Controlling complex virtual instruments – A setup with note~ for Max and prepared piano sound synthesis

Thomas Resch
Research and Development
University of Music Basel, FHNW
thomas.resch@fhnw.ch

Stefan Bilbao
Acoustics and Audio Group
University of Edinburgh
sbilbao@staffmail.ed.ac.uk

ABSTRACT
This paper describes a setup for embedding complex virtual instruments such as a physical model of the prepared piano sound synthesis in the sequencing library note~ for Max. Based on the requirements of contemporary music and media arts, note~ introduces computer-aided composition techniques and graphical user interfaces for sequencing and editing into the real time world of Max/MSP. A piano roll, a microtonal musical score and the capability to attach floating-point lists of (theoretically) arbitrary length to a single note-on event, enables artists to play, edit and record compound sound synthesis with the necessary precision.

Author Keywords
Computer-assisted composition, CAC, notation, rhythm tree, nested tuplets

ACM Classification
H.5.2 [Information Interfaces and Presentation] User Interfaces–Graphical user interfaces (GUI), D.2.6 [Software Engineering] Programming Environments–Graphical Environments, H.5.5 [Information Interfaces and Presentation] Sound and Music Computing.

1. INTRODUCTION
Containing only pitch, velocity and duration, a MIDI note-on event passes very limited information about the resulting sound to an instrument. In order to control more parameters, MIDI control or raw data has to be sent parallel to, or rather slightly in advance of, the corresponding note-on event. Microtonal pitches are often realized by detuning different channels of a MIDI instrument using pitch bend data, since most available instruments such as VST or Audio Unit plugins do not understand floating-point pitch information. These boundaries, introduced by the MIDI standard more than 30 years ago, not only disrupt the workflow of today's composers and sound artists, but often make it impossible to work with the necessary precision. By using more experimental environments such as Max/MSP, csound or Open Music, some of the MIDI standard's boundaries can be broken. But compared to commercial sequencer software they either lack the intuitive graphical user interface (GUI) or real time capabilities, or both.

The note~ library offers such an interface for Max/MSP. It includes the main timeline with an arrange view, a piano roll editor and even a musical score capable of displaying microtonal pitches and very complex rhythmical structures. The real time capabilities distinguish it from most other software tools for computer-aided composition.
The workflow described in this article is not limited to the prepared piano sound synthesis. It can be transferred easily to other instruments and therefore offers a new and inspiring way for controlling sound synthesis within Max/MSP. A brief introduction to note~ has been published by Resch in 2012 [1] but since then, the library has undergone a major revision. Among many other things, even the object names and basic functions have been modified. Therefore a new summary of the note~ library appears to be adequate although a detailed description outside the functionality used for the described setup goes beyond the scope of this article.

The prepared piano sound synthesis algorithm was originally developed by Bilbao. It was prototyped in Matlab and then ported to csound by John Fitch. The physical underpinnings, algorithm design, as well as the csound implementation have been explained in detail in their paper [2]. The current Max version – the object tr.ppiano – has been especially adjusted for the note~ library; its most significant features and implementation details will be described briefly, followed by the section about the integration of the prepared piano into the note~ environment.

For a better understanding of the two sections about implementation details, a basic knowledge about the Max/MSP software architecture is required, particularly regarding the difference between signal level and Max level. Such a distinction is described in detail at the Cycling'74 webpage [3].

NIME'16, July 11-15, 2016, Griffith University, Brisbane, Australia.

2. RELATED WORK
Several alternatives to note~ for the combination of a timeline with a graphical interface in Max/MSP exist, each focusing on slightly different areas. Their functionality varies accordingly. Max/MSP itself includes the object detonate which was originally developed by Miller Puckette [4]. It provides a 16-channel sequencer and a piano roll editor with the ability to assign up to six parameters to one event. It is easy to use but the functionality is limited. Agostini and Ghisi started the bach.project around the same time as note~ in 2010. It also offers a sequencer engine and a score for contemporary music [5]. It comes with a comprehensive library of objects for many types of data manipulation. The most significant difference with respect to note~ is the kind of data representation: note~ uses a completely linear approach while bach introduces a new data type called llll (Lisp Like Linked Lists). note~'s approach clearly has the advantage that its usage and workflow is very similar to common sequencer software; thus the training period is accordingly short. rs.delos by Roby Steinmetzer is another piano roll editor for Max [6]. Similar to note~ it can be synchronized to the internal Max timeline; it lacks a musical score. Max for Live provides an integration of Max/MSP into the Ableton Live environment but also does not offer a musical notation. The MaxScore project incorporates a musical score in Max/MSP and Max for Live [7]. It is based on the integration of the Java Music Specification Language (JMSL) into Max/MSP [8] and therefore requires a JMSL license; it lacks a piano roll editor.

While there is a lot of work on mapping strategies for playing instruments in a Max/MSP live setup, research into how to control, play back and preserve scores with sound synthesis beyond the scope of standard instrument plugins is rather sparse, at least for Max/MSP. One rare example is a paper about controlling CATART with the aforementioned bach.project [9]. For Open Music (OM) [10], the OMChroma framework provides functionality for this purpose [11].

But because of its LISP and PatchWork parentage, OM has a very different workflow compared to Max/MSP and common sequencers. Bresson published a paper about the integration of the complex CHANT Synthesizer in OM [12]. In collaboration with Agon they also wrote about symbolic control of sound synthesis in OM [13]. PWGL (Patchwork Graphical Language) is another visual programming language language already integrating a musical score and functionality for contemporary composing techniques [14]. The paper by Laurson, Norilo and Penttinen deals with a PWGL extension for real time control of sound synthesis [15]. As with Open Music, PWGL is based on PatchWork and LISP and therefore in many aspects similar to OM. Therefore a comparison with Max/MSP and the note~ library would not yield particularly useful insights.

3. THE NOTE~ LIBRARY

The library introduces a real time sequencer, a microtonal score, a piano roll editor and computer-aided composition techniques into the comfort zone of the Max/MSP environment.

All necessary objects and functions for the setup are described in this section. For a deeper insight, especially into note~'s scripting capabilities and the remaining objects, the reader is referred to the project website [16]. It provides the current version of note~ including tutorials, the tr.ppiano~ object and the patcher described in this article. note~ is currently available for OS X; a version for Windows is under development. The musical score requires the accidental font from Mathew Hinson which is available from his website [17].

3.1 The backend objects

3.1.1 note.seq~

The note.seq~ object represents the data backend and the playback engine of the note~ library. All of the library's other objects are useless without a note.seq~ object to connect to; its only (optional) argument is its name. It should be set to a unique identifier. Otherwise it is set automatically to *defaultnoteobject*. If the *embed* attribute is enabled, note.seq~ saves all data together with the patcher. Playback is controlled by sending "1" for start and "0" for stop to the object.

3.1.2 note.region

An object within note~'s timeline containing the event data is called a region. Regions can be generated by sending the message *newregion [regionname] [track] [timestamp] [duration]* to the note.seq~ object as shown below:

newregion myfirstregion 1 1. 16.

The object note.region is the Max representation of a note~ region. The first argument is the name of the note.seq~ mother object. The second (optional) argument is the name of an already existing region. Without the second argument, note.region connects automatically to the last created or touched region. The message *newevent [optional eventtype] [timestamp] [pitch] [duration] [optional additional parameters] [optional eventtext]* generates a new event as in the following example message:

newevent 0 1. 60. 1. 0.53 100. 0.25 "Some event text"

The *eventtype* can be viewed as the MIDI channel. The *duration* must be passed in MIDI format, where a "1." denotes a quarter note. In addition to *eventtype*, *timestamp*, *pitch* and *duration*, three more parameters plus text are generated with this message. Figure 1. shows a patcher containing the most simple note~ setup containing only the two backend objects and the messages explained above.

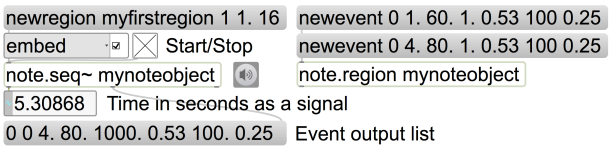

Figure 1. Minimal setup in Max/MSP

3.2 The note~ editor objects

3.2.1 note.regioneditor and note.eventeditor

The note.regioneditor provides the user with the main timeline and an interface for creating and editing regions.

The note.eventeditor enables creating and editing events within a region. It resembles the piano roll view of common sequencers. The most significant difference is the adjustable pitch resolution, which is not limited to semitones but can be set to any value by using the attribute *pitchresolution*.

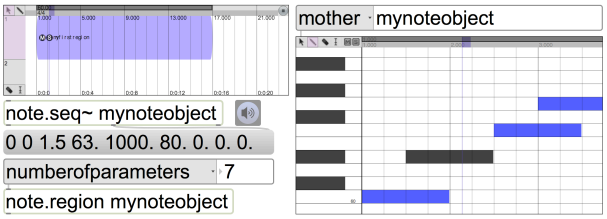

Figure 2. note.regioneditor (left) and note.eventeditor (right)

A minimal setup containing the two backend and the two editor objects is shown in Figure 2. Both regioneditor and eventeditor provide access to most functions through context menus.

3.3 The note.score

The note.score object is a musical score for standard western notation. Deviations from equal tempered pitches are displayed in cents. Nested tuplets up to six levels deep can be created by either choosing the *splitevent* option from the event context menu (which appears on right-click on a note) or by sending a *newevent* message to note.region with an extra rhythm-tree argument after the duration as shown below.

newevent 0 1. 60. 1. "3 (1 3 5)" 0.53 100. 0.25

This message generates several notes with a total length of a quarter note: the first number after the quotation mark divides the quarter note into three triplets. The second triplet is split again into three notes, the third into five. This rhythmical structure is shown in the first staff of Figure 3.

Since the score is merely a frontend, most rendering attributes must be set either in the note.seq~ or the note.region object. By default, redrawing occurs only on loading a patcher or on interactions with the other editors; it can be forced by sending a bang message to its inlet. Enabling the attribute *scoreredrawonmessage* forces the score to re-render also on messages. The user should be aware that this might freeze Max for some time if, for example, a very large number of events is generated algorithmically.

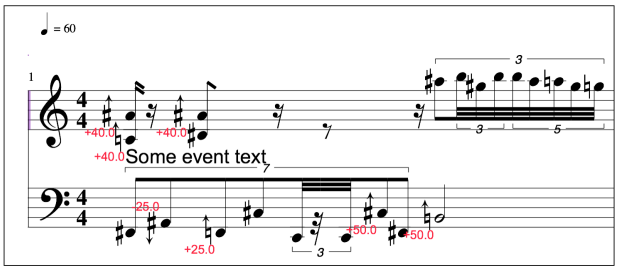

Figure 3. The note.score object

3.4 Implementation Details

The communication between note~ objects is realized with the observer design-pattern provided by the Max/MSP SDK. After setting the mother attribute of any note~ object to the name of an existing note.seq~ object, it attaches itself as an observer to the mother, the (main) publisher. In any kind of user interaction, the corresponding object sends a notification to the mother which itself then notifies all attached clients. Depending on the included message they can then determine whether they have to redraw or send data to an outlet. The signal processing for an MSP object is done within the perform method, called for every signal vector. However, Max/MSP does not permit sending Max messages on signal level to a non-signal outlet (or to be precise, Cycling'74 does not recommend it). Therefore note.seq~ simply calculates the progressing time on signal level. The sequencing is done on Max level with an extra clock running inside the Max main thread; this limits the maximal precision for playback to 1 ms. note.seq~ sends its clock to its left outlet for synchronizing purposes. It also accepts a signal as an external clock.

In memory, the data is represented as a linked list of regions, each containing a sorted linked list of events. While this is efficient for playback, it makes the rendering of the musical notation a difficult task. For lack of a better algorithm a lot of backtracking is necessary in order to detect the beginnings and endings of beams and tuplet brackets. To gain as much speed as possible, note~ formats the score only once on loading. From this point onward, only the altered sections are analyzed and reformatted. The data can be exported as a text file or saved internally together with the patcher. A note script is merely a list of note~ attribute settings, *newregion* and *newevent* function calls.

4. THE PREPARED PIANO

The prepared piano was ported to Max 4 in 2006 by Resch, but at that time the maximum number of simultaneously playable voices on an ordinary computer was somewhere between one and six for single precision, depending on the frequency of the strings (and of course on the CPU). Almost ten years later, a mid range laptop is fast enough to play this instrument in a polyphonic setup with 16 voices in double precision in real time.

4.1 Implementation Details

The implementation of string, preparations and hammer is more or less the same as described by Bilbao and Fitch [2]. In order to improve usability – finding parameters for string and preparations is already enough work – the number of preparations has been limited to two of each kind and every parameter is individually accessible through Max attributes. As an alternative to the hammer strike, a method for plucking has been added. It is realized by setting the string to a given velocity at the plucking position; the event duration has been taken into account through a simple (very fast) fade-out at the listening points after the given amount of time, or a note-off event.

The prepared piano is implemented as a finite difference model; hence the whole string must be calculated for each sample of every vector. Although this is much more CPU intensive than a waveguide model, it is a very flexible approach: Theoretically, any number of listening points, piano hammers and preparations can be attached and moved along the string during playback without causing any audible artifacts.

4.2 Usage

The first release of the prepared piano object for Max 4 was very flexible, but rather difficult to use. Attributes did not exist at that time for Max objects; parameters were set with messages containing long lists of floating-point numbers. Though the configuration is still possible with messages, the newly implemented attributes provide direct access to the values and are more or less self-explanatory: For example, *rattle1fundamentalfreq* sets the fundamental frequency of the first rattle preparation, the *hammerposition* attribute sets the position of the hammer.

On instantiation, two arguments need to be passed to the object: The number of listening points and the number of strings. The first – at least in most cases – will usually reflect the number of audio channels. The number-of-strings argument considers the fact that a piano has two (respectively three) strings for each note in the middle (respectively high) registers.

A simple list of floating-point or integer values triggers a hammer strike; it must contain at least pitch and velocity. The pitch can be passed either in Hz or as a MIDI floating-point pitch, depending on whether the attribute *pitchasmidi* is enabled. If the attribute *noteonwithduration* is enabled, the duration is expected as the third argument. While this makes no sense for live setups, it is a practical bonus for sequencing applications because sending separate note-off events is not necessary. The hammer position is expected as the fourth parameter, the detuning as the fifth, and the string decay as the sixth.

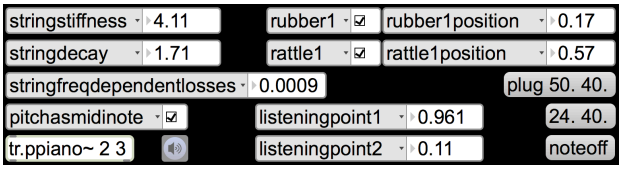

Figure 4. tr.ppiano in Max with attributes

If desired, all parameters can be set for every hammer strike by appending them to a note-on message list, either in the required order (which can be found in the tr.ppiano help file), or by prepending the name of the corresponding parameter. A note-on hammer strike message setting only pitch, velocity and string stiffness would look as in the following example:

62. 80. stringstiffness 4.1

5. EMBEDDING THE INSTRUMENT

The tr.ppiano~ is a purely monophonic instrument. Polyphony is achieved by putting the object inside a Max poly~ object which is capable of handling the voice allocation. With the aim of simulating a real piano accurately (and also saving some CPU resources), the Max patcher contains three poly~ objects, each of them holding a different instance of tr.ppiano~: One with seven voices and three strings for the descant, the bass with three voices and one string and the middle registers with five voices and two strings. Therefore the detuning parameter has no effect on the lower pitches. On playback, a simple subpatcher sorts the event lists by pitch and sends them to the corresponding poly~ object. The subpatcher inside poly~ merely measures the magnitude of the corresponding voice. If it falls under a certain threshhold, the instance is released and can be reallocated by the next incoming note. For all poly~ instances, parallel processing

and voice stealing should be enabled. If all voices are busy, the incoming note-on event steals the longest occupied.

The region that holds the events must be configured to include three additional values by setting the attribute *numberofparameters* to seven. When creating events in the editor, these are attached to every newly created note and initialized with zero. The default value can be changed with the note.region message *defaultparametervalue [parameter number/name] [default parameter value]*. Normally the eventeditor renders the pitch in a piano roll view. For the purpose of getting instant access to velocity, hammer position, detuning and string decay, at least one eventeditor should be configured with the attribute *parametersoverride* as shown in Figure 4. This changes the interface into a multislider-like view. For good order's sake also the *parametername* and the *vscaling* should be set for every parameter individually.

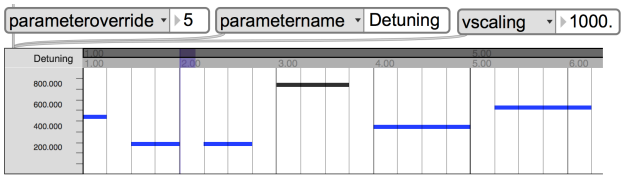

Figure 5. note.eventeditor in multislider mode

6. RESULTS

The proposed setup is very easy to handle; the patcher merely consists of objects of the note~ library, several attrui objects, the poly~ instances and a few messages.

The note.eventeditor provides an immediate visual feedback and makes it possible to access every value of every event. The musical score renders the data in common western music notation and denotes the deviation from equal tempered pitches in cents. These are necessary requirements for many composers, which are not integrated in Max/MSP. All data for the resulting sound of every note is bound to a single event instead of being subdivided into a note-on event and control data. In the case of rearranging events in time, either by using the interface objects or algorithmically, all data is moved together with every newly arranged note.

The prepared piano provides the user with a rich variety of sounds. The new release makes it very easy to explore these and use them afterwards with this setup. Altogether, 15 instances of the tr.ppiano object are created within the poly~ objects; the number of simulated strings is 34. The performance has been measured with two simple test cases: A worst-case scenario composed of a 15-voice cluster, which contains the (more or less) lowest frequencies possible for every poly~ instance and an algorithmically generated score. In both scenarios one rubber and one rattle is attached to all voices. On an Intel Core i7 with 3.1GHz, a vector size of 256 samples and a sample rate of 44.1 kHz, the CPU load for the cluster varies between 55% and 60%; the random score comes a little cheaper at 45% to 50% CPU load. The limited maximum accuracy of the event scheduler to 1 ms is not notable.

7. CONCLUSION

The large number of packages besides note~ (bach.project, rs.delos, MaxScore, Max for Live), which provide a timeline with a GUI for Max/MSP, proves that there is a definite need for this kind of application. note~ differs from these tools in that it enables access to CAC techniques while at the same time staying as close as possible to most well-know sequencers in its look and feel. The time it takes to learn new functionality is accordingly short.

The depicted Max patcher – built almost exclusively with objects from the note~ library – integrates the prepared piano sound synthesis perfectly. It enables the user to record, edit and play back the instrument in a very precise manner and does not even require very good skills in Max/MSP. Only three additional parameters are controlled additionally to the standard MIDI values: the hammer position, the detuning, and the decay. Nevertheless, this is already one parameter more than the internal Max object detonate could handle, and note~ would be able to control and change all 24 parameters of the tr.ppiano~ object individually for every single hammer strike. This accurate control of every value is a feature difficult to find in any comparable software.

Here, the arrangement is played with an especially adjusted instrument. But it can be considered as a proof of concept. Hundreds of third party instrument objects already exist for Max/MSP. Among them, all kinds of sound synthesis can be found. VST or Audio Unit instruments could also be used, even in a very similar manner. For this scenario, the subpatcher for the poly~ object would hold an audiounit~ object instead of the tr.ppiano~. The floating-point part of the pitch information could be extracted and sent to the corresponding poly~ instance before the note-on event as pitchbend data; additional parameters could be mapped to the corresponding MIDI controller values before the note-on event occurs.

Several projects have already been realized with note~, for example, Opium [18] by Philippe Olivier and the LEAP engine [19] by Duffield and Hopkins. The latter one is a good example of a non-audio system controlled by note~ – it is used as a laser pattern sequencer/level editor – and demonstrates that beyond its usefulness in music, note~ can be utilized for a broad range of applications.

8. ACKNOWLEDGMENTS

Thanks to Dr. Michael Kunkel and to the Electronic Studio Basel for their support. S. Bilbao is supported by the European Research Council, under grant number ERC-StG-2011-279068-NESS.

9. REFERENCES

[1] T. Resch (2013): "note~ for Max – An extension for media arts & music". In: *Proceedings of the international conference on new interfaces for musical expression, Daejeon, Republic of Korea, pp. 210-212*

[2] S. Bilbao, J Fitch (2006): "Prepared Piano Sound Synthesis". In: *Proceedings of the 9 th International Digital Audio Effects Conference, Montreal, Canada, pp. 77-82*

[3] Cycling'74 (n.y): "Max/MSP" [online]. URL: https://cycling74.com [accessed January 31st 2016]

[4] M. Puckette (1990): "EXPLODE: a user interface for sequencing and score following". In: *Proceedings of the International Computer Music Conference, Glasgow, Scotland, pp. 259-261*

[5] A. Agostini, D Ghisi (2012): "Bach: an environment for computer-aided composition in Max". In: *Proceedings of the International Computer Music Conference, Ljubljana, Slovenia, p. 373–378*

[6] R. Steinmetzer (n.y): "R. Steinmetzer" [online] URL: http://arts.lu/roby/index.php/site/maxmsp/rs_delos [accessed January 31st 2016]

[7] N. Didkovsky, G. Hajdu (2008): "MaxScore: music notation in Max/MSP". In: *Proceedings of the International Computer Music Conference, Belfast, N. Ireland*

[8] Didkovsky, N. and L. Crawford (2007): "Java Music Specification Language and Max/MSP". In: *Proceedings of the International Computer Music Conference, Copenhagen, Denmark, pp. 620-623*

[9] A. Einbond, C. Trapani, A. Agostini, D. Ghisi, D. Schwarz (2014): "Fine-tuned Control of Concatenative Synthesis with Catart Using the Bach Library for Max". In: *Proceedings of the International Computer Music Conference, Athens, Greece, pp. 1037-1042*

[10] J. Bresson, C. Agon, G. Assayag (2011): "OpenMusic: visual programming environment for music composition, analysis and research". In: *Proceedings of the 19th ACM international conference on Mulitmedia, New York, USA, pp. 743-746*

[11] C. Agon, J. Bresson, M. Stroppa (2011): "OMChroma: Compositional Control of Sound Synthesis". In: *Computer Music Journal, Summer 2011, Vol. 35, No. 2, pp. 67-83*, MIT Press Cambridge, MA, USA

[12] J. Bresson and M. Stroppa (2011): "The Control of the Chant Synthesizer in OpenMusic: Modelling Continuous Aspects in Sound Synthesis". In: *Proceedings of the International Computer Music Conference, Huddersfield, United Kingdom*

[13] J. J. Bresson, M. Stroppa, and C. Agon (2005), "Symbolic control of sound synthesis in computer assisted composition". In: *Proc. Int. Comp. Music Conf. (ICMC'05), Barcelona, Spain, pp. 303–306*

[14] M. Laurson, M. Kuuskankare, V. Norilo (2009): "An Overview of PWGL, a Visual Programming Environment for Music". In: "*Computer Music Journal, Spring 2009, Vol. 33, No. 2, pp. 19–31"*, MIT Press Cambridge, MA, USA

[15] M. Laurson, M. Kuuskankare, V. Norilo (2005): "PWGLSynth: A Visual Synthesis Language for Virual Instrument Dsign and Control". In: "*Computer Music Journal, September 2005, Vol. 29, No. 2, pp. 29–41, MIT Press Cambridge, MA, USA*

[16] T. Resch (n.y.): "note~ for Max" [online]. URL: http://www.noteformax.net [accessed January 31st 2016]

[17] M. Hindson (n.y.): "Mathew Hinson" [online]. URL: http://hindson.com.au/info/ [accessed January 31st 2016]

[18] Philippe Ollivier (2012): "Opium" [online]. URL: http://www.logellou.com/opium/ [accessed January 31st 2016]

[19] D. Hopkins, K. Duffield (2014): "Documentation of the leap engine" [online]. URL: http://hopkinsduffield.com/2015/02/13/documentation-of-the-leap-engine/ [accessed January 31st 2016]