INTERACTIVE DSP EDUCATION USING JAVA

Y. Cheneval[*], *L. Balmelli*[*], *P. Prandoni*[*], *J. Kovacevic*[†], *M. Vetterli*[*]

[*] = LCAV-EPFL, CH-1015 Lausanne, Switzerland

[†] = Bell Labs, Lucent Technologies, Murray Hill, NJ

ABSTRACT

In this paper, we argue that Java is a natural language to develop interactive teaching material that can be shared and distributed widely. Unlike any other programming language or platform we know, Java development is justified because of its almost universal acceptance. We develop a Block Diagram (BD) based approach that allows to develop interactive and downloadable signal processing laboratories. As an example, we show how specific experiments for a DSP class, as well as for an advanced course on wavelets have been developed.

The article first explains why the Java language has been chosen, and then describes what has been realized today. Finally, we show how the BD representation can be efficiently used for the development of a wavelet theory course. It is shown that only a few simple blocks are sufficient for creating many didactic programs. This can be seen as an *a posteriori* justification of the BD model.

1. Platform and programming language for Web-based teaching

1.1 State-of-the-art

In the context of Computer Based Education (CBE), the Web has been mostly used as a method of disseminating static information or as a support medium to traditional books. This is due mainly to the lack of a Web programming standard for designing truly interactive courses. The arrival of the Java language has allowed to design programs (*applets*) that can be executed on the student's computer. Many Java applets have been developed that demonstrate standard DSP algorithms [4]. In these applets, the interaction with the user is based on previously existing Matlab (or other packages) demos [6][8] (the student is using sliders to modify parameters and buttons to perform some actions), and therefore represents mostly converting code to Java.

Although this interaction mode is easy to use and represents what a researcher needs, it has several drawbacks. From the educational standpoint, the student does not know in detail how the algorithm works since he is seeing only input and output data. In this case, the algorithm is like a black box, whose behavior can be determined by monitoring input and output data. From the programming standpoint, there is an obvious advantage in writing code in Java directly; most algorithms can be decomposed in several phases; some may be reusable, like for instance all the I/O part and the graphical output that is similar for many algorithms. Moreover, these tasks are usually the most time-consuming and the least interesting to develop.

1.2 The Block Diagram model

In this article, we propose a model based on a Block Diagram (BD) representation. In this model, the algorithm is described as a set of blocks joined together, where basically the output of a block is linked to the input of the next.

The system allows to describe a BD-based algorithm that represents the flow of data through operators. An algorithm is decomposed into basic blocks (e.g. first get the data, then take an FFT and finally display the spectrum). The applet is created by building its BD model. When run, each block is highlighted as the data flows through it, so that the student knows what is taking place at every moment. The student can interactively place probes on the block diagram to monitor any signal coming out of any block.

A typical algorithm is depicted in Figure 1. Input data is fetched or generated, then the algorithm is run (processing). Many iterations can be necessary to reach the correct result and the user may change input data for computing a new result.

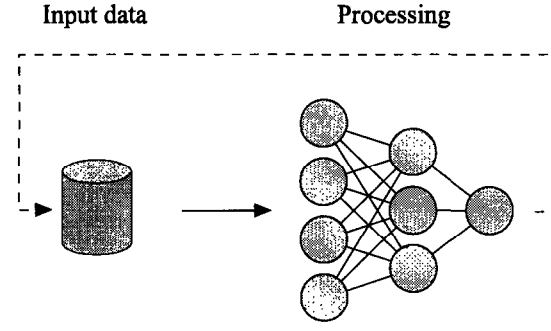

Figure 1: Typical setup for a data processing algorithm running in a loop

Figure 2 shows the BD representation of an algorithm based on the abstract setup of Figure 1. This representation uses the data flow model.

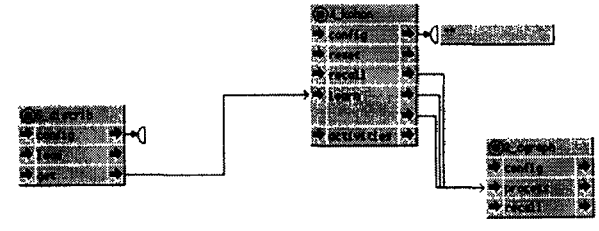

Figure 2: BD model for a data processing algorithm

Each box represents a component that possesses a processing capability. In this example, the leftmost box generates data for the middle box, whose results are displayed by the rightmost box. The data passes through the wires connecting the components.

The BD diagram is directly available so that the user can display and interact with it. For example, the user can attach probes to any wire to monitor the data that passes through it.

1.3 Suitability of the Block Diagram model

This system addresses the two drawbacks we mentioned above. First, the student can have a better understanding of the inner parts of the algorithm by immediately viewing how it has been implemented (In Figure 2 for instance, the middle component could be "opened" to give an inside view of the inner workings of the algorithm). It gives him the freedom to view intermediate signals that would not be available on the standard applet display. Second, the decomposition of the algorithm in block diagram allows to code each block as a separate entity. Different algorithms may then be created by assembling components that have already been developed for another algorithm. For example, graphical output and I/O can be easily reused.

Previous work [2] has shown that this model is well suited to data processing. Moreover, experience (derived from other programming languages) has shown that the interesting part (the algorithm) is taking approximately 15% of the total development time, whereas 85% is spent to develop the Graphical User Interface (GUI), data display and I/O parts. It is therefore clear that providing tools to take care of the 85% of the code is a great advantage for minimizing applications development time.

1.4 The resources mechanism

The BD model provides excellent reusability by clearly splitting the algorithm into its different parts. Since each block must be often customized for different uses (e.g. sampling rate changes for a downsampler block), we need a mechanism to easily modify internal variables inside each block. This can be achieved using resources. The idea is to allow the designer to export (publish) some variables that can be accessed from outside the module. A resource interactor can then be used to manipulate the resources (Figure 3).

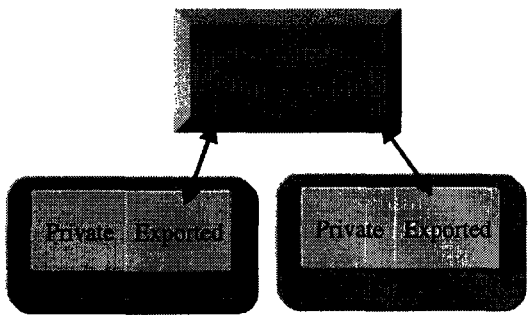

Figure 3: Resource interactor

The resources provide a flexible way of exporting parameters. However, modification of resources must be handled properly, so that the module can take appropriate actions when the value has been modified. We have established the following protocol for resource modification:

Suppose that a resource has been modified in the resource interactor. Once the user validates the change:

a) The resource interactor asks the owner of the resource if the change is permitted, proposing the new value.

b) The owner examines the change. It can veto the change for various reasons (e.g. value out of range). If it is vetoed, the change is not allowed. If it is allowed, the change is applied and the owner can take the appropriate steps to validate the change (e.g. recompute a formula that includes the modified value).

Experience has shown that these mechanisms can be implemented in a very efficient way. The process of adding or removing resources is simple from a programming point of view.

1.5 Features of Java in a CBE framework

There are advantages and disadvantages to using Java for teaching purposes. Here are the most important ones:

a) Java (coupled with HTML) allows us to have an *interactive* and dynamic presentation running on the host computer, therefore diminishing time-consuming connections between the server and the host which would prevent the necessary interactivity.

b) Since the application is running locally, it uses the local processing power. Most computers have enough power today to run the kind of applications that are needed in a reasonable amount of time. Moreover, this local vs. centralized computation model is very useful in the case of a full classroom performing the same tasks at the same time. In such a case, a centralized computing resource would be a bottleneck for performance.

c) The machine independence feature of Java allows us to develop the application once and run it on all Java-compatible platforms (in practice, any platform such as Unix, Macintosh or MS Windows).

d) The disadvantages of Java are mostly due to the youth of the language. Security and performance problems are the ones that matter the most. These problems are being looked into by the designers of the Java language and should be solved in the near future.

2. Project status

Based on the BD model (including the resource mechanism) and an existing implementation developed with the C language [2], a set of basic primitives has been developed. These primitives can be separated broadly into: a) handling of data formats and I/O, b) graphical data display and c) DSP operators. They form the core from which we will write the block diagrams.

2.1 Data formats and I/O

Numerous possibilities exist to generate or acquire a signal (from a range, random data, list, file, ...). We have developed a `Signal` class that represents a discrete signal on which it is possible to apply standard functions to create complex input, like in Listing 1:

```
Signal   v1 = new Signal(),
         v2 = new Signal();
  v1.range(0,6.28/40,6.28+(6.28/40));
```

```
v2.load("Dauchechies4T.data");
```

Listing 1: Working with the Signal class

We can also use the IEEE SP Signal Database [5]. Data can be directly loaded as an input signal by any applet that is using our set of components. Resulting data can be stored back in the database if needed.

2.2 Graphical Data Display

In order to display computation results, a `Figure` class has been developed. This class can handle the display of many data types, such as 1D, 2D and 3D functions and images. Automatic or constrained axis handling is possible, as well as adding a title or text annotation. Using the overloaded `plot` method, the user can display any supported data type. We show the result of iterations of the 4-tap Dauchechies filter in Figure 4. The menu (Zoom Level) on the bottom of the figure is a context menu corresponding to the data type passed to the plot function.

All events (e.g. interactive zoom selection, mouse clicks) are handled internally in the class and are completely transparent to the user. Some routines for creating buttons, text areas, ... to coontrol the resources of the algorithm are also available.

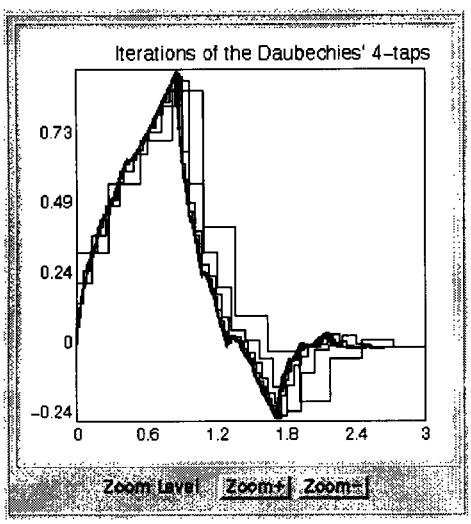

Figure 4: Iterations of the 4-tap Daubechies filter

2.3 DSP Operators

Inspired by the environment provided in Matlab, we have decided to use a class hierarchy of DSP operators as close as possible (in terms of semantic and nomenclature) to the one used in Matlab.

As a result, the bottom level regroups the basic operations such as convolution, rate operator, FFT. Higher level operators (one-dimensional and two-dimensional N-channel filter banks and wavelet transforms) are based on it (Figure 5). For example, the code to iterate the Daubechies 4-tap filter would be:

```
public class Algorithm extends Operator {
  ...
  public Signal compute( int n )
    { // Compute n iterations of the filter
    Signal Daubechies4T = new Signal();
    Signal IteratedFilter = new Signal();

    Daubechies4T.data(0.4830,0.8365, 0.2241,-0.1294);
    IteratedFilter = Daubechies4T.copy();

    for (int i = 0; i < n; i++) {
      IteratedFilter = upsample(IteratedFilter, 2);
      IteratedFilter = conv(Daubechies4T,
                            IteratedFilter);
    }

    return IteratedFilter;
    }
}
```

Listing 2: Java code describing iterations of a 4-tap Daubechies filter

The main program will declare instances of algorithms and drive their interactions or sequence of executions. The goal of this class architecture is to separate the development of DSP code and GUI. The user will progressively build a set of algorithms of his own and use them as Java code external to the DSP environment.

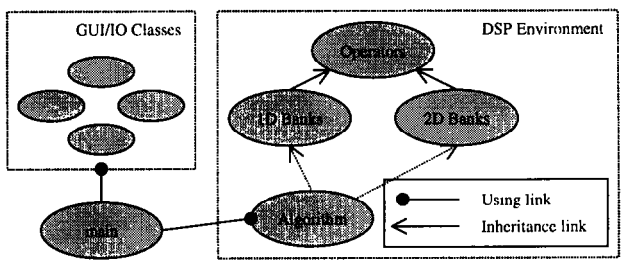

Figure 5: Example of object hierarchy

Based on these classes, some applets have already been written. They are grouped in an interactive DSP book [7] available online. Selected examples include the Gibbs phenomenon, an echo cancellation (Figure 6) and a 2D Wavelet Transform applets.

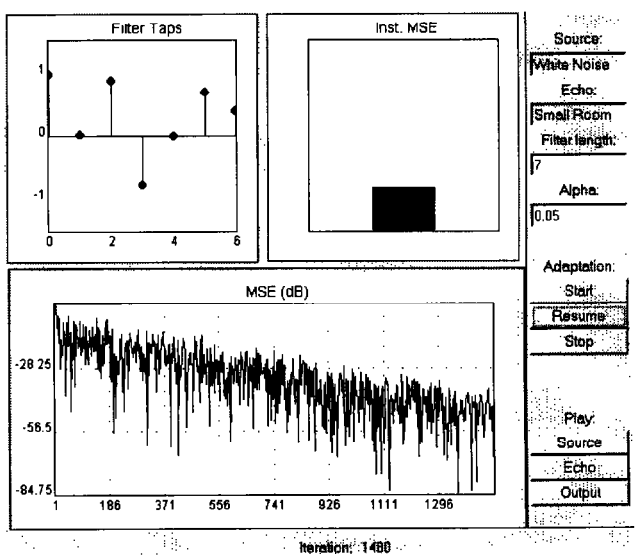

Figure 6: The echo cancellation applet

The student can change the different parameters. A theoretical explanation of each phenomenon is available in the Web documents.

3. Interactive course on wavelets and subband coding

3.1 The course

We now illustrate the BD concepts using examples from a wavelet course we are developing. For this course, 12 chapters have been chosen, each one featuring an introduction text in HTML plus one (or more) didactic applet(s) that demonstrates the subject in practice. Each applet is designed using our block diagram decomposition system, thus allowing the student to understand and interact with all the blocks that are part of the applet. Here is a list of the applets that are being developed.

1. Laplacian and Gaussian image pyramids
2. Filter banks, orthogonal, non-orthogonal, linear-phase
3. Regularity & 2-scale equation wavelets
4. Surface Approximation and computer graphics
5. Approximation theory: piecewise polynomial functions
6. Audio compression, segmentation and Musicam
7. Wavelet compression of images and video
8. Wavelet denoising of images
9. Communications: Multitone modulation/embedded QAM
10. Best bases and time frequency representations
11. Multiresolution motion estimation
12. Quantization in frames

This course is being designed for graduate students and interactive usage. As explained earlier, the basic assumption is that many features of the 12 applets will be the same: They will all need to display signals (1D and 2D such as images), load/save input signals (I/O), use a GUI, react to events generated by the user (e.g. press of a mouse button), ... All the applets are designed using the BD model described above. We must therefore identify the key components that are needed for this task.

3.2 The key components

To understand the basic concepts from the theory and application of wavelets and subband coding [9], we need a few basic primitives from multi-rate signal processing, namely:

a) convolution generators (filters)
b) downsamplers
c) upsamplers

These blocks need certain parameters (e.g. impulse response for convolution, sampling rate changes). Using these three blocks, any filter can be computed, that is, scaling functions and wavelets can be obtained.

For compression examples, we need:

a) standard linear transforms (e.g. DCT, DWT)
b) quantizers (e.g. dead zone scalar quantizers)
c) entropy coders (e.g. Huffman codes).

In addition to these elementary processing blocks, we need probes that can display time and frequency views of the data being processed. The above blocks are mostly implemented. For a current state of the project, see: http://lcavwww.epfl.ch/javaproject/

4. Conclusion & Perspectives

The BD model is a very flexible tool that can be further expanded to provide a fully cooperative model where each block could be directly fetched from the server in the network. The BD model has not yet been completely implemented in Java although a fully functional model is existing in C. Some applets and basic blocks do exist already. The goal is to have a set of blocks generic enough so that they can be combined to easily create different algorithms.

One further goal of having a complete set of Java applets for a specific topic such as "wavelets and subband coding" is to implement what is called "reproducible research" [3][1]. In that framework, a research paper has to be complemented by all the experimental data used in producing the results. The reader of the paper can actually rerun all experiments to verify the results, but also try the methods on his own data. The problem is now very similar to interactive teaching: one needs downloadable and portable software and thus, Java is a natural choice. An ambitious but worthwhile aim is that availability of a critical mass of applets will potentially allow reproducible research to be implemented relatively easily.

It is to be noted that certain communities are close to presenting their results in a form that allows easy comparison (e.g. the discrete optimization community has large sets of standard problems that are used as benchmarks).

While signal processing is moving towards better benchmarking, it is still a fact that many compression papers describe results that are difficult to reproduce by another researcher. It is our claim that such a state of affairs actually slows progress in a given field. Therefore, a potential benefit of our Java platform is to create an environment where reproducible research can become a reality.

5. REFERENCES

[1] Buckheit J., Donoho D., *WaveLab and Reproducible Research* , To Appear, Wavelets and Statistics, Anestis Antoniadis, ed. Springer-Verlag Lecture Notes, 1995.

[2] Cheneval Y., *PackLIB, un environnement de développement modulaire pour la création visuelle d'applications conviviales*, PhD dissertation, LAMI-EPFL, 1997.

[3] Claerbout J., *Hypertext Document about Reproducible Research*, http://sepwww.stanford.edu/.

[4] Guo H., Odeguard J. E., Burrus C. S., *Teaching Wavelets with Java on the Information Superhighway*, http://www-dsp.rice.edu/edu/wavelet/.

[5] IEEE SPIB Database, http://spib.rice.edu/spib.html.

[6] McClellan J., Schafer R., *Using Multimedia to Teach the Theory of Digital Multimedia Signals*, To appear, IEEE Tr. On Education, 1997.

[7] Prandoni P., Balmelli L., *An Interactive DSP book*, http://lcavwww.epfl.ch/~prandoni/dspbook.html.

[8] Rahkila M., Karjalainen M., *An Interactive DSP Tutorial on the Web*, ICASSP97 V3, p. 2253.

[9] Vetterli M., Kovacevic J., *Wavelets and subband coding*, Prentice-Hall, 1995.