A Philosophy and Software Tool For Teaching and Learning Signal Processing and Communication Systems

Jake Gunther
Dept. Elec. & Comp. Eng.
Utah State University
Logan, Utah 84322-4120
jake.gunther@usu.edu

Brandon Eames
Dept. Elec. & Comp. Eng.
Utah State University
Logan, Utah 84322-4120
beames@engineering.usu.edu

Darin Nelson
Dept. Elec. & Comp. Eng.
Utah State University
Logan, Utah 84322-4120
dsnelson@cc.usu.edu

***Abstract*—Graphical modeling software tools have become commonplace in educational laboratories associated with courses in signal processing and communication systems. This paper explains that the purely graphical approach to modeling and simulation can be improved upon by combining graphical modeling with some low-level programming tasks. To implement this philosophy, a new software tool called EduCOM was developed. EduCOM offers a graphical modeling environment to construct system models. It also translates a graphical model into C/C++ code that can be compiled into an executable to simulate the model. Students are required to program the functionality of important blocks in their models such as filters, minimum-distance hard-decision rules, and look-up-tables. Anecdotal evidence is provided indicating that this approach offers an improved learning experience over pure graphical modeling approaches.**

***Index Terms*—Signal processing education, engineering education, simulation software**

I. Introduction

Over the last ten to fifteen years, we have witnessed increasing use of software tools and computer modeling and simulation in electrical engineering education. Two of the disciplines most affected by this trend are signal processing and communication systems. Most classes in these areas involve computers in some way. Assignments range from simple programming assignments in a high-level language that run on a personal computer to complex projects that run in real-time on DSP hardware.

Besides preparing students for real engineering in the workplace, computer assignments serve practical pedagogical purposes. Programming assignments create a new paradigm in which the student plays the role of the teacher with the computer as student. Students must write a specific set of instructions for the computer to follow. If the student can teach the computer to perform a task, then the student must have mastered the concept as well. Otherwise, he could not teach it correctly.

To aid working engineers, many software vendors have developed graphical tools for modeling, simulating and analyzing signal processing systems. Examples include Simulink [1, 2], LabVIEW [3,4], SystemVue [5,6] and JDSP [7,8]. These tools are promoted by promising to enhance the performance of engineers. At a high level, they work like this. An engineer builds a graphical model of a signal processing system by dragging blocks from a predefined library into a modeling area on the screen using a mouse. The blocks implement specific functions such as filtering, up sampling, down sampling, addition, multiplication, etc. The blocks are connected by drawing wires between the blocks. In appearance, the software-based graphical model resembles quite closely the block diagrams that engineers typically use to describe systems. After configuring the individual blocks and specifying some global parameters, the engineer starts a simulation by pressing a button on the graphical modeling tool. This is billed as a performance enhancement because the engineer did not have to write a single line of code. Graphical models can be assembled and simulated quickly and easily.

Besides gaining acceptance in industry, graphical modeling tools have encroached on academic turf in the research laboratory, the classroom, and the instructional laboratory. Today graphical modeling tools are commonplace in classes on signal processing and communication systems. In some instances, and the number seems to be growing, modeling and simulation using graphical tools has supplanted more traditional means of simulating, i.e. writing computer programs in low-level programming languages such as C/C++.

In an educational setting, graphical modeling offers the same performance enhancements that it offers to practicing engineers. Students can assemble and simulate complicated signal processing systems quickly. They can vary system parameters and change input signals easily and visualize the corresponding responses. Thus, graphical modeling offers students the chance for experimentation that was not formerly available.

Besides the advantages of experimentation, graphical modeling is a natural extension of the theory developed in the classroom. In the field of communication systems, for example, teachers often pose a problem to students in the classroom. Then, using the fundamental principles students learned previously, teachers lead them through the steps of a

process to find, say, the optimal receiver for a given transmitted waveform and channel. In the end, all the mathematics of the receiver processing is converted into a block diagram representation. This is as far as the classroom discussion can go. Therefore, it seems natural to use the block diagram as the starting point for a laboratory exercise, and graphical modeling provides a means to construct a model for the receiver and to simulate it. These simulations serve to verify the performance characterization developed in class through mathematical analysis.

Having enumerated the strengths of using graphical modeling in educational settings, we believe that a purely graphical approach to modeling and simulation separates students from some of the low-level details that are an important component for education. In the past students did all their modeling and simulation using a programming language like C. This exercise forced them to think deeply about the low-level details of the system. In contrast to graphical modeling, programming is a slow, tedious, error prone, and often frustrating process. Nevertheless, it offers a process for learning about the details and function of system components that is lost when graphical modeling replaces computer programming.

In our view, the ideal modeling and simulation tool would combine the best features of graphical modeling and low-level programming. The ideal tool would use graphical modeling to construct models of signal processing systems because this can be done quickly and graphical models agree with block diagram representations developed in the classroom. The ideal tool would also require students to write code in a language like C to implement the functionality of the blocks in the system model. To test this philosophy, we developed a tool called EduCOM for education in communication systems. EduCOM is the ideal tool described above. This paper describes this tool and how it may be used in an undergraduate course on communication systems.

II. The EduCOM Tool

The EduCOM modeling and simulation tool was developed using the Generic Modeling Environment. This section first explains the Generic Modeling Environment and then describes the EduCOM tool.

A. Generic Modeling Environment

The Generic Modeling Environment (GME) [9–11] is a toolkit for doing three things:

1) **creating** graphical modeling languages for specific domains,
2) **using** graphical modeling languages to compose models, and
3) **translating** a graphical model into a form suitable for simulation.

Block diagrams are ubiquitous in the areas of signal processing and communication systems. Such diagrams represent signal flow from inputs to outputs. The blocks in the system identify the processing performed on intermediate signals. For example, Fig. 1 shows a block diagram for a QPSK transmitter/receiver system.

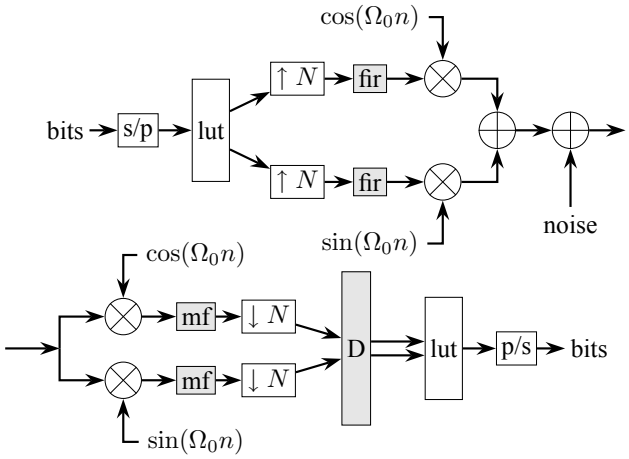

Fig. 1. Block diagram of the QPSK system implemented and simulated using EduCOM in a laboratory exercise. Students were required to implement the core functionality of the shaded blocks in C, while implementations for the remaining blocks were provided by the teacher. In the diagram: "lut" stands for look-up-table, "s/p" and "p/s" are serial-to-parallel and parallel-to-serial operations, "D" is a hard decision block, "fir" is a finite impulse response filter, and "mf" is a matched filter.

Signal processing block diagrams have a language in and of themselves. They have syntax, semantics, and present information. By following the rules of the language, an author (the creator of the diagram) can communicate information to readers (those who examine the diagram).

GME is a toolkit for creating graphical modeling languages. Recently, the authors used GME to create a graphical language to describe signal processing block diagrams. GME can enforce the constraints and rules governing the construction of block diagram models. For example, the output from one block may be connected to the input of another block and the sample rates should match. It does not make sense to connect two inputs together or two outputs together. All of the information relating to a specific graphical modeling language constitutes a modeling paradigm and is stored in a paradigm file.

GME provides a graphical editor for model construction. To use a particular modeling language, the paradigm file for that language is loaded into GME. Then the modeler may construct models using the language. Models are created using a mouse to drive a graphical model construction process in a model editing window.

While GME can be used to create graphical models according to a specific modeling paradigm, it natively offers no support for analyzing or extracting the information captured in the models. Rather, it offers an intuitive interface whereon tool developers can build translation tools, called model interpreters. For example, a model interpreter might translate a graphical model into C code. The C code could be compiled to build an executable file, which can be executed, perhaps, to simulate the system. The interpreter is usually written by the

same people who created the modeling paradigm.

In summary, GME can be used by tool developers to create new graphical modeling languages and model interpreters. System developers can then employ GME as a model development platform supporting the created modeling language. System modelers construct graphical models of a system, and then invoke the corresponding model interpreter to perform meaningful translations on the models, such as code generation.

B. EduCOM

The EduCOM language was created as a GME modeling paradigm designed specifically for students learning about communication systems. It implements the essential features needed for constructing rather general signal processing systems. A screen shot of the GME/EduCOM modeling window is shown in Fig. 2.

EduCOM consists of three kinds of blocks: source blocks, sink blocks, and processing blocks. Source blocks are the inputs to a signal processing system. The source blocks currently implemented in EduCOM are: input from file, constant, sinusoid, random integer, and random Gaussian noise. Sink blocks are destinations for signals. The sink blocks currently implemented in EduCOM are: output to file, eye pattern, and scatter plot. Because EduCOM was designed for simulating communication systems and because eye patterns and scatter plots are useful diagnostic tools, these sink blocks were provided. However, it would be easy to add other blocks for visualization in time or frequency domains.

Processing blocks perform transformations on their input signals. The processing blocks currently implemented in EduCOM are: sum block, product block, gain block (scale the input by a constant), filter, look-up-table, hard decision, serial-to-parallel, parallel-to-serial, up sample, and down sample. Obviously, some of these blocks are geared specifically for communication systems. However, many of these blocks, such as filter, up sample, and down sample, are useful in general for (multirate) signal processing systems.

Besides blocks, EduCOM has two other language elements: ports and connections. Ports come in basically two types: inputs and outputs. There are inverting and non-inverting input ports. Inverting ports can be used to change a sum block to a difference block, for example. In addition, there are also named ports that associate particular input signals with variables in the C code generated by the model interpreter.

Connections are represented in EduCOM by a line between an input port and an output port. Connections are drawn using a simple, intuitive point-and-click process. GME takes care of intelligently routing lines for connections around blocks so that the graphical models stay uncluttered and easy to read.

The EduCOM interpreter is a C program that translates graphical models into C/C++ code and is invoked by pressing a button on the GME modeling window. The interpreter iterates through the model and instantiates one C++ block object for each block in the model and one C++ connection object for each connection. A reference to the connection object is passed

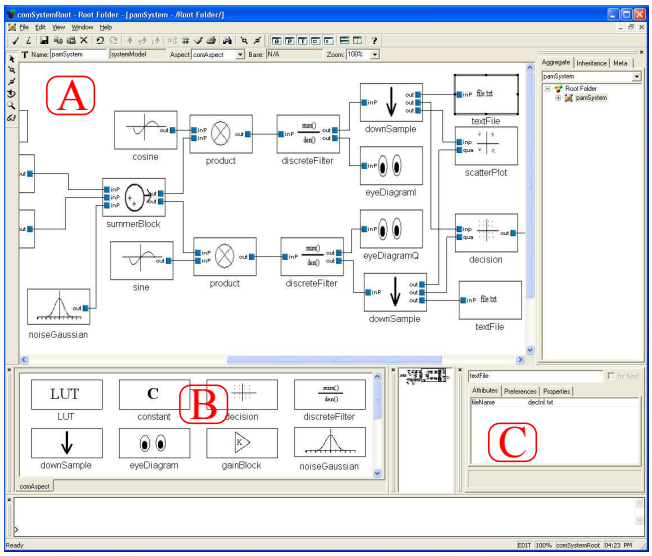

Fig. 2. Screen shot of the GME/EduCOM graphical user interface. A partially built model is shown in the main modeling area (A). The block pallette (B) appears below and to the left of the main modeling area. The pane below and to the right of the main modeling area shows block attributes (C). The solid squares on each block are ports, and lines represent connections.

to the block that outputs to the connection and the block that takes inputs from the connection.

Each block class in EduCOM inherits from a base class. The key feature of the base class is a method called "fire". The fire method is called to invoke the processing of the block. For example, the fire method in a filter block implements convolution, i.e. it takes an input, shifts it into a shift register, multiplies the data in the shift register with the filter coefficients, adds the products, and outputs the result. Each block has a fire method.

The last step for the EduCOM interpreter is to write the simulation main loop. Through a process known as topological enumeration [12], the interpreter analyzes the model and determines the proper order to call the fire methods for the blocks in the system. It then places calls to the appropriate fire methods in a for loop. The duration of the simulation is easily controlled by changing the termination condition for the for loop. While EduCOM writes the simulation main loop, it does not write code for the fire methods. These are written either by the student or by the teacher. The next section explains this.

III. Modeling and Simulation Process Using EduCOM

The goal of this research was to test the hypothesis that a combination of graphical modeling and low-level programming provides a better learning experience than a purely graphical approach. To test this idea, EduCOM was developed. The main feature of EduCOM that is different from other graphical modeling tools (Simulink, LabVIEW, SystemVue, JDSP) and that makes testing our hypothesis possible is that EduCOM does not provide code implementations for the blocks, i.e. the fire methods. Like other graphical modeling

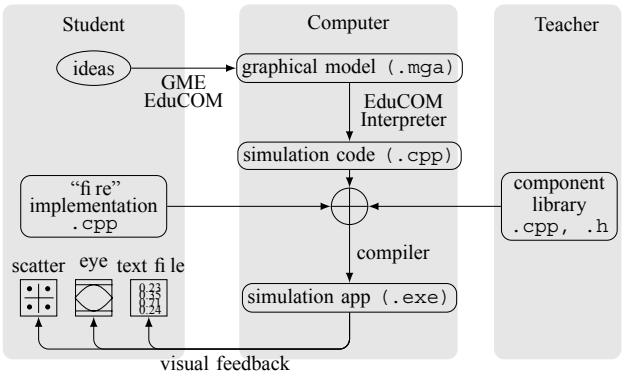

Fig. 3. EduCOM communication system model development and simulation process.

tools, EduCOM provides a library of blocks to choose from in constructing models. Other graphical modeling tools, provide implementations of blocks. EduCOM does not. The block implementations are provided by either teacher or student, and with out the block implementations the model will not simulate. Therefore, EduCOM offers the teacher the flexibility to vary the student experience. The teacher could provide fire method code for every block. Then, EduCOM would be no different from any other graphical modeling tool. At the other extreme, the teacher could provide nothing and require students to write the fire methods for every block in the system. We believe that some middle ground may be best because there seems to be little to be gained, from an educational standpoint, in requiring students to implement the fire methods for simple blocks such as sum, multiply, etc. Instead, it seems more productive to require students to focus their attention on what may be called the "high value" blocks in the system. In the case of communication systems, the high value blocks are the matched filter, the hard decision block, and the look-up-table. These are operations that are discussed at length in classroom lectures and investigated in homework exercises.

The model development and simulation process of EduCOM is illustrated in Fig. 3. Students come into the computer lab with ideas about a system model block diagram. Using GME and the EduCOM language they build a graphical model. The EduCOM interpreter translates the model into C/C++ code. Students provide fire method implementations for the high value blocks. Teachers provide fire method implementations for the pedestrian blocks. Together the student's code, the teacher's code and the simulation main loop provided by EduCOM, students build an executable and run it. The executable provides visual feedback in the form of eye patterns, scatter plots, and text files so that students can debug errors. When their systems are in working order, students can experiment with variations in modulation type, pulse shape and excess bandwidth, signal to noise ratio, carrier frequency and phase offsets, and so on. This experimentation provides valuable insights into the purpose and function of the components in the system.

IV. Observations From a Pilot Study

EduCOM was used in a pilot study with students in an undergraduate communication systems class. In this study, students built the QPSK system shown in Fig. 1 in Simulink and EduCOM. This group of students were also experienced with the C/C++ programming languages and could therefore also offer their opinions about the three approaches to modeling and simulation: pure graphical approach using Simulink, pure programming approach using C/C++, combination of graphical modeling and programming using EduCOM.

In this experiment using EduCOM, we saw something new that we had not previously encountered when using Simulink in our teaching. The first part of the lab exercise with EduCOM was to construct a graphical model for the QPSK system. As in our previous experience with Simulink, students quickly constructed their models. The next step in a Simulink lab is to press the "simulate" button. With EduCOM, the next step was to invoke the interpreter and then write code implementations of the fire methods for the filter block, the hard decision block, and the look-up-table block. (These high value blocks are shaded in Fig. 1.) At this point, students backed away from their computers and opened their notes and their textbooks to review concepts. They thought hard about the filtering and decision concepts. They got out paper and pencil and worked things out by hand before returning to the computer to write code. The requirement to program the fire methods for these blocks forced the students to think more deeply about what these blocks were doing and to review concepts from class. In Simulink, students utilized pre-built blocks from a block library, but they did not actually know what the blocks were doing. Therefore, with EduCOM, we were able to recover an important process for learning that was lost when we used Simulink. Further assessment of the EduCOM tool is underway.

References

[1] www.mathworks.com.
[2] S. T. Karris, *Signals and Systems with Matlab Computing and Simulink Modeling*. Orchard Publications, 3e ed., 2007.
[3] www.ni.com/labview.
[4] M. A. Yoder and B. A. Black, "Teaching DSP First wtih LabVIEW," in *Signal Processing Education Workshop*, pp. 278–280, IEEE, 2006.
[5] eesof.tm.agilent.com/products/systemvue.
[6] D. Silage, *Digital Communication Systems Using SystemVue*. DaVinci engineering Press, 2006.
[7] jdsp.asu.edu/jdsp.html.
[8] A. Spanias and V. Atti, "Interactive online undergraduate laboratories using J-DSP," *IEEE Transactions on Education*, vol. 48, pp. 735–749, Nov. 2005.
[9] G. Karsai, M. Maroti, A. Ledeczi, J. Gray, and J. Sztipanovits, "Composition and cloning in modeling and meta-modeling," *IEEE Transactions on Control Systems Technology*, vol. 12, pp. 263–278, Mar. 2004.
[10] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai, "omposing domain-specifi c design environments," *Computer*, vol. 34, pp. 44–+, Nov. 2001.
[11] www.isis.vanderbilt.edu/Projects/gme.
[12] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2nd ed., 2003.