# RESURRECTING A DINOSAUR - THE ADAPTATION OF CLARENCE BARLOW'S LEGACY SOFTWARE AUTOBUSK

**Georg Hajdu**
Center for Microtonal Music and Multimedia
Hamburg University of Music and Theater (HfMT)
Harvestehuder Weg 12
20148 Hamburg

georg.hajdu@hfmt-hamburg.de

## ABSTRACT

This paper aims at describing efforts to conserve and further develop the legacy real-time generative music program *AUTOBUSK* by Clarence Barlow. We present a case study demonstrating that a simple port of 30+ year old code may not suffice to infuse new life into a project that suffered from the abandonment of the hardware it was developed on/for. In the process of resurrecting this "dinosaur," *AUTOBUSK* was entirely redesigned for the popular music software environments Max and Ableton Live (via Max for Live) and renamed *DJster*. It comes in several incarnations, the most recent ones being *DJster Autobus* for Ableton Live, a device for real-time event generation and *DJster Autobus Scorepion*, a plugin for the MaxScore Editor. These incarnations take advantage of being embedded in current environments running on modern operating systems and have since acquired some new and useful features. As *AUTOBUSK/DJster* is based on universal musical principles, which Barlow formalized during the 1970's while working on his generative piano piece *Çoğluotobüsişletmesi*, its algorithms are of general applicability for composers and performers working in diverse fields such as microtonality, interactive installations and/or film music. It has therefore inspired me to lay the foundations of a shorthand notation, which we will discuss in the last section.

## 1. INTRODUCTION

*AUTOBUSK* is a real-time generative program developed by Clarence Barlow, which "took 272 days to write, spread between 18 August 1986 and 30 October 2000." [1] It is one of the first ones of its genre, which includes applications such as *M* by David Zicarelli, Joel Chadabe, John Offenhartz, and Antony Widoff (launched in 1987) [2], the *Lexikon Sonate* by Karlheinz Essl (development starting in 1992) [3], George Lewis' *Voyager* system (development starting in 1993) [4] as well as *Koan* (released by SSEYO in 1995), which Brian Eno used extensively for his generative pieces [5]. AUTOBUSK uses a probabilistic approach and is based on universal[1] musical principles which Barlow formalized while working on *Çoğluotobüsişletmesi* (completed in 1979). It was written in Pascal for the Atari ST computer platform, which became immensely popular amongst European musicians in the mid to late 1980's—mainly due to its low price and built-in MIDI ports. The first piece realized with this system was Barlow's performance art piece *Variazioni e un pianoforte meccanico* in which a performer plays the opening bars of the *Arietta* from Beethoven's piano sonata op. 111 on a Disklavier, with the performance gradually being taken over by *AUTOBUSK* controlling the piano via MIDI. The performer eventually leaves the piano to play by itself, only to return at the very end in order to conclude the piece. Another piece of his, *Pandora* (1989) took advantage of *AUTOBUSK's* ability to save generated scores as MIDI files. Cologne-based American pianist Kristi Becker took the 3-part Finale printout and arranged it for piano. Barlow composed a few more pieces on the Atari but was eventually forced to resort to a Windows emulator called Steem after Atari stopped building computers in 1993.



## 2. AUTOBUSK - A CASE STUDY FOR THE PRESERVATION OF A DIGITAL MEDIA ART WORK

The issue *AUTOBUSK* users are facing are typical for legacy software products. These are (1) decay of physical

[1] I always appreciated the fact that Barlow's algorithms aimed at capturing universal musical principles such as tonal and metric hierarchies, independent from personal style and bias. While the existence of universals in music is still under much debate, we are inclined to assume that distinct cultural efforts manifest themselves as distortions of universals (which would be much easier to formalize) rather than as unrelated particulars. Hence, using *AUTOBUSK/DJster* as a universal music machine justifies, the continued effort I have poured into further developing this environment and exposing my students to it, who repeatedly have raised, using someone else's brainchild to compose music, interesting questions about ownership and intellectual property.

media and/or (2) abandonment of media types, formats software, OS's and hardware.

Efforts to save digital works and thus to contribute to their longevity require pro-active preservation which comes in the form of migrating files to current media, using emulators to run old software or porting the code to more recent platforms. The most radical approach is to re-design the software, either by transcribing the algorithms or re-creating them entirely. Fortunately, in case of DJster I was able to fall back on several of Barlow's publications as well as his personal input. The most useful resource was the website maintained by the University of Mainz department of music informatics. It contains links to a zip archive with the Atari files and a 54-page user manual called "AUTOBUSK: A real-time pitch and rhythm generator" [1].
The *AUTOBUSK* GUI has several panes (Figure 1) of which the one on the left is most relevant. It features, for three individual "streams", a number of GUI items which serve to control the generative process. Its algorithms have been described by Barlow in his book *Bus Journey to Parametron* [6]. An excellent introduction to the underlying theory including an explanation of the terms *indispensability*, *indigestibility* and *harmonicity* is given in Barlow's textbook On Musiquantics [7].

The panes on the right pertain to real-time MIDI control as well as the creation and editing of parameter scores via additional "helper" programs.

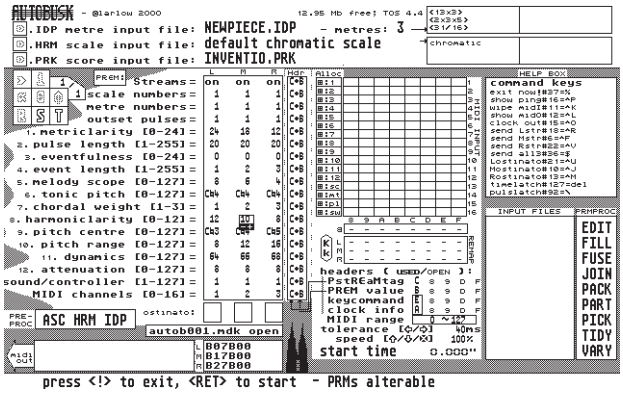


Figure 1: The GUI of Barlow's legacy program *AUTOBUSK*, written in Pascal for the Atari ST platform.

For its input, AUTOBUSK reads three different file types—some of which in binary format to save valuable hard-disk space (mind that a floppy disk in the early 1990's could only hold 1.4 MB). All three file types rely on the conversion of human-readable precursor files in text format:

- .mtr files, holding the stratifications of meters such as "2 2 3" (for a multiplicative meter with 12 pulses), to be converted into .idp files, spelling out the *indispensability* values for the given meters.
- .cts files holding the cent values for the chosen scales such as "0 200 400 500 700 900 1100 1200." Upon conversion into .hrm files, these values are expanded to the range from -9600 to 9600 cents, 0 cents being tonic pitch. In the process, for each scale step, various tuning alternatives are taken into consideration of which the one contributing to the best overall *harmonicity* (Barlow's term for tonal consonance) is chosen and prioritized according to its individual harmonicity value. This rationalization of the interval set is a lengthy process, as large numbers of combinations may need to be assessed.
- .prm files, to be converted into binary .prk files, consist of the temporal sequence of control messages such "5 900 L 9 40," denoting "set *pitch centre* for player 1 (left) to 40 at 5.9 seconds."

## 3. DJSTER - A RESURRECTION?

In 1992 while working as graduate student instructor at CNMAT/UC Berkeley under David Wessel, I oversaw a student project aiming at implementing the concept of *indispensability* in Max 2.0. The resulting abstraction was called *dispenser* and was used by me (and others) in various projects. It later added support for additive meters as well as user-defined arbitrary meters, which deliberately work against the grain of Barlow's "natural metric order" which conceptualizes syncopations in terms of low "metriclarity" or phase-shifted *outset pulses* (see Figure 1) in respect to a reference meter. It also became the basis for the first version of DJster, completed in early 2008. Its name is a reference to DJing as well as Barlow's notion of the *indigestibility* of a number, a concept central to his notion of tonality [8].

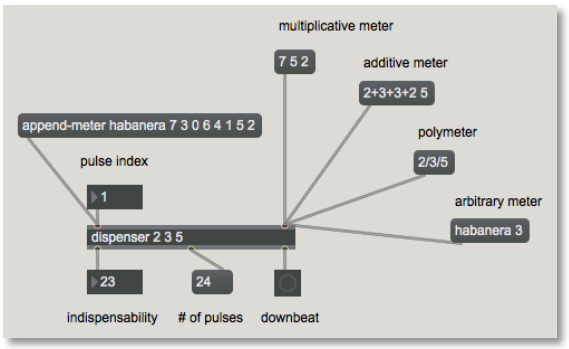


Figure 2: The Max *dispenser* abstraction accepts various meters in its right input. An arbitrary meter such as Habanera requires the meter to first be appended to dispenser's internal table with a message consisting of the message name *append-meter*, a symbolic value and the corresponding list of *indipensabilities*.

### 3.1 The Original Max Implementation

DJster was programmed in MaxMSP 4.6, preserving as much as possible the original layout of the left pane, while forgoing the implementation of MIDI control and helper applications (Figure 3). Instead, all parameters are

exposed via the Max *pattr* system and controllable via OSC messages. For backwards compatibility it can also read and play *AUTOBUSK* input files. Translation of .cts into .hrm files is achieved on the fly by table look-up of "harmonic energy values," thus no longer requiring the time-consuming rationalization of the scales' interval set.

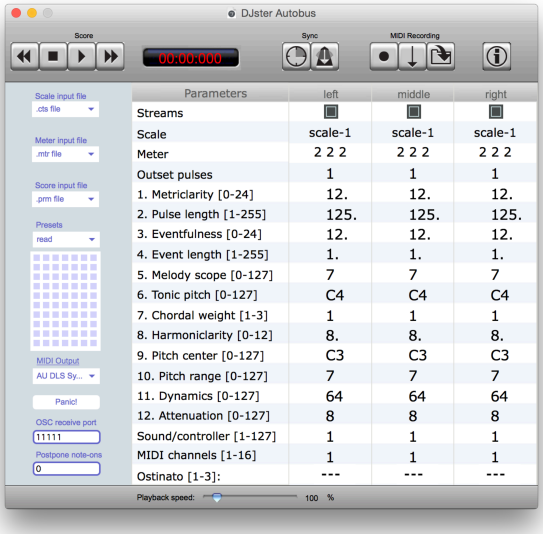


Figure 3 shows the GUI of the 2008 version of *DJster*, based on the generative music part of *AUTOBUSK* and implemented in *MaxMSP*.

### 3.2 Version for Quintet.net

A second Max implementation exists for the networked multimedia performance environment *Quintet.net* (Figure 4) [9]. It separates the streams into individual instances and is capable of dealing with microtonality. Scale files use key and value pairs, of which the key (a symbol such *Pentatonic*) will appears in the scale menu (instead of just an index). Since *indispensability* values are (as with *harmonicity*) calculated on the fly, the conversion of .mtr to .idp files is no longer necessary either. *Pulse length* and *event length* are no longer limited to the original ranges and all parameters dealing with pitch use cent resolution for display and microtonal playback. Real-time control can be exerted by sending OSC messages between various *Quintet.net* components. In my classes, it has been used as the target of Brain-Computer Music Interfaces or gestural interfaces such as the LeapMotion (via a few-to-many mapping performed by an artificial neural network).

### 3.3 Ableton Live Device

The third incarnation represents the biggest leap (Figure 5): By creating a Ableton Live device via the Max for Live API, DJster needed to be adapted to the philosophy of its host, a DAW driven by beat and loop-oriented *electronica*. Pulse length and meter had to be reconsidered and reworked into two parameters owned by the host (*song tempo* and *time signature*) in addition to *subdivision* of the beat—settable in DJster device.

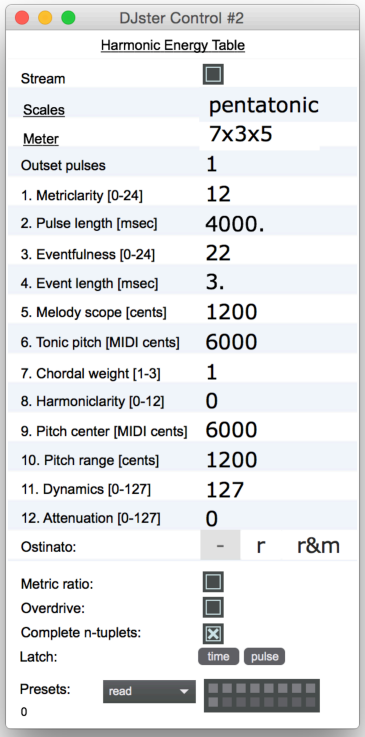


Figure 4: The GUI of the *Quintet.net* version of *DJster*. It contains a few additional parameters on the bottom added for the performance of *Just Her – Jester – Gesture.*

Internally, in the DJster Autobus device, *pulse length* is thus re-calculated by dividing 240000 by the product of tempo, time signature denominator and the number of pulses subdividing a beat:

$$Pulse\_length[m\sec] = \frac{240000}{tempo \cdot ts\_denom \cdot num\_psub} \qquad (1)$$

In turn, *meter* is derived by concatenating the prime factorization of the time signature numerator with the *stratification divisors* of the subdivision of the beat. E.g. for a 12/4 time signature with a quintuple subdivision of the beat we get 2x2x3x5 as meter.

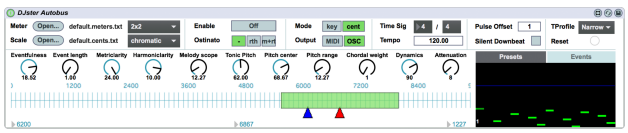


Figure 5: The GUI for the *Ableton Live* incarnation of *DJster*.

As all of DJster's parameters are automatable, .prm/.prk score files are no longer supported. Interpolations between parameter values will now be performed automatically by the host. Ableton's arrangement view allows continuous tempo changes as well as time signature changes from one measure to another. The repertoire of scales can easily be expanded simply by dropping files

from Manuel op de Coul's Scala archive [10] onto the Scale menu.

### 3.4 MaxScore Plugin

The last incarnation of DJster is a MaxScore Editor Scorepion plugin, which shares Ableton's tempo/time signature/subdivision concept and is able to exchange presets with the latter (Figure 7). MaxScore is a Max Java notation object programmed by Nick Didkovsky [11]. A visual editor for it was created by the author. It also possesses a plugin structure called Scorepion, which extends MaxScore core functionality through the use of Max patches invoked from a particular folder in the MaxScore folder hierarchy. The DJster Autobus Scorepion fills selected measures of a MaxScore staff with notes (Figure 6) and—with its non-real-time approach—brings back to life *AUTOBUSK's* ability to write its output to a MIDI file readable by Finale, Sibelius and co. In DJster, though, this intermediary step is no longer necessary as this all happens in one environment. Therefore, work is more intuitive and combines the manual approach of traditional composition with the generative approach of computer composers who prefer to tweak code rather than music[2].

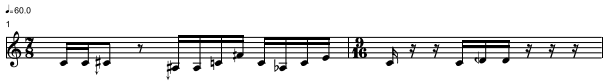

Figure 6: Two measures of music in Carlos Alpha tuning created after adding a scale file from the Scala archive to Djster's scale repertoire. See Figure 7 for the settings used in this example.

In order to fill the selected measures in MaxScore with music, the user first requests information about their time signature and tempo attributes by clicking on the "Gather Info" button (Figure 7).

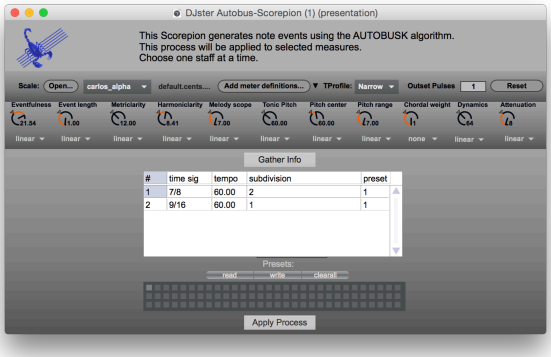


Figure 7: The GUI of the *DJster Autobus Scorepion* plugin for the *MaxScore* Editor. Note the three bottom rows of presets, allowing users to exchange settings with the Ableton Live device.

After creating any number of presets consisting of DJster parameters and interpolation modes, he/she can use the info pane to change the default subdivisions manually and assign preset numbers to each measure. Depending on the interpolation mode, parameter changes between measures will be either abrupt or gradual. Upon clicking on "Apply Process" the generative process will be triggered and the resulting events transcribed into notation.

## 4. EXAMPLES OF RECENT WORKS

After using the *dispenser* abstraction for real-time composition in the *Intermezzo* of my opera *Der Sprung – Beschreibung einer Oper* (1996) [12], I have employed DJster in the interactive composition *Just Her – Jester – Gesture* and focused on the non-realtime application of the Scorepion in my works *In ein anderes Blau* (2012) (Figure 8) and *No, I won't* (2014) [13]. *Dispenser* was used for real-time composition/notation in the percussion and multimedia piece *Slices* by Jacob Sello [14] while the Ableton Live version of DJster was employed in the real-time interactive dance performance *Mond in Wogen* by Xiao Fu [15].

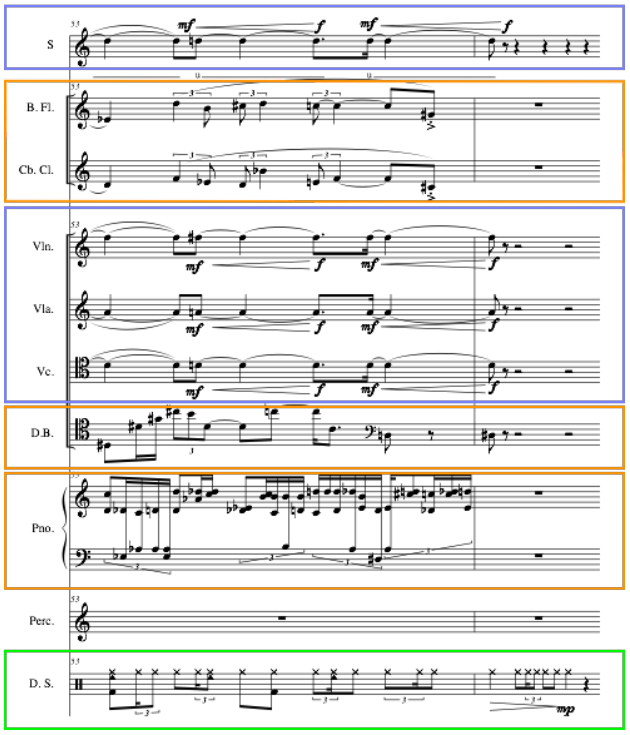


Figure 8: A page from Georg Hajdu's composition *In ein anderes Blau* in which the transcription of improvised music (blue frame) is combined with music generated with the DJster Scorepion. For this, meter and scale definitions were created to match the pitch set of the improvisation as well as the underlying 2/3/5 polymeter.

[2] I have elaborated this relationship in a presentation given at the 2012 Ligeti symposium in which I compared György Ligeti's and Clarence Barlow's approaches to generative music (http://quintetnet.hfmt-hamburg.de/Ligeti-Symposium/?page_id=90)—the former composer characterized by his non-computer PPP approach (paper, pencil, pocket-calculator).

## 5. TOWARDS A NOTATION OF DJSTER

In *Just Her – Jester – Gesture* MaxScore sends out, in sync with the performer, messages to several instances of the Quintet.net version of DJster. The notation consists of single notes to which lists of parameter values have been attached via the MaxScore Editor note-slot feature (Figure 10). Since there is no way of guessing those values from the appearance of a note, a specific DJster shorthand notation could be handy while serving two purposes: Firstly, it could symbolically represent the parameter constellations to be sent as sequenced message to the real-time version of DJster, or, secondly, the notation could serve as a control track for a non-real-time composition in which the DJster Scorepion actually spells out the notes, which bears some resembles to figured bass.

Figure 9 shows a mockup of this shorthand notation. It consists of a regular note (denoting *tonic pitch*) and two smaller notes denoting *pitch center* (diamond) and *melody scope* (in terms of the interval between the diamond and the circle). On top of the note, there is a slider box with five sliders and a symbol referring to the scale currently in use. Table 1 delineates the relationship between DJster parameters and their symbolic representation in shorthand notation.

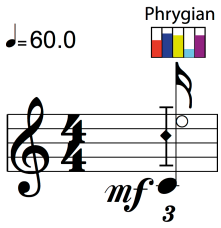


Figure 9. DJster shorthand notation. Refer to Table 1 for a detailed explanation of the parameters involved.

| Parameter | Representation |
|---|---|
| Scale | String above slider box |
| Meter/Subdivision | Ratio between Denominator and note value (MaxScore duration property) |
| Eventfulness | 1. element of slider box |
| Event Length | MaxScore hold property |
| Metriclarity | 2. element of slider box |
| Harmoniclarity | 3. element of slider box |
| Melody Scope | Interval between pitch center and round notehead |
| Tonic Pitch | Regular notehead (MaxScore pitch property) |
| Pitch Center | Diamond notehead |
| Pitch Range | Brackets extending from pitch center |
| Chordal weight | 4. element of slider box |
| Dynamics | Dynamics symbol (MaxScore amplitude property) |
| Attenuation | 5. element of slider box |

Table 1: List of parameters represented by DJster's shorthand notation.

The DJster notation editor will be implemented as a MaxScore slot-editor module (Figure 10). Upon clicking on Slot in the Note Attributes palette, the pitch, duration, amplitude, hold and tuplet attributes of the selected note will set the corresponding DJster parameters as defaults. The other 9 parameters will then have to be specifically set in the GUI. Once all values are set, MaxScore generates two types of data:

- A list of data to be output by the playback engine
- A *Picster* graphic [11] to be embedded in the score

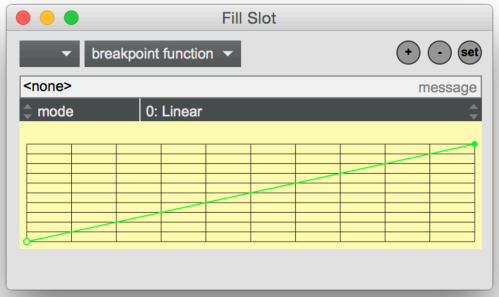


Figure 10. Example for a MaxScore Editor slot module. The DJster slot module will inherit its controls from the DJster Scorepion.

## 6. CONCLUSIONS

In this paper I gave an account of how a legacy computer music program can be revived by adapting its algorithms to modern environments. Similar attempts have successfully been accomplished by David Zicarelli who in 1997 resumed maintaining his "Intelligent Composing and Performing System" *M* and most recently by Gottfried Michael Koenig whose *Projekt 1* from 1964 was just recently translated into SuperCollider by Rainer Wehinger. In case of DJster we went through a evolutionary process leading to several versions of the original system. The last one, a non-real-time plugin for MaxScore, allows users to intuitively combine traditional and generative approaches to music composition. As the DJster project has implications that touch on the unabating issues of real-time music generation, microtonality, man-machine interaction as well as symbolic data representation and mapping, development will continue with a strong focus on documentation, user friendliness and flexibility. Currently, its usability for music generation within hospital environments is being studied within the Healing Environment project, jointly organized between two department at the HfMT Hamburg departments and the university hospital Hamburg-Eppendorf (UKE).

DJster in its various incarnations can be downloaded from http://djster.georghajdu.de.

**Acknowledgment**

I would like to thank the *Behörde für Forschung und Wissenschaft Hamburg* for supporting our research in the framework of its *Landesforschungsförderung*.

## 7. REFERENCES


[1] Clarence Barlow's AUTOBUSK user manual "AUTOBUSK: A real-time pitch and rhythm generator" for the Atari ST version is available as report no. 44 (2001) in the series "Musikinformatik & Medientechnik" or as download from http://www.musikwissenschaft.uni-mainz.de/Autobusk/.

[2] D.Zicarelli, "M and Jam Factory," *Computer Music Journal*, vol. 11, no. 4, pp. 13-29, 1987.

[3] K. Essl: "Lexikon-Sonate. An Interactive Realtime Composition for Computer-Controlled Piano" *Proceedings of the II. Brazilian Symposium on Computer Music,* 1995. Also available online at http://www.essl.at/works/Lexikon-Sonate.html.

[4] G. E. Lewis, "Too Many Notes: Computers, Complexity and Culture in Voyager," *Leonardo Music Journal,* vol. 10, pp. 33-39, 2000.

[5] N. Collins, "The Analysis of Generative Music Programs," *Organised Sound,* vol. 13 , no. 3, pp. 237 248, 2008.

[6] C. Barlow, "Bus journey to parametron," *Feedback Papers*, vol. 21–23, 1980.

[7] C. Barlow, *On Musiquantics – English translation of Von der Musiquantenlehre (2008). Issue 51 of Musikinformatik & Medientechnik*, University of Mainz, 2012.

[8] C. Barlow, "Two Essays on Theory," *Computer Music Journal,* vol. 11, no. 1, pp. 44-60, 1987.

[9] G. Hajdu, "Quintet.net: An Environment for Composing and Performing Music on the Internet," *LEONARDO,* vol. 38, no. 1, pp. 23-30, 2005.

[10] http://www.huygens-fokker.org/scala/

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

[12] G. Hajdu, "Research and Technology in the Opera Der Sprung". In *Nova Acta Leopoldina*, vol. 92, no. 341, pp. 63-89, 2005.

[13] http://georghajdu.de/compositions/

[14] http://zkm.de/media/video/jacob-sello-slices

[15] http://quintetnet.hfmt-hamburg.de/fuxiao/?page_id=206


Webpages all accessed on November 15, 2015.