff (our term for the cross section of a measure and a staff, for lack of a better term). In addition, tempo and time signature are queried to determine the speed of the cursor travelling across the canvas.

Furthermore, cursors can also be controlled with the start, stop, resume, blink, unblink und hide messages. Instance number -1 can be used if all cursors are to be affected at once.

There is a difference between rendering cursors in MaxScore and Mira: While in MaxScore cursors are rendered just like any other graphics elements, they are represented by the actions of GUI objects in Mira (such as the visible line of a transparent multislider). This way, the network load can be decreased dramatically as only control messages need to be sent to the clients to adjust the position and size a multislider and move its line.

In 2016, one of us (Hajdu) participated in an academic exchange with Cat Hope, Lindsay Vickery and other members of the Decibel Ensemble (all at WAAPA, Edith Cowan University, Perth, at the time). The aim was to mutually expose ourselves to the developments of the other group [20]. A concert was organized at the end of our first stay. For this, the piece "Carnage" was written as a guided improvisation for the Decibel ensemble (flute, bass clarinet, viola, cello, percussion) and premiered on July 29, 2016. The piece (based on the eponymous film by Roman Polanski and Yasmina Reza) featured five lines of emoticons. The musicians were instructed to interpret the moods represented by the emoticons and were guided by the movements of the cursors. The notation was read from a single projection of the MaxScore canvas.

In November of 2017, this piece was performed again in Tel Aviv by the Meitar ensemble (featuring flute, bass clarinet, violin, cello and piano). During this performance, the musicians read the music from individual iPads running the Mira app (Figure 17).

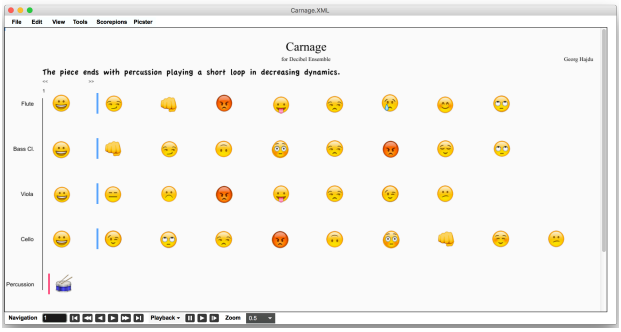

Figure 17. Hajdu's piece Carnage consists of a one-page score with individual cursors guiding the performance.

6. OUTLOOK AND CONCLUSIONS

MaxScore development is currently focusing on areas including network connectivity (MaxScore.NetCanvas and MaxScore.toMira), guided improvisation (MaxScore.Cursor) as well as compatibility with other software developments (bach and conTimbre). This has been facilitated by changes to the MaxScore object itself. A promising door has been opened by the introduction of Expressions which will allow users to pursue ideas akin to the Spatialization Symbolic Music Notation [16], which combines a language of icons with clearly defined spatial trajectories. For this, we will be working on a GUI accommodating a number of editors both in the graphical and control domains. It is also planned to use MaxScore and its peripheral components in a performance in the St. Pauli Elbtunnel in Hamburg—a 100-year old tunnel under the Elbe river, involving 144 musicians reading their music off portable devices in 2019. Until then, further strides will have to be done towards robustness and efficiency.

Acknowledgments

The authors would like to thank the Hamburg *Ministry for Science, Research and Equality* (BWFG) and the *Federal Ministry for Education and Research* (BMBF) for their support within their *Landesforschungsförderung* and *Innovative Hochschule* frameworks.

7. REFERENCES

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

[2] A. Agostini and D. Ghisi, "Bach: An Environment for Computer-Aided Composition in Max," in *Proceedings of the International Computer Music Conference*. Ljubljana, Slovenia, 2012, pp. 373–378.

[3] D. Fober, Y. Orlarey, S. Letz, "INScore – An Environment for the Design of Live Music Scores," in *Proceedings of the Linux Audio Conference*, CCRMA, Stanford University, 2012, pp. 47–54.

[4] Erich Gamma and Richard Helm, *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1994.

[5] D. Blostein and L. Haken, "Justification of Printed Music," *Communications of the ACM*, vol. 34, no. 3, pp. 88–99, 1991.

[6] G. Hajdu, "Dynamic Notation – A Solution to the Conundrum of Non-Standard Music Practice". *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR'15)*, Paris, France, 2015.

[7] C. Agon, G. Assayag and J. Bresson, *The OM Composer's Book Vol. 1*, Editions Delatour France / IRCAM, 2006.

[8] G. Hajdu, "Die Transkription des Poème Vers la flamme op. 72 von Alexander Skrjabin für Bohlen-Pierce-Ensemble," in A. Bense, M. Gieseking, B. Müßgens (Eds.), *Musik im Spektrum technologischer Entwicklungen und Neuer Medien. Festschrift für Bernd Enders*, Osnabrück: epOs, 2015, pp. 65-80.

[9] D. R. Tuohy and W. D. Potter, "Generating Guitar Tablature with LHF Notation Via DGA and ANN," in *Advances in Applied Artificial Intelligence. IEA/AIE 2006. Lecture Notes in Computer Science,* vol. 4031. Springer, pp. 244–253, 2006.

[10] N.-L. Müller, K. Orlandatou and G. Hajdu, "Starting Over – Chances Afforded by a New Scale," in M. Stahnke and S. Safari (Eds.), *1001 Microtones*, Neumünster: Von Bockel, 2014, pp. 127–172.

[11] G. Hajdu, "Resurrecting a Dinosaur – The Adaption of Clarence Barlow's Legacy Software AUTOBUSK," in *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR'16)*, Cambridge, UK, 2016.

[12] J. Sello, "The Hexenkessel: A Hybrid Musical Instrument for Multimedia Performances," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Brisbane, Australia, 2016.

[13] B. Carey and G. Hajdu, "Netscore: An image server/client package for transmitting notated music to browser and virtual reality interfaces," in *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR'16)*, Cambridge, UK, 2016.

[14] G. Hajdu and N. Didkovsky, "On the Evolution of Music Notation in Network Music Environments," *Contemporary Music Review,* vol. 28, nos. 4/5, 2009, pp. 395-407.

[15] 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 *Proceedings of the International Conference on Technologies for Music Notation and Representation (TENOR'17)*, A Coruña, Spain, 2017.

[16] E. Ellberger, G. Toro Pérez, J. Schuett, G. Zoia, and L. Cavaliero, "Spatialization Symbolic Music Notation at ICST," in *Proceedings of the joint International Computer Music Conference / Sound and Music Computing conference ICMC|SMC|2014*, Athens, Greece, 2014, pp. 1120-25.