PWCollider - A Visual Composition Tool for Software Synthesis

Mikael Laurson
Sibelius Academy
Centre for Music Technology
P.O.Box 86
00251 Helsinki, Finland
laurson@amadeus.siba.fi

Abstract

This paper presents a visual interface between PatchWork (PW) and SuperCollider2 (SC2) called PWCollider. This work has been motivated by the need to form a bridge between various composition tools found in PW and sound synthesis. PWCollider gives an alternative way to work with SC2. The user can generate SC2 patches visually by choosing a set of boxes and making connections between them. The system can be used either in draft-mode or score-mode. In the latter case PWCollider is used in conjunction with a compositional tool which allows to generate complex, multi-layered musical textures.

1. Background

PWCollider is a tool for software synthesis control implemented in PatchWork (PW, [1 and 2]) and provides a graphical interface to SuperCollider2 (SC2, [3]). PW is a visual programming environment (based on Lisp and CLOS) for computer-assisted composition. SC2, having a syntax that resembles Smalltalk, is a text-based software synthesiser. The main aim is to combine the various visual PW elements (such as musical scores, breakpoint-function editors) with the real-time capabilities of SC2. This interaction is in this paper a very natural one as both environments support similar high-level programming structures (lists, functions and objects) and dynamic memory management. While the visual approach of PWCollider offers several advantages compared to text-based programming, a great effort has been made to preserve the dynamic nature of SC2.

Graphical synthesis environments have been criticised of being static [3]. This paper hopes to demonstrate that a visual system can be dynamic if it is based on a dynamic base language. Furthermore we would like to stress the benefits of combining visual and text-based approaches in complex musical problems. In our case the user can add new synthesis modules to SC2 using text-based programming. These modules can then be easily incorporated into PWCollider. For instance the following expression is sufficient to create a new menu-item in PWCollider:

```
(create-SC-menu-item
  "SC-GUI-menu"
  "SCslider(pos, str, val, minval, maxval, step, wrap)"
  '(1 "sl" 1.0 0.0 1.0 0.0 linear)
  "SC slider with label and numbox"
  "C-SC-patch-slider-box")
```

When this menu-item is selected a new instance of a “SCslider” box is created in the current patch window (the “SCslider” box is discussed in section 6).

2. Environment

PWCollider contains all (currently over 200) unit generators found in SC2. Besides these basic modules PWCollider has several special tools that are used to solve problems that one encounters with more complex examples. These cases include: cascading tools, SC functions, event spawning (scheduling events), assignment statements, control windows with real-time sliders and order of execution.

A PWCollider patch consists of boxes and connections. The patch is translated into a text file and executed in SC2. Figure 1 gives a simple PWCollider example:

```
frequency -><duration
```

Figure 1. A sinosc example with an envelope, vibrato and pan.

PWCollider is operated in two basic modes: draft-mode and score-mode.
3. Draft-mode

In the draft-mode there is no explicit notion of a score or notes (note-kind of behaviour can be simulated by scheduling the patch). Figure 1 above gives a simple example of a draft-mode patch. The user can use ordinary PW boxes to calculate constants at the leaves of a patch. One can also mix PW boxes that create lists. This feature gives the system great flexibility as it allows to create complex SC2 structures using Lisp lists. Thus for instance the powerful "multi channel expansion" property [3 and 4] of SC2 is supported automatically by the system.

Figure 2 below gives a more complex example (this is a PWCollider version of the SC2 "cymbalism accellerando" example by James McCartney) containing several list-boxes that are used to build SC-arrays.

This patch demonstrates the dynamic nature of both SC2 and PWCollider. The number of filters of the "klank" module can be changed simply by editing the numbox in the upper left corner of the patch.

4. Score-mode

Score-mode is used in conjunction with a user-library called Viuhka [5]. Viuhka is a compositional environment for creating complex, multi-layered musical textures. These can be used to produce scores for instrumental music and to provide material for sound synthesis. Score-mode is similar to the Csound system in that it produces among others an orchestra-file and a score-file. An important improvement is though that the parameter fields are not restricted to floating point numbers but can also be symbols, lists, list of lists, path names, breakpoint-functions and SC2 expressions.

Figure 3 gives a score-mode example. It defines a five second musical texture. The central box is the "mk-viuhka" box that calculates the resulting texture. Its first input is connected to a breakpoint function editor ("viuhka-bpfs") whose contents defines the overall pitch skeleton of the result. Viuhka-bpfs provide the main pitch material which can further be reacted to (or elaborated by) different sorts of Viuhka ornaments. The second input defines the duration of the texture (500 PW-ticks = 5 seconds). The third input takes the parameter lists which are in turn used to build the resulting note information. This input is connected to a "v-params" box which provides various parameters that define timing, ornament types, viuhka-bpf indexes, etc. Parameter fields for the individual notes are defined with the "i-params" box. The inputs can take constants, lists, Lisp expressions, break-point functions, etc.

The result can be inspected both as a piano-roll representation and as a chord sequence (Figure 4).
The instrument definitions in Figure 3 are given inside the "orch" box. This box defines the orchestra used by the patch. Figure 5 shows the contents of the "orch" box. The upper part of the figure gives a "sine" instrument abstraction. The lower part in turn shows the contents of the “sine” box.

5. Data-base Libraries

In order to simplify the organisation of large collections of synthesis material PWCollider contains specialised data-base libraries (sample-instruments, vowels, filter-banks). These are used to pass various sorts of information (path names, analysis data) between note-events and instruments. Data-base libraries are organised as an object hierarchy allowing data to react intelligently to the current context via specialised methods.

For instance a sample-instrument is a collection of samples that contains besides path names of the individual samples a key-range information for each sample. The sample-instrument protocol resembles the key-mapping scheme found in commercial samplers: a given pitch determines which sample is to be played. Also the scheme is responsible for finding the right speed factor so that the chosen sample will have correct pitch. Unlike commercial samplers sample-instruments are not limited to the equal tempered scale. Any micro tonal tuning can be used here. The following expression demonstrates how to add a new data-base item called “piccolo” to the library of sample-instruments:

\[
\text{(add-new-item-to-library *basic-sample-lib* "piccolo"}
\text{("picc C5" 72 1 72)}
\text{("picc F#5" 78 72 78)}
\text{("picc G#6" 84 78 84)}
\text{("picc F#6" 90 84 90)}
\text{("picc C7" 96 90 96)}
\text{("picc F#7" 102 96 127)\text{HD:Sound:samples:Orchestra:Woodwind:piccolo:})}
\]

After the desired sample-instruments have been added into PWCollider they can be referred to with a box called “basicsamples”. Figure 6 shows an instance of this box. It has one input giving the name of the selected sample-instrument (in our example the name is "pianof"). The “basicsamples” box can be opened by double-clicking it. When this is done a dialogue appears containing all names of sample-instruments known by the system. The user can choose another name by selecting the desired sample-instrument:

![Figure 6. Choosing a sample-instrument from the sample library.](image)
6. User Interface Windows

PW Collider allows the user to build simple SC2 user interface windows. Currently the only controller that is supported is a real-time slider with a number box and a text string. Sliders are generated with a “SC slider” box. Figure 8 gives an example where the frequencies of a sinosc module are controlled with sliders (“sl1” and “sl2”). A third slider (“pan”) is used for the pan position:

Figure 7. A Viuhka patch containing a sample-instrument.

Figure 8. A patch that generates three sliders.

Figure 9 shows the resulting SC2 window.

Figure 9. A SC2 user interface window with three sliders.

7. Conclusions

While PW Collider is still under development we believe that it is already a useful compositional tool. The user can prototype and test ideas in a fluent and efficient manner. By combining the benefits of visual and textual programming it provides a platform for both novice and expert users.

Acknowledgements

The author wishes to thank Prof. Paavo Heininen (Sibelius Academy) for his help in formalising and developing the compositional ideas behind the Viuhka project.

This work has been supported by the Academy of Finland.

References