PWSYNTH: A LISP-BASED BRIDGE BETWEEN COMPUTER ASSISTED COMPOSITION AND SOUND SYNTHESIS

BY MIKAEL LAURSON AND MIKA KUUSKANKARE

Center of Music and Technology, Sibelius Academy, Helsinki, Finland

laurson@siba.fi, Mika.Kuuskankare@siba.fi

Abstract

This paper introduces a new Lisp based software synthesizer called PWSynth. In simple cases PWSynth can be used like other visual synthesis languages with the help graphical patches. The main aim with this project is, however, to integrate sound synthesis with a novel music notation package. This scheme allows to control sound synthesis interactively with a familiar representation of complex musical information. The system has also proven to be useful when designing different control strategies for physical models of acoustical instruments.

Introduction and Background

Computer music software has been divided in the past roughly in two main categories: computer assisted composition and sound synthesis. The former category (PatchWork, Laurson 1996; Open Music, Assayag et al. 1999) consists of systems that are typically built on top of a high level programming languages – for instance Lisp - and are meant to be used as non-real-time environments. The main emphasis has been in representing and generating musical material for instrumental music. The latter category consists of real-time sound synthesis environments (e.g. SuperCollider2, McCartney 1998) where performance is of primary importance. This has led to systems that do not include extensive tools for representing complex musical entities.

During recent years the distinction between these two approaches has become more obscure. On the one hand SuperCollider2 has built-in tools that allow the user to specify compositional structures. On the other hand PatchWork contains user-libraries – such as PWCollider (Laurson 1999c) - which are meant to serve as bridges between non-real-time compositional work and real-time environments. Although the latter approach has made it possible to work in both domains, the idea of interfacing two systems with different logic and syntax causes several problems.

PWSynth is a PatchWork user-library that aims at a better integration of computer assisted composition and sound synthesis. PWSynth is a part of a project which investigates different control strategies for physical models of acoustical instruments. PatchWork is used to generate control data from an extended score representation system called Expressive Notation Package or ENP (Laurson 1999b; Kuuskankare and Laurson 2000). The actual synthesis was meant to be realized with a synthesis environment outside PatchWork (either with MSP or SuperCollider2). This approach, however, turned out not to be fruitful. It became obvious that we need better integration of the high-level aspects and the DSP parts of the system.

The rest of this paper is divided in four main sections. First we discuss in general terms ENP and describe how it can be used for sound synthesis control. After this we go over to PWSynth starting with the basic structure of the system and extending it to cover more complex cases that are needed when designing model-based instruments. The next section gives some ideas how ENP and PWSynth are interfaced and how ENP is used to

control the synthesis engine. We end with a concluding section and give some ideas for future work.

Synthesis Control using Expressive Notation Package (ENP)

ENP is a PatchWork user-library written in Common Lisp and CLOS. It can be further extended and customized by the user. It has a graphical interface and musical objects and their properties are editable with the mouse or with specialized editors. The system provides full access to the musical structures behind the notation. Thus ENP can be controlled algorithmically. Furthermore, ENP provides both standard and user definable expressions. The user can create new expressions using inheritance. Note-heads, stem-lengths and slur-shapes can be set either locally or by using global preferences. Scores can be saved as MIDI-, ENIGMA- or ENP-files or they can be exported and printed as PostScript.

ENP has already been used to solve constraint-based problems and to produce control information for several model-based instruments (Laurson 1999a, Laurson et al. 1999b, Laurson 2000, and Välimäki et al. 2000). The main purpose of ENP is to provide in the same package both professional notation capabilities and powerful object structures for compositional use.

In synthesis control the user enters in ENP the musical piece or phrase in standard notation. The system requires no textual input and includes facilities for fine tuning of the timing information. The user can also add both standard and non-standard expressions that allow to specify instrument specific playing styles with great precision. Expressions can be applied to a single note (such as string number, pluck position, vibrato, or dynamics) or to a group of notes (left-hand slurs, finger-pedals). Groups can overlap and they may contain other objects, such as breakpoint functions. The latter case is called a group-BPF. Macro expressions generate additional note events (tremolo, trills, portamento). After the input is finished the score is translated into control information. During this process the expressions trigger instrument specific rules that generate appropriate control data. Figures 1 (from “Madroños” by Federico Moreno Torroba) and 2 (from “Lettera Amorosa” by Juan Antonio Muro) give two short musical excerpts containing both standard and non-standard expressions.

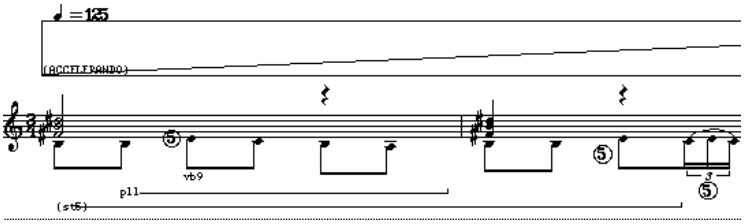

Figure 1. ***A musical excerpt from the standard classical guitar repertoire.***

Figure 2. ***A musical texture with some modern notational conventions.***

ENP contains an instrumental database consisting of instruments typically used in the modern western orchestra. The system can be easily extended to include also purely synthetic instruments. The root class is an abstract class - called ‘instrument’ - that has slots for general information such range, clef and transposition. The ‘instrument’ class has subclasses such as ‘strings’, ‘wind-instruments’, ‘percussion’, ‘keyboard-instruments’,

'human-voice' and 'synth-instrument'. The 'strings' class, in turn, has subclasses for 'bowed-strings' and 'plucked-strings', the 'plucked-strings' class has subclasses such as 'guitar', 'mandolin', and so on. This class hierarchy allows to add specialized methods that define how various instruments should react to note information and score expressions during synthesis.

ENP is described in more detail in Laurson et al. 1999b, Kuuskankare and Laurson 2000, and Laurson 2000.

PWSynth

The starting point in PWSynth is a patch consisting of boxes and connections. This patch is as any other PatchWork patch except that each PWSynth box contains a private C-structure. The C-structures are visible both to the Lisp environment and to a collection of C subroutines which are interfaced to the Lisp system. The Lisp environment is responsible for converting a PWSynth patch to a tree of C-structures. Also it fills and initializes all needed structure slots with appropriate data. Once the patch has been initialized PWSynth calls the main synthesis C routine which in turn starts to evaluate the C-structure tree. Real-time sliders, MIDI devices and mouse coordinates can be used to control the patch. These controllers can be scaled to the desired range either using linear or exponential scaling.

In simple cases PWSynth resembles many other visual synthesis languages where the user works with basic entities such as oscillators, noise generators, filters, delays, envelopes and other modules typically used in software synthesis. Figure 3 gives a simple PWSynth example containing a 'sinosc' module with two inputs: 'freq' and 'amp'. These inputs are connected to two slider modules that can be modified by the user in real-time. The output of the 'sinosc' module is connected to the first input of a 'synth' box that in turn is responsible in converting the synthesis patch to a C-structure tree. Thus every PWSynth patch must contain at least one 'synth' box. The second input gives the current synthesis mode. There are three modes. In our case the mode is 'RT' (or real-time) which means that the output is sent directly to the DACs. If the mode is 'File' then the system runs in a non-real-time mode and the output is written to an AIFF file. In the 'List' mode the 'synth' box returns the output as a list of floating-point numbers. The latter case is especially useful when the user wants to inspect the output either as numerical values or plot them in PatchWork in a special 'points' view box (see below Figure 4). Figure 3 shows also the 'Synth RT Status' window that is always shown when the system runs in real-time. It contains some general synthesis information such as the number of channels, the number of the C-structures used by the current patch, etc.

Figure 3: ***A simple PWSynth patch.***

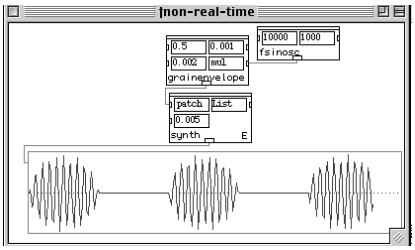

Figure 4: ***A 'List' mode patch with a 'points' view.***

A slightly more complex patch is given in the next example (Figure 5). It contains besides several 'sinosc', 'smooth', '+', and '*' modules also two real-time modules – 'mousex' and 'mousey' - that allow to control the synthesis according to the current mouse position within the main window.

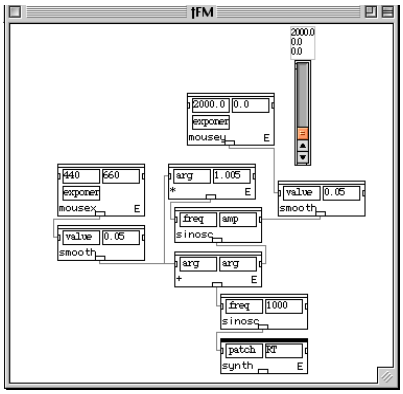

Figure 5: ***A PWSynth patch containing real-time mouse control.***

Figure 6, in turn, gives a fairly complex patch where there are two banks of resonators (band-pass filters). The interesting point here is to note that both banks - see the boxes with the name 'resonbank' – accept Lisp lists as the third input. In our case the list consists of three sublists – amplitudes, frequencies and bandwidths - that define the behaviour of the resonators. This scheme is very powerful as it allows the user to pass PatchWork data to the synthesizer in a flexible and dynamic way. Thus, for instance the number of filters can be modified simply by changing the size of the sublists of the main-list.

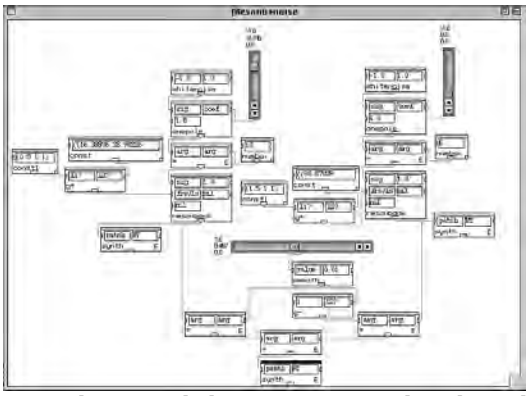

Figure 6: ***A PWSynth patch containing resonator banks with list inputs.***

PWSynth supports also imbedded definitions of C-structures (a C-structure can contain other C-structures to any depth). This feature is of primary importance when designing complex instrument models. For instance, a commuted waveguide guitar model (Smith 1993; Karjalainen et al. 1993) - consisting of a coupling matrix and six dual-polarization string models - contains a C-structure defining the complete instrument. This top-level C-structure contains seven sub-structures, one for the sympathetic couplings and six for

the strings. The most complex entity used in this example is the dual-polarization string model. It is a C-structure built out of nine sub-structures: two delay lines with 3rd order Lagrange interpolation, two loop-filters, three sample players, a delay and a timbre filter. The guitar model will be discussed also in the next section.

4. Interface between ENP and PWSynth

Figure 7 gives a typical example of how ENP is interfaced with the synthesis part of the system. On the left we have a window containing two patches: the first one, called '*SYNTH*', defines the synthesis part whereas the second one—'*ENP*'—is responsible for the calculation of the control data. The input ENP score is shown in the background behind the main window.

The synthesis part defines—using a box called 'guitar1'—a guitar synthesizer. The 'guitar1' box has three inputs. The first input defines the overall amplitude of the output. The second input is connected to a matrix-object, which contains a 6×6 matrix defining the sympathetic coupling of the strings. The third input is connected to a 6×18 parameter matrix where each row gives the parameters required for a single string. The contents of the parameter matrix is shown below in Figure 8. Thus our guitar synthesizer consists of a coupling matrix and 6 strings each having 18 parameters. The output of the 'guitar1' box is connected to a 'synth' box.

The control information is calculated with a box called 'enp→synth' having three inputs. The first one is an ENP input score. The second input is used for ENP performance rules (for more details see Laurson et al. 1999b). The third input defines the output mode for the calculated control information.

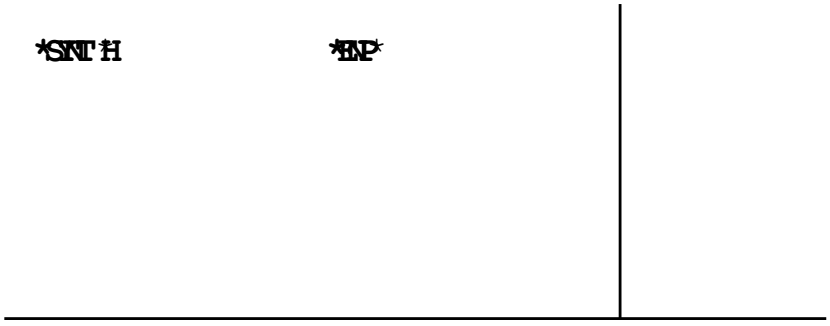

Figure 7: ***Overview of the synthesis and control part of the system.***

The interaction between ENP and PWSynth is realized in complex cases – such as the guitar model example given in Figure 7 - with the help of graphical matrixes (Figure 8). Each row and column are named. By taking the intersection of the row and column names a large symbolic parameter space is easily generated. Each resulting parameter name (pointing to an address in a C-structure) is a pathname similar to the ones found in other synthesis protocol schemes such as OSC (Wright and Freed 1997). Thus, if we assume that our current guitar synthesizer has the name 'guitar1', we can for instance refer to the loop filter coefficient ('lfcoef') of the second string ('2') of our guitar model with the pathname 'guitar1/2/lfcoef'. This naming scheme is powerful because it allows the simultaneous use of many instrument instances with separate parameter paths. For instance we can easily add new instrument boxes (such as 'guitar2', 'guitar3', etc.) each having a unique parameter name space.

Figure 8: ***Graphical parameter name matrix.***

5. Conclusions and Future Work

We have discussed a new visual software synthesizer called PWSynth. PWSynth has been implemented as a PatchWork user-library and thus allows to integrate computer assisted composition and sound synthesis in a novel and interesting way. Also it has been proven to be a useful research tool when designing control strategies for complex model-based instruments.

Future work will include the extension and refinement of the current model-based instrument library. Interesting modeling and control challenges can be found in the woodwind-instrument family, like flute and clarinet (both with fingerholes) and in the bowed-string instrument family such as cello and violin.

6. Acknowledgements

This research has been conducted within the project “Sounding Score—Modeling of Musical Instruments, Virtual Musical Instruments and their Control” financed by the Academy of Finland. The authors are grateful to V. Välimäki, C. Erkut and M. Karjalainen from the Lab. of Acoustics and Audio Signal Processing of Helsinki University of Technology for their help and expertise during this project.

7. References

Assayag, G., C. Rueda, M. Laurson, C. Agon and O. Delerue. 1999. “Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic” *Computer Music Journal*, 23(3): pp. 59-72.

Karjalainen, M., V. Välimäki, and Z. Jánosy. 1993. “Towards high-quality sound synthesis of the guitar and string instruments.” *Proceedings of the 1993 International Computer Music,* pp. 56–63.

Kuuskankare, M., and M. Laurson. 2000. “Expressive Notation Package (ENP), a tool for creating complex musical output.” in *Proc. Les Journées d'Informatique Musicale*. Bordeaux, pp. 49–56.

Laurson, M. 1996. *PATCHWORK: A Visual Programming Language and Some Musical Applications.* Doctoral dissertation, Helsinki, Finland: Sibelius Academy.

Laurson, M. 1999a. "Recent Developments in PatchWork: PWConstraints - a Rule Based Approach to Complex Musical Problems." To be published in *Symposium on Systems Research in the Arts 1999 - Volume 1: Systems Research in the Arts - Musicology*, IIAS*,* Baden-Baden.

Laurson, M., J. Hiipakka, C. Erkut, M. Karjalainen, V. Välimäki, and M. Kuuskankare. 1999b. “From expressive notation to model-based sound synthesis: a case study of the acoustic guitar.” *Proceedings of the 1999 International Computer Music Conference*, pp. 1–4.

Laurson, M. 1999c. “PWCollider.” *Proceedings of the 1999 International Computer Music Conference*, pp. 20-23.

Laurson, M. 2000. “Real-time implementation and control of a classical guitar synthesizer in SuperCollider.” *Proceedings of the 2000 International Computer Music,* pp. 74–77.

McCartney, J. 1998. “Continued Evolution of the SuperCollider Real Time Environment.” *Proceedings of the 1998 International Computer Music Conference*, pp. 133-136.

Smith, J. O. 1993. “Efficient synthesis of stringed musical instruments.” *Proceedings of the 1993 International Computer Music Conference*, pp. 64–71.

Välimäki V., M. Laurson, C. Erkut, and T. Tolonen. “Model-Based Synthesis of the Clavichord.” *Proceedings of the 2000 International Computer Music Conference*, pp. 50-53.

Wright, M., and A. Freed. 1997. “Open Sound Control: a new protocol for communicating with sound synthesizers.” *Proceedings of the 1997 International Computer Music Conference*, pp. 101–104 "