Embedding Synthesis ToolKit (STK) Instruments in Pure Data (PD) Externals

REALSIMPLE Project*
Edgar J. Berdahl, Nelson Lee and Julius O. Smith III
Center for Computer Research in Music and Acoustics (CCRMA), and the
Department of Electrical Engineering
Stanford University
Stanford, CA

Abstract

The Synthesis ToolKit (STK) is an ideal environment for creating physical models of musical instruments. Here we explain how to embed STK models of musical instruments into `pd` external objects. This makes them accessible to the whole `pd` community, and in particular, to people who do not know how to write STK code with C++.

1 Introduction

In this lab, we will provide two sample files: one for instruments and one for effects in the STK. We will address major issues for interfacing STK with `pd`. We have provided sample C++ code for both effects and instruments: clarinet.cpp[1]. We have also provided the Makefile[2] for compiling clarinet.cpp and a test_patch[3] that uses the STK clarinet.

2 Required Software

- Text editor such as Emacs.
- Pure Data. This is the main software used in the RealSimPLE project. It can be downloaded here.[4]
- The Synthesis ToolKit, which can be downloaded here.[5]
- flext, a series of macros, that adds function calls that interface STK with `pd`. flext can be downloaded here.[6]

*Work supported by the Wallenberg Global Learning Network
[1]http://ccrma.stanford.edu/realsimple/stkforpd/clarinet.cpp
[2]http://ccrma.stanford.edu/realsimple/stkforpd/Makefile
[3]http://ccrma.stanford.edu/realsimple/stkforpd/test-patch.pd
[4]http://crca.ucsd.edu/ msp/software.html
[5]http://ccrma.stanford.edu/software/stk/
[6]http://grrrr.org/ext/flext/

3 The Architecture

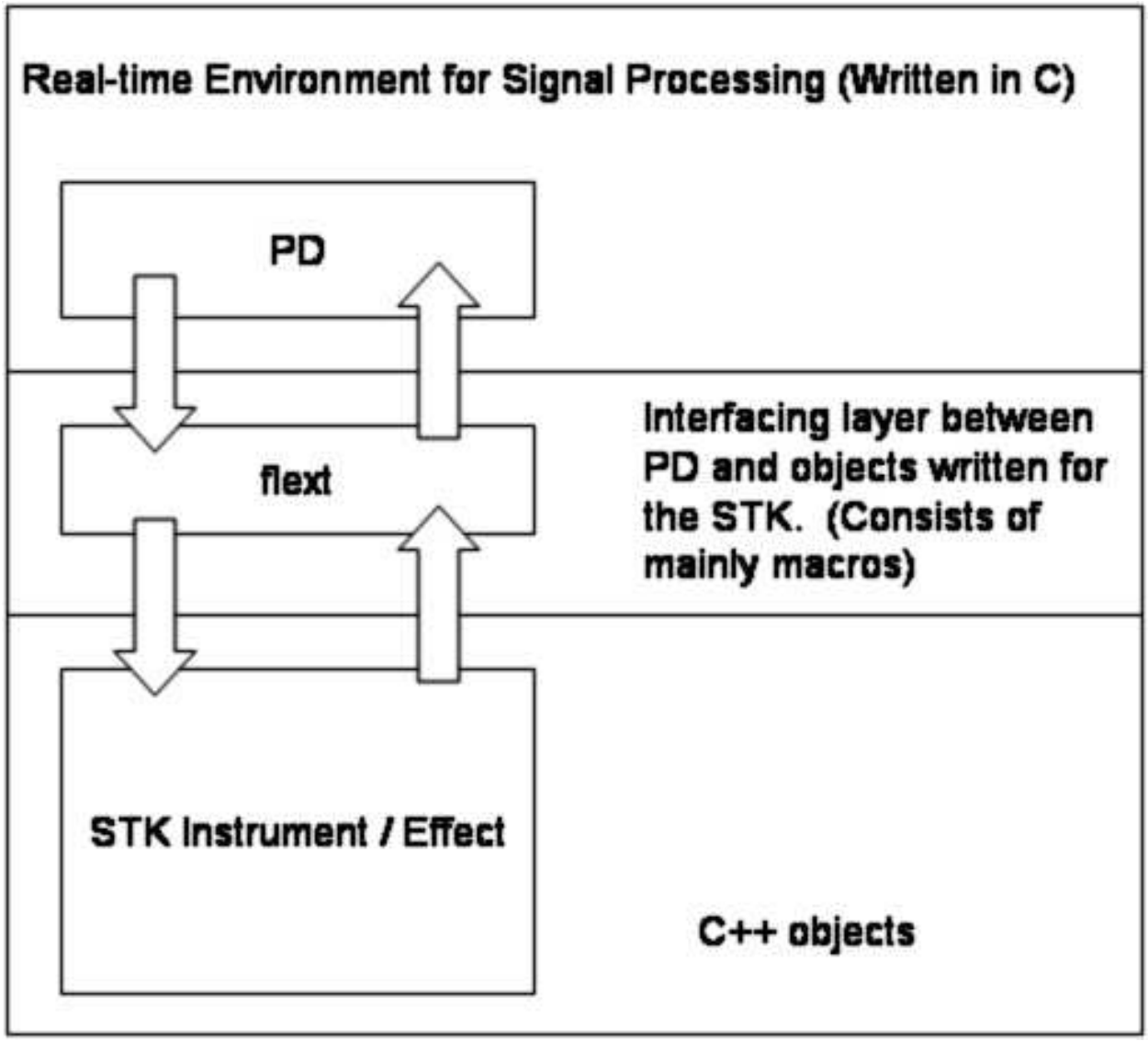

Figure 1: Graphical display of how `pd`, flext, and the STK interface with one another

Figure 1 shows how the three required software components interface with one another. Unlike C++, `pd` is an interactive signal processing suite that allows an individual to interactively create objects, link them together, and analyze data. The usefulness of objects written for the STK and in C++ lies in their ability to develop and represent complex physical models such as those representing models of how a wind instrument such as the clarinet is played.

Now, flext understands how `pd` communicates with externals, objects not designed and coded within the `pd` framework. Furthermore, it understands the nature of STK objects in how they are driven and how they are used. Thereby its main use is in providing the links between calls made out from `pd` into calls into STK objects, and vice versa.

To motivate our intentions once again, the driving application we are targetting is bringing STK instruments and effects into `pd` so that these objects created in C++ can be interactively used and instantiated within the `pd` framework. There are many advantages to this. Listing a few, one may be developing a unique guitar tone that uses a physically-based model of an acoustic guitar. Or, one may be developing a complex model for wavefield-synthesis and be wishing to easily connect the model to signals from the real-world such as the output of three microphones.

4 A Wrapper Class for Your STK Class

The file `clarinet.cpp` defines a wrapper class that interfaces `pd` with the actual STK Clarinet object. Here we will discuss the necessary header files for the clarinet example.

4.1 Header Files, Libraries, Compilation Issues

```
#include <flstk.h>


#if !defined(FLEXT_VERSION) || (FLEXT_VERSION < 401)
#error You need at least flext version 0.4.1
#endif

// Include here whatever particular STK declarations are needed
#include "Clarinet.h"
#include "PRCRev.h"
#include "JCRev.h"
#include "NRev.h"
#include "Echo.h"
#include "PitShift.h"
#include "Chorus.h"
#include "BiQuad.h"
#include "Resonate.h"
```

The code above tells the C++ compiler what the definition of the flext classes and the STK classes are. When compiling `clarinet.cpp`, it will be crucial to specify where the STK header files, such as `Clarinet.h` are as well as where the compiled library exists. In the `Makefile` included with this lab, the header files are found in /usr/include/flext and /usr/include/stk/. The compiled libraries are found in /usr/lib/libflext-pd_s.a and /usr/lib/libstk.a. It is standard to have the header and library files in these locations, but you will weant to double-check to ensure that they are located in those paths.

5 Using Given Files as Templates

Once you get the provided clarinet code compiling and running in `pd`, you are well on your way into bringing your own STK classes into `pd`. Simply use the above code as a starting template. A

good way of re-using the work that has been done is to incrementally code your changes into the template. Here's a work-plan of how we would recommend using the provided files.

- Copy the files into a new directory. Make sure the project still compiles and runs without any changes.
- Rename the files and class names to match yours.
- Do the following until all changes to the code have been made.
  - Make single changes to the working code.
  - Ensure that the code still compiles and runs after each change.
  - Repeat.

Again, since we have provided for you a working link from STK to `pd`, we highly recommend you begin with the working code and incrementally build/replace it making sure it always stays compiling and working.

6 The Code

Now that you understand the architecture and libraries, it's time to jump in! This is the beauty of open-source code. You can see everything that everyone else has written, and you can change it however you like. We have tried to self-document our code. This means that given the context and variable names, you should be able to tell what is going on. We added extra comments around portions where flext macros are called. Place the following files in the same directory:

- `clarinet.cpp`[7] contains the C++ code for creating the clarinet~ object. The Clarinet() object from STK is embedded inside of clarinet~.
- The `Makefile`[8] tells the compiler how to combine all of the different pieces of code.
- Once you run the `make` command in the same directory as the above files, the subdirectory `pd_linux` will be created, and the dynamically-linked library file `clarinet~.pd_linux` will be placed there.
- Add `pd_linux` to the path for `pd`, and then open the `pd` patch `test-patch.pd`[9] in `pd` to test your code!

7 Acknowledgments

We would like to thank Larry Wu for providing the initial source code for implementing STK wrapper classes.

[7] http://ccrma.stanford.edu/realsimple/stkforpd/clarinet.cpp
[8] http://ccrma.stanford.edu/realsimple/stkforpd/Makefile
[9] http://ccrma.stanford.edu/realsimple/stkforpd/test-patch.pd