A Java Framework for FX Development

Pablo Fernández-Cid

Dpto. Electrónica y Teoría de Circuitos
Universidad Europea de Madrid
pablo.fernandez-cid@tel.uem.es

Javier Casajús, Lino García

Dpto. Señales, Sistemas y Radiocomunicaciones
Universidad Politécnica de Madrid
javier, lino@gaps.ssr.upm.es

ABSTRACT

This paper describes the first version of a Java archive (a term basically equivalent to 'library' in other programming languages) that has been developed and made available as public domain software for the benefit of the DAFX community, and the COST-G6 web pages in particular. The library is available both as source code and ready to run bytecodes.

The archive defines an easy to use set of classes that are modelled after an effects processor. Ready made classes like 'Effect', 'Page of Parameters', 'Integer Range Parameter', 'Real Range Parameter', etc. serve as a basis to implement effects and share them. Effects can run from web pages or as stand alone applications, sharing unified look and feel in a platform independent graphical user interface. The programmer only needs to specify the parameters the effect will use, and the method (function) that will apply the effect to each new sample. An automatic GUI interface is created, that enables the adjustment of parameters as well as the specification of input and output files to be used during processing.

Developing Java audio effects according to the proposed scheme will allow transparent integration into more complex multiband and multieffect architectures that will be added on a second version of the archive.

1. INTRODUCTION

There are many ways in which researchers and teachers on DAFX can make programmed implementations of the effects available to third parties. On the COST G6 action the preferred method has been the use of Matlab, as a high level language that can be compact, easy to read, reliable across a wide range of platforms and operating systems, and with tools usually available at research departments.

In order to gain attention from the end users and also for teaching and demoing the effects, we have programmed a set of classes in Java language that can be used as a starting point for Java DAFX implementation. Given the closeness of Java and C/C++ (particularly for inside-function definition), it is expected that even non-Java programmers will be able to use the framework with little effort. The fact that Java development and debugging tools are available for free on a wide variety of platforms also supports the opportunity of this approach.

The most repetitive tasks on building both autonomous stand-alone applications and applets (applications that will run from html pages for www, started by the navigator) for DAFX are been already resolved on the archive (not unlike the conventional plug-in approach in commercial software systems, but with a more limited scope in terms of GUI). The programmer only needs to describe the set of parameters his/her effect will use, and the method (function) that performs the effect for each sample of the signal.

Java allows easy integration of the applets inside DAFX tutorials on the web or as part of electronic books to be read with web browsers, making possible true interactive use of the effects on the reader's side. Also true machine and O.S. independence coming from Java should be welcome: the same single code will operate properly from most conventional platforms.

Simple implementations of well known effects have also been included as examples in the first release (delay, chorus, flanger, waveshaping). Some of them are direct clones of Matlab implementations available at COST G6 web site.

2. ARCHIVE CONTENTS: CLASSES

The archive contains already made definitions for a set of classes that ease the coding of Digital Audio FX into Java language. Classes available model an 'Effect', a 'Page of Parameters' (a collection of related parameters), an abstract 'Parameter' (defining several characteristics that are common for anykind of parameter), and several particular parameter styles (like those that take values from an integer range, a real range, a set of names, etc.). There are also classes that hide the management of audio files in standard formats.

Basically, the programmer of a new effect defines a new class as a child of the general Effect class. All the capabilities of the Effect class are inherited in such a way that the programmer only has to define two functions or methods to complete his particular effect: one that defines the names, limits, and units for the set of parameters the effect includes, and another one to apply the effect to a single sample. The framework will add, with a default standard look and feel that mimics what is usually available from hardware

FX units, a graphical user interface to the application, allowing control of parameters, processing of files, etc.

In a second phase, a set of new classes will be added giving support to the use of these Java FX as building blocks of more complex processing systems. The process of turning a conventional single-band description of an effect in to a multiband implementation will be automated, opening doors to experimentation on the multiband approach to DAFX, which many of us consider as a must on current high quality audio effects.

3. EFFECT CLASS

The framework available at the first phase of the project defines the 'Effect' class as the central piece in the framework. 'Effect' models a general FX. The Effect class supports stand-alone execution of the FX on the users site, as well as execution from an html page with the help of an Internet browser (in such a way that COST G6 web pages will be able to include online executable applets).

The Effect class includes properties for administrative purposes like name, version, author, date (these will ease search through the FX database if ever necessary). Other properties allow the inclusion of html files: 'htmlHelp' (the name of an html file that will be displayed if the user recalls help on a particular FX) and 'htmlAbout' (another html file describing author/s name, affiliation, address, e-mail, etc.).

At the Effect level there is a 'paramArray' property giving access to a collection of parameters for the effect, and a 'paramPageArray' property holding a set of pages each one with several parameters. Accordingly, 'selectedPageNum' is the property that holds the index of the page currently being used/displayed. This arrangement is similar to how commercial effect units allow access to its many internal parameters. The property 'programName' is used to allow naming of 'presets' for a particular effect that can be stored in and recalled from the file system. Being able to share not only the effect code but also some particular adjustment for its parameters, is welcome for many purposes: the effect author can include several examples of adjustments to be applied on a given original sound file (no need to store or distribute several processed files to illustrate the differences according to the values of parameters); if anyone finds some unexpected problem or curiosity for a given adjustment of parameters, it can be shared with the author or other users; etc.

Methods available at the Effect class include: the constructor (always needed, is where the programmer describes the parameter pages and parameters for the effect), apply (the core of the FX process); nextParamPage and prevParamPage (to allow adjustement of the 'current page' number); savePreset and loadPreset; listParamsToTextFile (for documentation purposes, generates a list of the parameters and values applied for a given program).

It must be stressed that only the constructor and the 'apply to one sample' methods must be defined by the programmer: the JAVADAFX framework will support and integrate the new effect properly (including a graphical user interface, access to the file system, etc.).

Defining just the 'apply to one sample' method, a default behaviour for 'apply to block' and 'apply to file' (based on successive calls to 'apply to one sample') exists. Of course, if a particular effect benefits from block processing, the programmer can override the default behaviour for those methods.

The Effect class is responsible (both for stand-alone and applet-like execution) for setting up the basic GUI. In order to operate properly with non up-to-date web browsers and java interpreters, many of the latest additions to Java have not been included (like Swing), and instead a somewhat rude look has been chosen. We prefer the benefit of universality to a price-awarding GUI. The GUI gives access to help and about files, allows input and output file specification, saving and loading of presets, and it also manages navigation through the collection of pages the Effect possesses. Displaying of the parameters themselves is handled by the pagaParameter class: all the Effect does is to reserve an screen area for the needs of the parameter page. In this way, if there is a page of parameters that can benefit from a non-standard presentation of parameters, the programmer can extend the parameter page class and override the display page method.

4. PARAMETER PAGE CLASS

The 'ParamPage' class is useful to collect several related parameters in a single entity (i.e. the LFO adjustments of a Chorus). If the programmer wants it, a single parameter may appear at several pages (think of a 'most used parameters' initial page, and several other more detailed pages for advanced editing).

Properties at the paramPage class include pageName, paramArray (links to the collection of parameters the page holds), selectedParam (index of 'current' parameter relative to the page), and methods like nextParam and prevParam (to highlight or select a particular parameter for editing), and a default version of displayPage.

The default display uses a simple interface, modelled after an multi-line alphanumeric LCD with cursor and inc/dec buttons (that callback inc/dec methods of the proper parameter). Of course the default can be override. In particular an scrollParamPage class (extending from the basic paramPage) is also included that uses on-screen 'faders' as the default display function for a more comfortable feel.

Programmers who feel they need a particular way of arranging the presentation of the parameters can extend the paramPage class and generate their own pages with a particular look that suits their needs. Of course, several 'styles' of display can be combined at different pages in a single effect.

5. PARAMETER CLASSES

There is a global 'parameter' class that models capabilities common to any parameter style and allows unified treatment of all kinds of parameters. This abstract class includes properties for name, units,… but relies on children classes to define other properties like current value, max, min, resolution, default value. Also common for all styles of parameters are some methods like increment value, decrement value, set value, get value, get value full string (returns an string composed of name+value+units). Clearly some of these methods have to be defined on the child class. Children like 'paramIntRange', 'paramRealRange', 'paramListOfNames', 'paramBoolean',etc. are already available.

6. AUDIO FILE MANAGEMENT

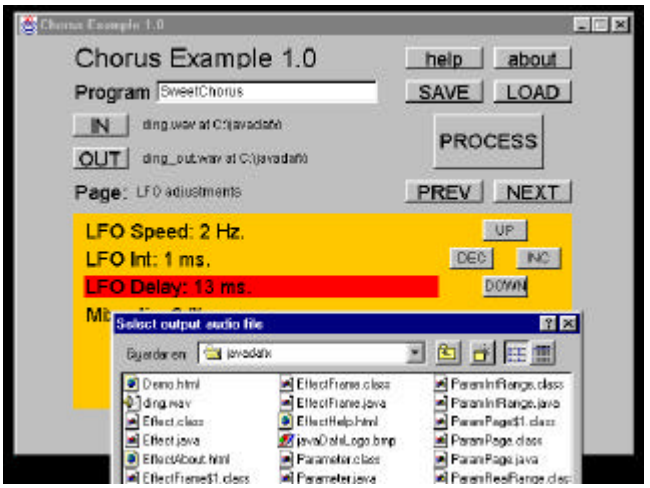
The Java2 current distribution (i.e. jdk 1.3 development tools and associated jre runtime environment), already includes an ambitious set of classes in order to access audio and MIDI in a properly high-level device and platform independent way (available as javax.sound.* classes). Some of the Java2 available classes are useful to load and save audio files in several conventional formats. Unfortunately, is not valid to presume that people's browsers are currently up to date on this particular subject. According to this fact, we have decided to program the wav read/save methods and only conventional file management classes are used in the current revision of the code. This extends the range of possible users.

Given the fact that file details should not be of concern for effects programmers, when the time comes that Java2 increases its installed base, the file access will use the javax.sound.* capabilities for a broader range of options in file formats, including direct access to sound hardware.

7. GUI

An screenshot of an example running stand-alone on its own window is shown in the figure (it is running on a PC with Windows95). The general name and version number of the effect is shown. The 'Program' text edit box allows naming of a particular set of parameter adjustments, that can be save for later reload, in order to build a library of preferred or useful adjustments (presets).

Input and output audio files can be selected (using standard file dialog boxes, as shown in figure). The parameters can be adjusted page by page, with each page showing its name. Navigation through pages is achieved by means of 'prev' and 'next' buttons. Then there is an area in the window that is reserved for the current page, in order to display (on its own style) the parameters and values. The 'standard' display style is shown in the figure. It uses a text list of parameters and values (with current parameter highlighted), accompanied by cursor and inc/dec buttons.



8. SECURITY ISSUES

Java has considered from its early days the issue of security. Given the fact that applications and/or applets are to be downloaded from a web site, its is important to assure end users that they can run the applications without concerns about possible damages to their systems and files.

Early Java allowed full access only to the stand-alone applications, and no file, system variables, etc. access at all to the applets. From Java 1.1 on, it becomes possible to sign and certify applets with conventional cryptography algorithms. In this way the end user could relax these access restrictions also to applets, if he/she is confident about the organisation responsible for the web site.

The archive classes are signed and certified and are available from the original COST-G6 site. If the new effects are programmed according to the stated guidelines, they will not need to be signed as they will not proceed to critical operations (these operations should only be performed from the originally supplied classes within the archive). So the effect programmers do not need to perform the tedious tasks of signing, etc. If the end user allows access to the file system only to the original COST G6 site signed and certified classes, he/she can be confident that no dangerous operations will be carried out by a maliciuos effect.

9. HOW TO PROGRAM FOR JAVA DAFX

In order to program a new effect using the Java DAFX archive, only the description of parameters and the method or function to apply the effect to one sample are needed.

The JavaDAFX archive can be imported with:

```
import javadafx.*;
```

Then a new class has to be defined for the desired effect, extending the parent Effect class. The template for new effects forces the programmer to define values for some static properties, as follows:

```
class Template extends Effect {

 // Fixed properties (fill with your own data)
 static String fxName="My Own Effect";
 static float fxVersion=(float)1.0;
 static String fxAuthor="Pablo Fernandez-Cid";
 static int fxNumParams=20;//no. of parameters
 static int fxNumPages=5;  //no. of pages
 static String HelpFile="MulTpDelHelp.html";
 static String AboutFile="MulTpDelAbout.html";

 // Constructor  (…edited: see later…)

 // Apply  (…edited: see later…)

}
```

The constructor method (with the same name as the class) is where the programmer describes the set of parameter his/her effect will use. Also the arrangement of effects into pages has to be carried out. Definition of the parameters must be made using the parameter classes available from the archives (advanced programmers who may need an special not available type of parameter can define their own style as an extension of the parameter class, and it will integrate transparently).

```
// constructor (same name as the class)

Template() {

 //this line is mandatory
 super();

 //define parameters
 //with name,units,min,max,resolution,default

 ParamArray[0]=new
   paramRealRange("Del","ms",0,2000,10,250);
 ParamArray[1]=new
   paramIntRange("FdBack","%",0,100,1,50);
 (…etc…)

 //create parameter pages
 //with name and no of parameters

 ParamPageArray[0]=new paramPage("1st tap",5);
 (…etc…)

 //arrange parameters into pages
 //with parameters and position into page
 ParamPageArray[0].addParam(paramArray[0],0);
 ParamPageArray[0].addParam(paramArray[1],1);
 ParamPageArray[0].addParam(paramArray[2],2);
 (…etc…)

}
```

The apply to one sample method is where the programmer describes the action the effect is supposed to perform.

```
 // Apply (defines the process)

 float applyOneSample (float sample) {
     (…)
     //Example:
     //output=sample*parameter[5];
     //return output;
 }
```

From this 'apply to one sample' method several other methods are defined in a default way: apply to a block, apply to a file. In particular, if a multichannel or stereo audio file is open, the call to apply to one sample will use and produce an array of floats instead of a single float. The default behaviour is described in the archive as follows (separately processes each channel):

```
float[] applyOneSample (float sample[]) {
  //default, override in your class if needed
  int n;
  output=new float[sample.length];
  for (n=0;n<sample.length;n++){
    output[n]=applyOneSample(sample[n]);
  }
  return output;
}
```

10. INCLUDED EXAMPLES

The first release of Java DAFX includes examples that implement a multitap delay, a modulated delay (for chorus/flanger processing), and waveshaping. An automatic multiband extender is on the works that will take any of the effects described as full-band, and allow its use in a multiband approach.
Current revision will operate properly from Java 1.1 enabled browsers and interpreters. There is no need to upgrade to the more up to date Java2, though at some later date (as Java2 increases its installed base) the archive will transparently migrate to Java2 in order to use the javax.sound capabilities.

11. REFERENCES

[1] Java home page (downloads, tutorials, etc.): http://java.sun.com
[2] Java Sound: http://java.sun.com/products/java-media/sound/