14th European Signal Processing Conference (EUSIPCO 2006), Florence, Italy, September 4-8, 2006, copyright by EURASIP

TEACHING SIGNAL PROCESSING APPLICATIONS WITH “JOPAS”: JAVA TO OCTAVE BRIDGE

J. Vicente, B. García, A. Mendez, I. Ruiz, O. Lage

ESIDE, University of Deusto
Avda/ Universidades 24, 48007, Bilbao, Spain
phone: + (34) 944139000, fax: + (34) 944139101, email: jvicente,mbgarcia,ibruiz,amendez,olage@eside.deusto.es
web: http://www.eside.deusto.es/grupos/eside_pas/

ABSTRACT

This paper introduces the "joPAS" programming API, which has been developed by the PAS research team at the University of Deusto. “joPAS” enables the use of “octave” variables and functions through a JAVA program. Therefore, this API makes it possible to not only develop signal processing applications quickly by implementing the application's graphic interfaces in Java language but also to carry out the scientific calculation in “Octave”. Students can easily learn the implementation of digital signal processing applications with this API.

1. INTRODUCTION

The development of the “joPAS” programming API arose from the PAS research team's need to develop prototypes of the algorithms used with a user's interface in its research areas. One of the main lines of research followed by the PAS research team is the regeneration of the oesophageal voice, in which several algorithms for improvement -already presented at a number of congresses [1][2][3][4]- have been developed. These algorithms were developed in “Octave” language, which, despite being ideal for signal processing algorithms, lacks a user interface. Therefore, so as to program a demo application of the work carried out by the team, we saw it necessary to re-implement the code in another programming language that enables the programming of graphic interfaces. This meant that the team would lose out on research performance in order to devote more time to less productive tasks. Thus the need for a tool allowing the reuse of “Octave”-developed algorithms as well as providing the possibility of user interfaces. This gave rise to “joPAS”, a bridge between "java" and "octave" for the rapid development of signal processing applications.

Once "joPAS" had been developed, the great potential this API had not only for research work and application development, but also for the sphere of education, due to its user friendliness. This API provides us with a tool that enables our students to carry out their own signal processing applications in a fast, simple way. When using “joPAS”, the student can focus his/her attention on the development of digital processing signal algorithm as the graphic interface implementation is simplified by using "joPAS".

2. AIMS

The main aim of the "joPAS" application is to make a tool available that can enable the rapid development of applications with a user interface that use signal processing algorithms implemented with other tools (octave). This being the main aim, a number of secondary objectives could also be achieved:

1. A reduction in the cost of licences for mathematical programs, through the use of free software, such as Octave, which specialises in digital signal processing.
2. Motivating students developing projects using digital signal processing techniques by avoiding programming difficulties.
3. An increase in the integration of an algorithm's results in a graphic environment of simple programming.
4. Favouring the creativity of students when carrying out the projects and dissertations necessary for their university degree.

3. METHODS

3.1 OCTAVE

“Octave” is a high-level language for numerical calculation, whose syntax is compatible with Matlab, but is developed by the free software community [5].

But what makes "octave" different from other programming languages?

“Octave” is particularly oriented towards the scientific world. Among its main differences from other programming languages, the following stand out:

1. Native matrix operation.
2. Native operation with complex numbers.
3. Language is interpreted.

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 a great many toolboxes that allow the user to avoid having to start from scratch when wishing to deal with a particular subject matter. For instance, if somebody wants to develop a digital voice-processing algorithm and needs to filter the signal by means of a Butterworth filter, he/she needn't implement this function as it already exists in the signal processing toolbox, which means that its use is unnecessary in the algorithm. This kind of toolbox, so highly specialised in scientific matters, cannot usually be found in other programming languages, which is yet another advantage for the development of this type of applications in "octave".

But what are the Disadvantages of “Octave”?

Although “octave” is an ideal language for the development of scientific applications as they can be carried out in a short time, it has some drawbacks, one of which is linked to the speed of computation. Being an interpreted programming language, “Octave” is slower than a compilable language, due to the fact that the latter generates native instructions for the processor, which takes less time.
The second disadvantage is related to the graphic environment. Applications with "octave" are executed on console with the single possibility of making graphic data displays. Therefore, this makes it impossible to develop user interfaces that he/she can interact with the application.

Therefore, “octave” is an extremely powerful programming language for the fast development of scientific algorithms, but one which is not used for the development of final applications due to its lack of graphic interface. So "octave" is usually used to validate algorithms. However, these algorithms are then *translated* into another programming language in order to obtain an application that enables the user to interact with it.

3.2 JAVA

Java is an object-oriented programming language developed by James Gosling and colleagues at Sun Microsystems in the early 1990s [6]. The language, which was designed to be platform independent, is a derivative of C++ with a simpler syntax, a more robust runtime environment and simplified memory management.

Operating on multiple platforms in heterogeneous networks invalidates the traditional schemes of binary distribution, release, upgrade, patch, and so on. To survive in this jungle, the Java programming language must be architecture neutral, portable, and dynamically adaptable.

The system that emerged to meet these needs is: simple, so it can be easily programmed by most developers; familiar, so that current developers can easily learn the Java programming language; object oriented, to take advantage of modern software development methodologies and to fit into distributed client-server applications; multithreaded, for high performance in applications that need to perform multiple concurrent activities, such as multimedia; and interpreted, for maximum portability and dynamic capabilities.

Primary characteristics of the Java programming language include a simple language that can be programmed without extensive programmer training while being attuned to current software practices. The fundamental concepts of Java technology are grasped quickly; programmers can be productive from the very beginning.

Programmers using the Java programming language can access existing libraries of tested objects that provide functionality ranging from basic data types through I/O and network interfaces to graphical user interface toolkits. These libraries can be extended to provide new behaviour.

Even though C++ was rejected as an implementation language, keeping the Java programming language looking like C++ as far as possible results in it being a familiar language, while removing the unnecessary complexities of C++. Having the Java programming language retain many of the object-oriented features and the "look and feel" of C++ means that programmers can migrate easily to the Java platform and be productive quickly.

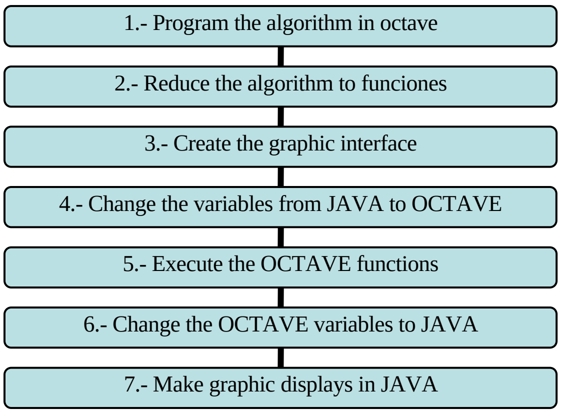
Can Java be used for scientific computation?

For scientific computation, there are more precise languages than Java and with more possibilities of calculation. In addition, these languages operate natively with matrixes and complex numbers, and Java does not do so. Therefore, languages like Matlab and “octave” are used in scientific computation. It is much more difficult to program scientific calculations in Java than in scientific languages, which implies a great disadvantage for Java.

Even so, there are people who use Java to develop programs that include scientific computation. In order to program scientific calculations in Java, it is necessary to spend much more time than in scientific languages, but Java has many options for the development of user interfaces. Free scientific languages such as “octave” only can be used in the command line and applications cannot be created with them either.

4. DESIGN

Taking the need to reuse the code of algorithms developed in "octave" as a premise (instead of encoding them into another programming language), and also considering that we had capable graphic user interfaces, the best possible proposal was searched for. It was resolved to develop the graphic interfaces in JAVA and to continue carrying out the scientific calculation in “OCTAVE”; however, no link existed between the two programming languages. Work was therefore started on the “joPAS” API, which was to serve as a link between JAVA and “OCTAVE”. “joPAS” enables the exchange of variables between both languages as well as the execution of “octave“ sentences from JAVA. The methodology for working with JAVA is as follows:


1.- Program the algorithm in octave
2.- Reduce the algorithm to funciones
3.- Create the graphic interface
4.- Change the variables from JAVA to OCTAVE
5.- Execute the OCTAVE functions
6.- Change the OCTAVE variables to JAVA
7.- Make graphic displays in JAVA

1. Program the required algorithm in “OCTAVE”, using all the power available.
2. Reduce the algorithm into “OCTAVE” functions so that its calculation is performed within these functions; functions to which only fundamental parameters need to be passed and which return the final results to the algorithm.
3. Create the graphic interface for the application in JAVA. This step can be simplified by using a development environment such as Netbeans.
4. The data necessary for the graphic interface are gathered in the user interface event functions and the corresponding “OCTAVE” variables are generated. .
5. The desired “OCTAVE” functions are invoked with the variables generated from the JAVA application, thus generating the exit variables.
6. After executing the “OCTAVE” algorithms, the exit variables obtained are requested from JAVA by passing the variables from “octave” to JAVA.
7. To end, the results can be seen on the graphic interface and, if some kind of graphic display were necessary, the functions added to the “joPAS” API would be invoked.

In short, the work procedure with “joPAS” would be as follows: develop the application in “octave”, as one would normally, and supply the “OCTAVE”-developed algorithm with a JAVA-developed bundle/wrapper which provides the user graphic interface. Therefore, the time spent implementing the the application with the graphic environment is minimised as only the application windows are used in JAVA and the logia of the application is obtained from invoking the functions developed in “OCTAVE”.

5. THE “joPAS” STRUCTURE

As already mentioned “joPAS” is a JAVA API that contains numerous classes; the most important would be the following:

- Jopas → This is the fundamental API class; it is in charge of administering the communication with “octave” in three following methods, *load*, *save* y *execute*.
- Matrix → This class contains the kind of data that understands the jopas class. It is a container of matrixes that can be either real or complex. The jopas class admits this kind of data in the *load* method and in the *save* method it always returns a Matrix-type object.
- JopasLabel → This class is a modified Label with an important *print* method. The *print* method is in charge of performing the graphic display of a signal on the label.

Below a brief code fragment can be observed, in which the above mentioned classes are used.

```
Double b = 2;
Matrix mA= new Matrix (a,"a");
jopas.Load(mA);
jopas.execute(“b=a+4”);
Matrix mB = jopas.Save("b");
```

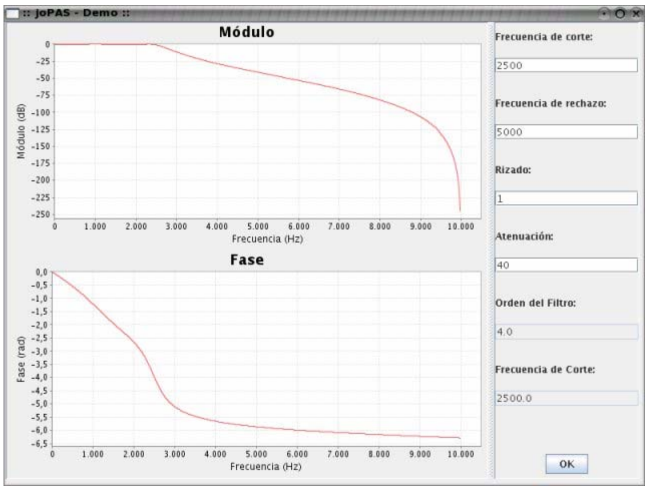
List 1. Example of “joPAS” use

In the list 1 fragment, it can be seen how a matrix object that will contain a scaled number is created. This matrix is loaded in “octave” with the Load method. Then an “OCTAVE” sentence is executed, which adds a unit to the generated variable. To finish, the result that has been obtained is recovered in “octave”, generating a matrix object in JAVA.

The API structure has been simplified in such a way that one only needs to know how to use three classes to be able to develop applications with a graphic interface. The API performs all the process of communication between java and "octave"- loading variables from java to "octave", executing "octave" sentences and saving variables from "octave" to java- in a transparent way for the user. Therefore, only a minimum time is needed by students who are able to programme in “OCTAVE” and in JAVA to learn how to use “joPAS”; which is our case.

6. RESULTS

As already mentioned, the implementation of signal processing applications using “joPAS” is extremely simple. Below a design application of Chevichev low pass filters, carried out as a demo of “joPas” use for our students, is described. The user interface implemented in JAVA can be seen in figure 1.



Frecuencia de corte: 2500

Frecuencia de rechazo: 5000

Rizado: 1

Atenuación: 40

Orden del Filtro: 4.0

Frecuencia de Corte: 2500.0

Figuer 1. Application user interface

First, one looks for “OCTAVE” sentences allowing the design of a Chevichev low pass filter as well as calculating its frequency response. The necessary instructions can be seen below (list 2)

```
[N,W]=cheb1ord(FC/10000,FR/10000,R,A);
[B,A]=cheby1(N,R,W);
[H,F]=freqz(B,A,512,20000);
modulodB=20*log10(abs(H));
fase=unwrap(angle(H));
Fc=W*10000;
```

List 2. “OCTAVE” codes

On the one hand, the application user interface is designed, a task that can be simplified with the use of an IDE that allows the creation of JAVA graphic interfaces in a visual form. On the other hand, the algorithm for the design of a digital filter will be implemented in “OCTAVE”.

The user interface is made up of:

- Two *JopasLabel*: in the top one, the frequency response module of the designed filter will be displayed, and in the bottom one, the frequency response phase of the filter.
- Four TextEntries, in which the following parameters will be introduced
    1. Filter cutoff frequency.
    2. Filter rejection frequency.
    3. Ripple effect in pass band.
    4. Rejection band attenuation.
- Two TextEntries, in which the following data can be seen:
    1. Filter order
    2. Designed filter cutoff frequency.
- A button to launch the process of the filter design. The actionPerformed method will make the necessary calls to implement the filter. In list 3, the sentences using “joPAS” to communicate the data in JAVA and “OCTAVE” can be seen.

```
//Read the values of the inut textFields of the GUI
String FC = jTextField1.getText();
String FR = this.jTFFrecCorte.getText();
String R = this.jTextField3.getText();
String A = this.jTextField4.getText();

//Generate de OCTAVE variables
jopas.Load(Double.parseDouble(FC), "FC");
jopas.Load(Double.parseDouble(FR), "FR");
jopas.Load(Double.parseDouble(R), "R");
jopas.Load(Double.parseDouble(A), "A");

 //Executes the Octave commands using local variables
jopas.Execute("[N,W]=cheb1ord(FC/10000,FR/10000, R , A )");
jopas.Execute("[B,A]=cheby1(N," + R + ",W)");
jopas.Execute("[H,F]=freqz(B,A,512,20000)");
jopas.Execute("modulodB=20*log10(abs(H))");
jopas.Execute("fase=unwrap(angle(H))");
jopas.Execute("Fc=W*10000");

//Write the values at the output textFields
this.jTextField6.setText(Double.toString(jopas.Save("N").getRealAt(0, 0)));
this.jTextField5.setText(Double.toString(jopas.Save("Fc").getRealAt(0,0)));

//Plot the graphic representation of the frequency response
this.jopasLabel2.paintLabel("F", "modulodB", "Módulo",
"Frequency (Hz)", "Module (dB)");
this.jopasLabel1.paintLabel("F", "phase", "Phase", "Frequency  (Hz)",
"Phse (rad)");
```

List 4. JAVA code for the GUI button method.

As can be seen in this example, by using “joPAS”, applications can be designed in a faster and a more simple way. Therefore, its use is quite straightforward for the students.

7. FUTURE PLANS

Our immediate aim for “joPAS” is that students continue using this API in their final year projects. We can therefore ensure that most of the time they devote to their final year projects will be spent on implementing processing algorithms in “OCTAVE”, the programming language they have the greatest knowledge of; thus minimising the time and effort required for the development of the graphic part of the application. This does not mean that the resulting applications will lack a quality graphic user interface, but, thanks to the use of “joPAS”, they will be obtained at in a faster way.

On the basis of the experiences undergone by our students using “joPAS”, points for improvement, which can be incorporated into the API, Hill be identified. A crucial step has been taken with the current version of “joPAS”; and it is our intention to keep “joPAS” going, to develop and improve it. The latter can be achieved by refuelling our minds with the feedback received from students’ experiences, which will not only encourage us to become even more deeply concerned with our students’ development process, but will also make them feel more deeply involved in the project.

8. CONCLUSIONS

Finally, it is appropriate to highlight the benefits obtained from using the “joPAS” API for the design of signal processing applications requiring a graphic user interface. These advantages are, on the one hand, that by means of this API the implementation of the digital processing signal algorithm can be carried out in "OCTAVE", a suitable processing language for this task. On the other hand, it enables the implementation of the user interface in JAVA, which is the most appropriate programming language for this kind of task. By dividing the process into the separate algorithm and visualisation parts, the applications can be designed more quickly, the “joPAS” API being the element enabling this link to be carried out.

9. ACKNOWLEDGEMENTS

The authors of this paper would like to thank the students, E. Arroyo, F. Becerra, X. Eguiluz and E. Zubiaur, for providing us with feedback on their “joPAS” API experiences, as they are using “joPAS” in the simulation of digital systems part of their projects. The project consists of designing an application for the graphic programming of dsPIC digital signal microcontrollers [7]. We would also like to express our gratitude to Sourceforge for letting us house the "joPAS" API in their servers, so that everybody can download it from the URL http://jopas.sourceforge.net, should they wish to do so.

REFERENCES

[1] B. García, J. Vicente, I. Ruiz, A. Alonso, E. Loyo, "Regeneration Model for Esophageal Voices" in *Proc.* BIOMED 2005, Innsbruck, Austria, 2005.
[2] B. García, J. Vicente, I. Ruiz, A. Alonso, E. Loyo, "Esophageal Voices: Glottal Flow Regeneration" in *Proc.* ICASSP ‘05, Philadelphia, EEUU, 2005.
[3] B. García, J. Vicente, I. Ruiz, A. Alonso, E. Loyo, "Noise Reduction Algorithm for Esophageal Voices" in *Proc.* IWSSIP’04, Poznan, Polonia, 2004.
[4] B. García, J. Vicente, "Adaptative Pitch Scaling Algorithm for Esophageal Speech" in *Proc.* BioSignal 2004, Brno, Czech Republic, 2004.
[5] Kurt Hornik, Friedrich Leisch, Achim Zeileis, "Ten Years of Octave Recent Developments and Plans for the Future" in *Proc.* DSC 2003, Vienna, Austria, 2004.
[6] Ken Arnold and James Gosling, " The Java Programming Language" The Java Series. Addison-Wesley, Reading, MA, 1996.
[7] J, Angulo, B. García, J, Vicente, I. Angulo, Microcontroladores avanzados dsPIC", International Thomson Editors, 2005.