EASYSP: THE EASIEST FORM TO LEARN SIGNAL PROCESSING INTERACTIVELY

Javier Vicente, Begoña García, Amaia Méndez, Ibon Ruiz, Oscar Lage

University of Deusto (*http://www.pas.deusto.es*)

ABSTRACT

This paper describes an educational tool to facilitate the study of digital signal processing. This application offers a GUI support for integrating signal processing demos, using plugins that execute Octave/Matlab functions. On the one hand, this application permits students to change parameters for several examples, classified in categories such as modulations, filter design, speech and image signals analysis and processing. On the other hand, it offers the possibility of easily implementing two kinds of new signal processing plugins, XML based and java based.

Index Terms— Education, Signal processing, Software prototyping, Simulation software

1. INTRODUCTION

As it is known, students assimilate knowledge better if they have interactive visual examples [1]. For example, if we wanted to explain to the students how to use the Chebyshev low pass filter, we would provide an application with which students could vary the parameters that define the filter. After that, we would check how these changes affect the magnitude and phase diagram of the frequency response. Those practices were developed with freely redistributable software Octave. The disadvantage of this tool is that it cannot carry out Graphic User Interfaces (GUI). This fact does not allow students to achieve interactive applications as they would develop.

The first step to make GUI possible in algorithms developed by students was the joPAS Application Program Interface (API). This API permit to implement GUI quickly in Java program language and keep signal processing calculus in Octave. This API was successful because it provides users with a powerful tool to develop signal processing applications. However, the main disadvantage of this API is that students have to know Java to develop user interfaces.

The second step was to release *easySP*. This application permits two kinds of signal processing plugins to be executed. The first are based on XML description files, in which, graphics user interfaces and Octave sentences are specified. The second ones, consist on Java programs. Thanks to joPAS API the execution of Octave sentences is allowed.

This application allows the create of demonstrations of algorithms of signal processing easily and quickly. The students do not need to learn another program language, apart from Octave.

Besides the educational use of the innovative tool presented, it could also be applied in a research environment. As an example, the authors have made use of easySP in a project for the analysis and processing of oesophageal voices, as well as in a research for modelizing vocal folds images.

2. OBJECTIVES

The principal objective is to obtain a tool that would allow the study of the signal processing in an effective, quick and pleasant way. The learning is divided into two depth levels. In a first level, the student is only a simple user of the tool, with which he would interact with the available demonstrations of each of the areas of the signal processing. And in a second level, the student must have a more creative attitude, being he/she who increases the options of the tool adding new demonstrations of signal processing.

For the real effectiveness of the tool, it must fulfill the next requirements:

- *Free Software based* → In this way an application that can be freely distributed is achieved. The students will be able to install it in their personal computers without paying any kind of license.
- *Multiplatform* → The same application would work in different operative systems without doing excessive changes.
- *Intuitive* → The program must have a very intuitive interface which allows the user to interact very quickly with the tool.
- *Complete* → The tool must have examples that cover all the areas of the digital signal processing. These examples would be divided in categories that would be the following: Filter design, modulations, and frequency analysis of signals.
- *Open Source* → It must make it possible for students to access the instructions written to implement each example that forms the application.

- *Scalable* → The application must be very easily extendable through the addition of simple plugins. These plugins allow the students to increase the options of the tool by contributing new examples.

3. METHODS

Octave is the signal processing chosen language to implements the algorithms of *easySP* [2]. Octave is a high-level language for numerical calculation, whose syntax is compatible with MATLAB, but is developed by the free software community.

Octave is particularly oriented towards the scientific world. Among its main differences from other programming languages, the following stand out: native matrix operation, native operation with complex numbers and interpreted language.

These characteristics mean that scientific algorithms can be developed in a far shorter time then in other programming languages. Therefore, Octave is the ideal language for the development of digital signal processing algorithms, digital image processing, control systems, statistics...etc. Furthermore, there are a great many toolboxes that allow the user to avoid having to start from scratch when wishing to deal with a particular subject matter.

The joPAS API has been used as link between *easySP* and Octave [3]. This API was developed by the Advanced Processing of the Signal Team (PAS in Spanish) at the University of Deusto. With joPAS the user can create Java programs easily of through the implementation of all the mathematical part in Octave. Thus joPAS is used by *easySP* as a link between Java and Octave.

XML has been chosen as the support of plugin description. The principal advantage of the use of XML files is the easiness of reading and editing those files, any user can edit a XML file from any text processor capable of producing plain text files [4]. XML is a language very easy to read, interpret and modify.

XML technology has allowed the creation of an easy language for the configuration of processing. The user will define the input parameters, the graphs that the application must draw, as well as the functions that the application will have to use to process the input signal.

4. DESIGN

The chosen method for implementing the modules of the application is the use of plugins that define both the GUI and the algorithm of the simulation. Thus, through the use of XML the content of each of the six areas of the interface is defined. The six areas in which the interface has been divided are the following:

1. Plugin title.
2. Graphical representations area.
3. Set of graphs to be displayed.
4. Input parameters area.
5. Processing execution buttons.
6. Text area for the theoretical explanation of the algorithm.

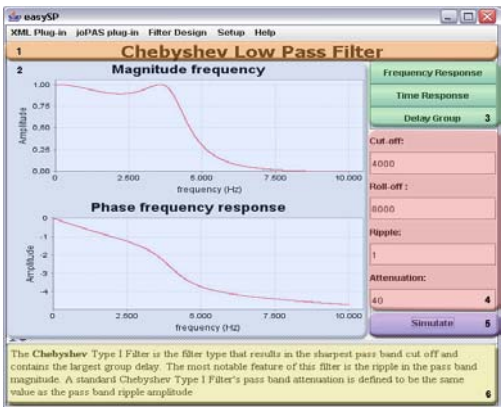

Figure 1. Parameterizable areas of the graphical interface

For the graphical interface definition it has been defined a specific XML language. The defined tags serve both for indicating the elements to be displayed in each area and to specify the algorithm to be executed. The tags defined in this language are the following:

- *Category*: element that shows the category in which the processing will be listed. Thanks to this element the application will arrange all the plugin that are added by categories.
- *Title*: this element contains the title of the window generated by the application.
- *Description*: theoretical explanation of the processing implemented in the plugin.
- *Input*: they allow defining the default value that the input will take, the name of the variable in Octave and the name of the input parameter.
- *Graph*: these elements will identify unequivocally every graph.
- *Button*: elements that will define every set of graphs to visualize.
- *Function*: define the call to the Ocatve's function or sentences, which implements the signal processing.

To examine the XML configuration file structure the following example will be analyzed, in which it has been defined an algorithm of a Chebyshev's low pass filter:

To generate plugin's XML configuration file, the desired elements for each area must be identified. Once it has been done, the description of the content of the XML file can be performed. This description of the elements could be grouped by each of the areas described previously.

```
<?xml version="1.0" encoding="UTF-8"?>
<Plugin version="1.0">
  <Category>Filter Design</Category>
  <Title>Chebyshev Low Pass Filter</Title>
  <Description> The Chebyshev Type I Filter is the filter type that
      results in the sharpest pass band cut off and contains the largest
      group delay.</Description >
  <Input value="3000"  OctaveName="fs">Cut-off</Input>
  <Input value="4000" OctaveName="fc">Roll-off </Input>
  <Input value="1"  OctaveName="R">Ripple</Input>
  <Input value="40" OctaveName="A">Attenuation</Input>
  <Graph>northGraph</Graph>
  <Graph>southGraph</Graph>
  <Button> <Name>Frequency Response</Name>
    <Graph Title="Magnitude frequency response" xTitle="Frequency
      (Hz)" yTitle="Amplitude" showIn="northGraph" varX="F"
      varY="m"/>
    <Graph Title="Phase frequency response" xTitle="Frequency (Hz)"
      yTitle="Amplitude" showIn="southGraph" varX="F"
      varY="p"/> </Button>
  <Button> <Name>Time Response</Name>
    <Graph Title="Impulse Response" xTitle="Time (s)"
      yTitle="Amplitude" showIn="northGraph " varX="t" varY="i"/>
    <Graph Title="Step Response" xTitle="Time (s)"
      yTitle="Amplitude (Hz)" showIn="southGraph " varX="t"
      varY="s"/> </Button>
  <Button> <Name>Delay Group</Name>
    <Graph Title="Delay Group" xTitle="Frequency (Hz)"
      yTitle="Delay (s)" showIn="northGraph" varX="F" varY="g"/>
  </Button>
  <Function> <Name>Simulate</Name>
    <OctaveFile>filterDesign.m</OctaveFile>
    <Callback>[m,p,F,s,i,g,t]=filterDesign(fs,fp,R,A)</Callback>
    <RunOnStartUp>true</RunOnStartUp>
  </Function>
</Plugin>
```

Figure 2. XML code of the example configuration file.

Figure 3 shows the methodology to generate the XML files describing the easySP application plugins.

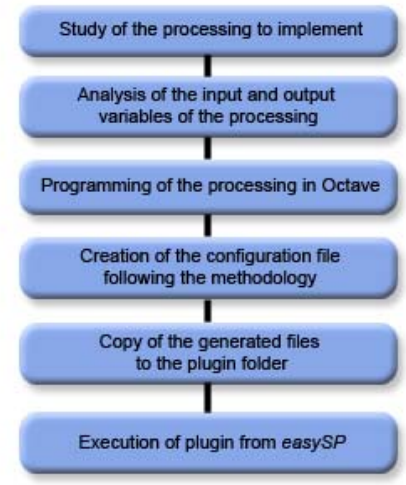

Figure 3. Configuration file definition methodology.

The plugin implementation using joPAS API is much more powerful, especially for creating graphic interfaces. The user, in this case, must create a Java file instead of configurating an XML file. This means that the user is totally free to develop his own joPAS plugins. Therefore, the students could enhance their creativity. Moreover, when they could make more complex and powerful plugins than using XML format, because joPAS plugins are not limited to a predefined structure. Anyway, the user should follow a few rules:

- The class of plugin must extend from the *JopasPlugins* class.
- The constructor of the class must have the following as input parameters: a *Jopas* reference and a *JFrame* reference.
- It must implement two methods: *getCategory* and *getTitle*.

5. RESULTS

As a result of the achievement of the specified objectives, the easySP tool has been obtained, covering the learning of the signal processing through the two proposed depth levels. Once the student chooses the module with which he/she wants to practice, a window like the one that can be seen in figure 1 is shown.

The example in figure 1 corresponds to a simulation of a Chebyshev low-pass filter, in which the user can modify the fundamental parameters that define the fundamental characteristics of the filter and analyze the behavior of it depending on the variation of those parameters. In the lower part of the window, the student finds a theoretical explanation of the filter behavior which can be verified undertaking different tests.

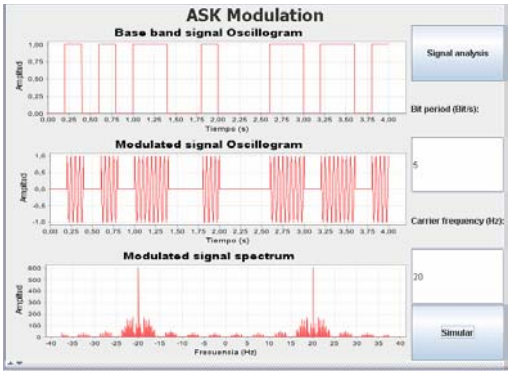

Figure 4. Simulation of an ASK modulation.

Once the student has understood the theoretical concepts proposed in the module he/she can continue with another one. For example, an ASK modulation, in which the time signal and the spectrum signal of the modulation can be analysed as shown in figure 4. This kind of teaching methodology permits the concepts of the signal processing to be assimilated in a pleasant way.

In the second depth level, students can develop new modules following the XML structure defined in Figure 2, or analyze the existing ones, to check the Octave sentences that are necessary to implement the suggested module. In Figure 5 the structure of the necessary XML file for the parameterization of the example of the low-pass filter in the figure 1 can be seen. As it is shown the creation of this file, that defines the user interface's structure, is simple and easy to understand.

```
function [m,p,F,s,i,g,t]=filterDesign(fs,fp,R,A)
    Fs=20000;
    [n,w]=cheb1ord(fs/(Fs/2),f/(Fs/2),R,A);
    [b,a]=cheby1(n,R,w);
    [H,F]=freqz(b,a,1024,Fs);
    m=abs(H);
    p=unwrap(angle(H));
    [i,t]=impz(b,a,[],Fs);
    s=filter(b,a,ones(1,length(t)));
    g = grpdelay(b,a,1024);
Endfunction;
```

Figure 5. Octave's function for a Chebyshev's filter.

Students can analyze all Octave instructions of the simulation of the filter that is implemented in the Octave's function *filterDesign* (figure 5), which contains all the instructions to calculate the frequency, impulse and step response, and group delay of the filter. This function has as input parameter the variables defined in the *Input* elements and returns the variables handled by the *Graph* elements.

```
function
[m,p,F,s,i,g,t]=filterDesign2(fs,fp,R,A)
    Fs=20000;
    [n,w]=buttord(fs/(Fs/2),f/(Fs/2),R,A);
    [b,a]=butter(n, w);
    [H,F]=freqz(b,a,1024,Fs);
    m=abs(H);
    p=unwrap(angle(H));
    [i,t]=impz(b,a,[],Fs);
    s=filter(b,a,ones(1,length(t)));
    g = grpdelay(b,a,1024);
Endfunction;
```

Figure 6. Octave's function for a Butterwoth filter.

If the student would want to implement a new module that performs the same but a Butterworth filter (Figure 7), the XML file to user would be practically the same, he/she would only have to change the *Title* field, the theoretical description of the module, the Octave's function to be invoked and write that routine (Figure 6).

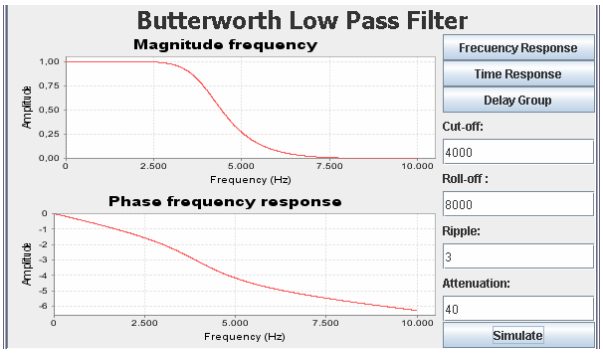

Figure 7. Simulation of the Butterword filter.

6. CONCLUSIONS

Thanks to the developed application, both the work of the lecturer and work of the student has been facilitated. The lecturer has a tool that allows him to transmit the knowledge of digital signal processing, by means of the small implementation of plugins like support to the theoretical contents that wishes to teach.

The students assimilate better the knowledge of signal processing by means of a tool with which they can interact. In addition, they can develop new modules without too much additional knowledge about Octave.

easySP could also help as an optimal framework for the development and evaluation of both degree and research projects.

7. ACKNOWLEDGEMENTS

The authors of this article want to be thankful to the students, especially to Mikel Mendezona and to Ekhi Arroyo Fernandez de Leceta, to have transmitted their experiences with the application to us and to contribute actively in the extension of the application by means of the creation of new plugins.

8. REFERENCES

[1] Begoña García, Javier Vicente, "Herramienta para la Simulación e Implementación Real de Sistemas Discretos FIR e IIR" in *Proc. TAEE'02*, Las Palmas, Spain, 2002.

[2] K. Hornik, F. Leisch, A. Zeileis, "Ten Years of Octave Recent Developments and Plans for the Future" in *Proc. DSC 2003*, Wien, Austria, 2003.

[3] Javier Vicente, Begoña García, Amaia Mendez, Ibon Ruiz, Oscar Lage, "Teaching Signal Processing Applications With joPAS: Java To Octave Bridge" in *Proc. EUSIPCO2006*, Firence, Italy, 2006.

[4] Clemens H. Cap, "XML goes to School: Markup for Computer Assisted Learning and Teaching" in *The European Journal of Open and Distance Learning*, 2000.