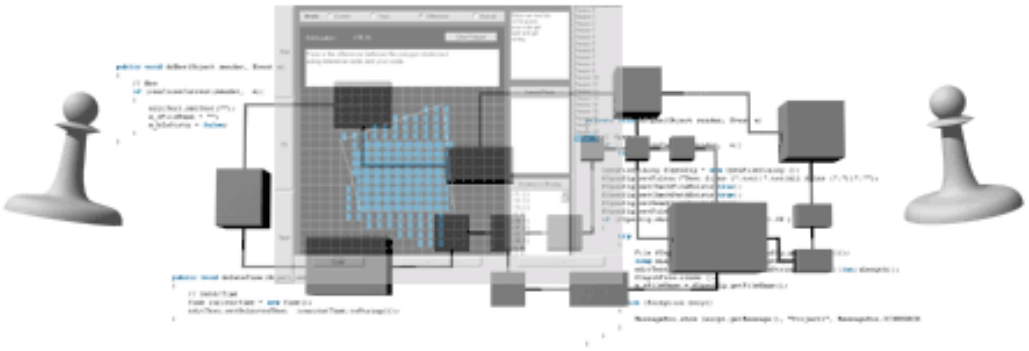
# Educational Fusion

## Implementing a Production Quality User Interface With JFC



**Kevin Kennedy**

**Prof. Seth Teller**

**6.199 May 1999**

## Abstract

Educational Fusion is a online algorithmic teaching program implemented in Java. In has been in the process of development for the last few years. When the system was initially started the only available UI production classes were those provided in Java's Abstract Windowing Toolkit. As the system has evolved, multiple developers have worked on different portions of the interface, each with their own style and conventions. Recently, Sun released the Java Foundation Classes which include a rich set of UI development tools called Swing. At the same time, Fusion has matured to a state where it should have a production quality user interface.

## 1. Overview of the System

Educational Fusion is a system for developing and utilizing advanced interactive educational algorithmic visualizations on the World Wide Web. Educational Fusion allows educators to create interactive visualizations which require students to use problem-solving skills to implement solutions. The students can then immediately visualize their algorithm's output, and compare it to that of a hidden reference algorithm. This is an improvement over other educational systems which are limited to semi-interactive visualizations and simple question-and-answer forms. There are four main components to the Fusion system: the concept graph, the editor, the visualization panels, and the collaboration tools.

### 1.1 Concept Graph

The concept graph is a two dimensional grid that contains algorithmic components represented as blocks which are connected together to create a more complex algorithm. These modules can be either functional components with well-defined inputs and outputs or another lower level grouping of functional components.

In most computer science classes, a typical exercise asks the student to contribute small portions of code to a large system. Often times, the student feels frustrated by having to make sense of hundreds of lines of code while only learning a small portion of the system. The concept graph provides a way for a student to get a feel for the entire system without reading all of the source code. With a feel for the overall architecture of the given system, the student can then write code for particular components with specified inputs and outputs.

### 1.2 Editor

The Fusion editor is a fully mature software development editor. It is written entirely in Java and has all of the features that one would expect from such an editor. These include parenthesis matching, auto indenting,

and keyword highlighting. There will soon be an option to switch between Windows and Emacs key bindings. Once the code for a particular component has been written, the student presses the compile button and the code is sent to the server where it is automatically compiled. At this point, the student may visualize the functionality or continue working elsewhere.

### 1.3 Visualization

Every component in the concept graph may have an associated visualization panel. There is a wizard that allows a developer to create new components. A developer may also create an applet to visualize the code working. There are four modes of operation in a visualization panel. The first mode allows the student to see the desired behavior of the algorithm without looking at any code. Second, the student may view the behavior of his or her compiled code. The third mode shows the difference between the student's code and the actual algorithm. Finally, the student may use manual mode to play around with any ideas when considering the algorithm.

### 1.4 Collaboration

Educational Fusion also has a collaboration system that allows students, teachers, and administrators to interact in useful ways over a network. In some respects, the collaboration component of the Fusion system is a chat program with dynamically created rooms and discussions. Students can initiate discussions with their friends or they can participate in global chat environments. There is a help queue and a database that stores all messages so the contents of a discussion will be available at a later time.

## 2. Work Completed Prior To This Term

So far while working on the Educational Fusion team, my main work has been on the collaboration tools. When I began, the system had a global chat window and instant messaging capabilities. Realizing that this was not sufficient for our goals, I constructed a new collaboration system that allowed for multiple methods of interaction. The new system consists of three main concepts: groups, discussion and the help queue.

### 2.1 Groups

Groups are basically aliases for a group of people. This is all the functionality that they have. If you send a message to a group, only those people in the group that are currently logged on will receive the message. One special group called the Global List is a list of everyone who has ever logged onto the system.

### 2.2 Discussions

Discussions are a very robust chat dialog. They can be used in various ways. The special discussion called Global Discussion can be thought of as a chat room for everyone logged onto the system. A user can create a small discussion with his or her friends to use whenever they log in. The system will store the entire history of the discussion. Another use of a discussion is to treat it like a newsgroup. A user can create a public discussion with a name describing the topic and himself or herself as the only recipient. Then other people can go into the discussion, look at the history, and post a response. Basically, discussions can be used however one might want to communicate with others.

### 2.3 Help Queue

The queue is a means for allowing students to engage administrators in a help discussion. A student begins by posting a question to the queue. Once this is accomplished, the student is presented with information about the number of students ahead in line and the estimated time until an administrator is available. At the same time, administrators may view the new question through their special panel. They can choose to take a student and begin a dialog.

## 3. Overview of Swing

Swing is a powerful set of UI component classes developed by Sun Microsystems for use with the Java programming language. When Java was first released, Sun provided the Abstract Windowing Toolkit (AWT) for creating user interfaces. Unfortunately AWT has a number of shortcomings. First, the variety of components is lacking. For example, AWT does not provide any means to create tables or trees. Another problem is that AWT relies on the client system to determine how to render the components. Although this was supposed to lead to a more natural cross-platform functionality, inconsistencies between various virtual machines create a number of problems. To address these and other issues, Sun created the Java Foundation Classes (JFC) which include an entire suite of classes to create better user interfaces. These libraries were called Swing.

### 3.1 Lightweight Components

AWT relies on native C code on each platform to render UI components. Swing components, on the other hand are written entirely in Java. Thus, they use methods that draw lines, display text, handle events, etc. This means that they expect the virtual machines on different platforms or in different browsers only to correctly interpret the Java code. Obviously this allows for greater flexibility and customization. Another advantage of lightweight components is that a Java program can have complete control over events that occur with a particular UI object. For example, AWT provides an event called `MOUSE_MOVED` that

provides mouse screen coordinates so that a user's cursor motions may be tracked. However, in AWT when the cursor moves over a component such as a button or a list, the cursor is effectively lost since there is no way to catch the event while the mouse is over a component displayed by native C code. Swing overcomes these difficulties.

### 3.2 Plugable Look and Feel

Another important feature of the JFC Swing packages is that the components support a plugable look and feel (PL&F) architecture. This means that the developer my choose a particular look and feel to use when displaying the interface. The default look and feel was produced by JavaSoft and is a cross-platform look and feel in that it is displayed the same regardless of the platform. Code-named Metal, the default look and feel specifies styles and behaviors for the various components. This architecture gives developers great freedom in the applications they develop since they may choose to use the default PL&F, a PL&F that acts like a particular platform (such as the Macintosh PL&F also developed by JavaSoft), a PL&F they design, or a modification of a particular PL&F.

## 4. Changes Made to the System

While working on this project, I made a number of changes to the Fusion system. Most notably, I began changing the UI from an AWT based interface to a Swing interface. In the process, I created a number of Fusion specific classes that modify the Swing components and abstract away from some of their initialization. This allows for a consistent user interface across the entire Fusion system.

### 4.1 Description of the Changes

I chose the collaboration tools to be the first part of the Fusion System to be converted to a Swing UI. There were a couple of reasons for this choice: I was familiar with that section of the code base, the tools are UI intensive without much need for additional graphics such as drawing polygons, and they can benefit by components available in Swing but not in AWT such as tables. Overall, the conversion was a success. I chose to use the default look and feel since it is the most refined and provides a truly cross-platform approach. However, I chose to modify certain aspects of some of the components to create what I thought was a more appealing look.

Most Swing component classes such as the `JButton` class follow the basic functionality and API of their analogous AWT component and usually have the same class name with the "`J`" in front. To make a consistent interface that does not use all of the defaults given by the Metal look and feel, I chose to extend most of the swing components by subclassing them and assigning certain attributes. As an example, the `JLabel` component renders a lavender label in bold Arial font. I decided that the black would be a better

choice for the labels. Therefore, I created a class called `FLabel` as Shown in Figure 1. All of the constructors for this class mirror the constructors for the superclass `JLabel`. This is necessary so that all cases are covered.

```
public class FLabel extends JLabel
{
    public FLabel(Icon image, int horizontalAlignment)
        {super(image, horizontalAlignment); initialize();}

    public FLabel(Icon image)
        {super(image); initialize();}

    public FLabel()
        {super(); initialize();}

    public FLabel(String text, Icon icon, int horizontalAlignment)
        {super(text, icon, horizontalAlignment); initialize();}

    public FLabel(String text, int horizontalAlignment)
        {super(text, horizontalAlignment); initialize();}

    public FLabel(String text)
        {super(text); initialize();}

    private void initialize()
        setForeground(Color.black);
}
```

Figure 1. Code for a typical Swing component subclass.

Basically, this class is composed of a bunch of constructors which parallel the set of constructors for the parent class `JLabel` and an initializer which sets the color of the font. Although this may seem like a relatively useless class, it is actually quite powerful. If an application has a few hundred labels and suddenly the developer decides that he no longer wants a bold Arial font and if `FLabel` has been used in every instance, then all that it takes is to add another line to the `initialize()` method of `FLabel`. There are a number of additional benefits for this approach. First, it saves time for the developer since some components such as a text area take a number of lines of initialization which are the same in virtually every case. Second, it gives the developer an elegant way to add new functionality to UI components since it is possible to add as many additional methods to the subclass as desired. Third, a particular Swing component such as `JLabel` may have multiple subclasses. For example, it may be the developer's desire to have three different styles of labels. Thus, there could be three different subclasses: `FLabelBig`, `FLabelSmall`, and `FLabelNormal`; each having different initializations. Fourth, new components can be designed by specializing particular Swing components. One such object I created is a `FButtonRow` which subclasses `JPanel`. The motivation for this was that every time a multiple number of equal width buttons need to be displayed in a row, they need to be put in a `JPanel` with a `GridLayout` layout manager. `FButtonRow` simply takes in an array of `JButton`s and returns a component which is the panel containing them neatly laid out.

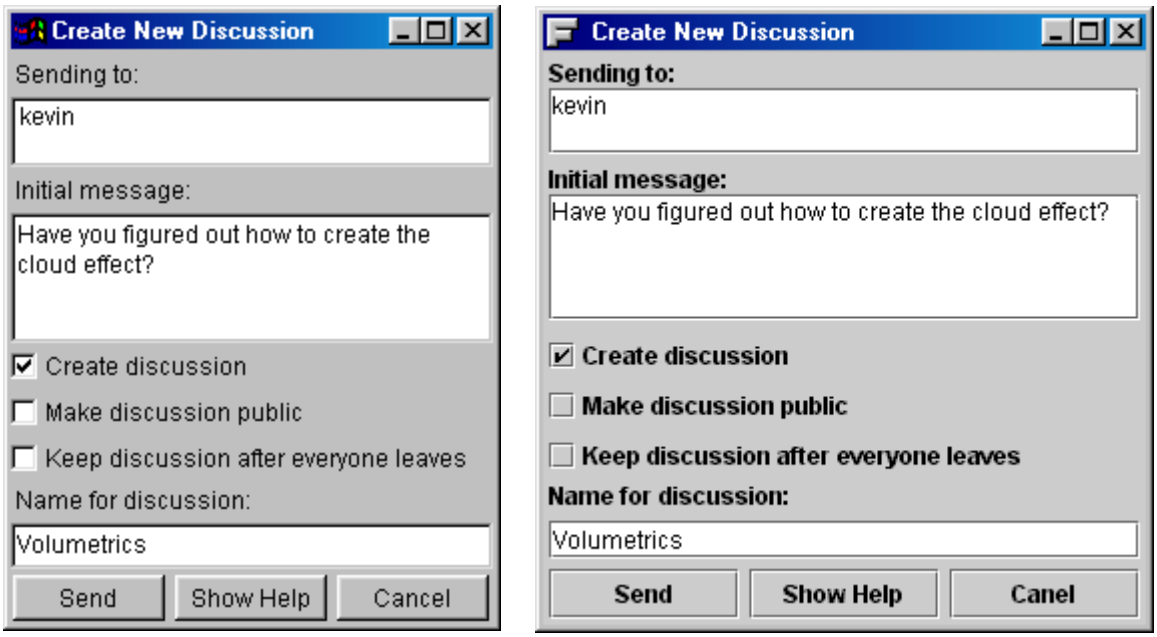


Figure 1. Visual appearance of a pop-up window in the old and new interface.

Using the approach described above in addition to some layout conventions I designed in order to make the UI consistent, I began changing the Fusion interface from a AWT based design to a Swing based design. At this point, I have completely renovated the collaboration tools to a one hundred percent Swing design. The images in Figure 2 above show the visual differences between an old pop-up window and a new pop-up window.

### 4.2 Benefits of the Changes

In addition to the ideas mentioned above, there are a number of additional benefits. In the past, we had a lot of trouble with UI components displaying incorrectly on different platforms and on different browsers. Because Swing is written entirely in Java, it basically solves all of these problems. The change to Swing has also provided us with numerous new UI components such as tables, borders, and tool tips. In the past, we had to write custom code whenever we wanted to achieve something of that nature. Earlier, it was mentioned that with AWT, mouse movements are lost when the mouse moves over a component such as a button. One of the technologies we are developing for the Fusion system is a way for two users to share the same screen. This is based on the idea of network transparent events that are sent back and forth so each user will have their own cursor with which to work. However, this breaks down if the cursor is frequently lost over UI components. Swing also solves this problem because of the lightweight nature of its components. Finally, one could argue that the default look and feel is more aesthetically pleasing than some browsers' rendering of the old AWT interface.

## 5. Additional Work

In addition to converting the collaboration tools to a Swing based interface and creating a suite of classes which subclass Swing components, I also did some investigative development of an entirely new paradigm for Fusion's high level UI architecture. The basis for restructuring Fusion's UI architecture is that Swing provides new functionality that did not exist with AWT and the old Fusion interface did conform to the typical style of a piece of software.

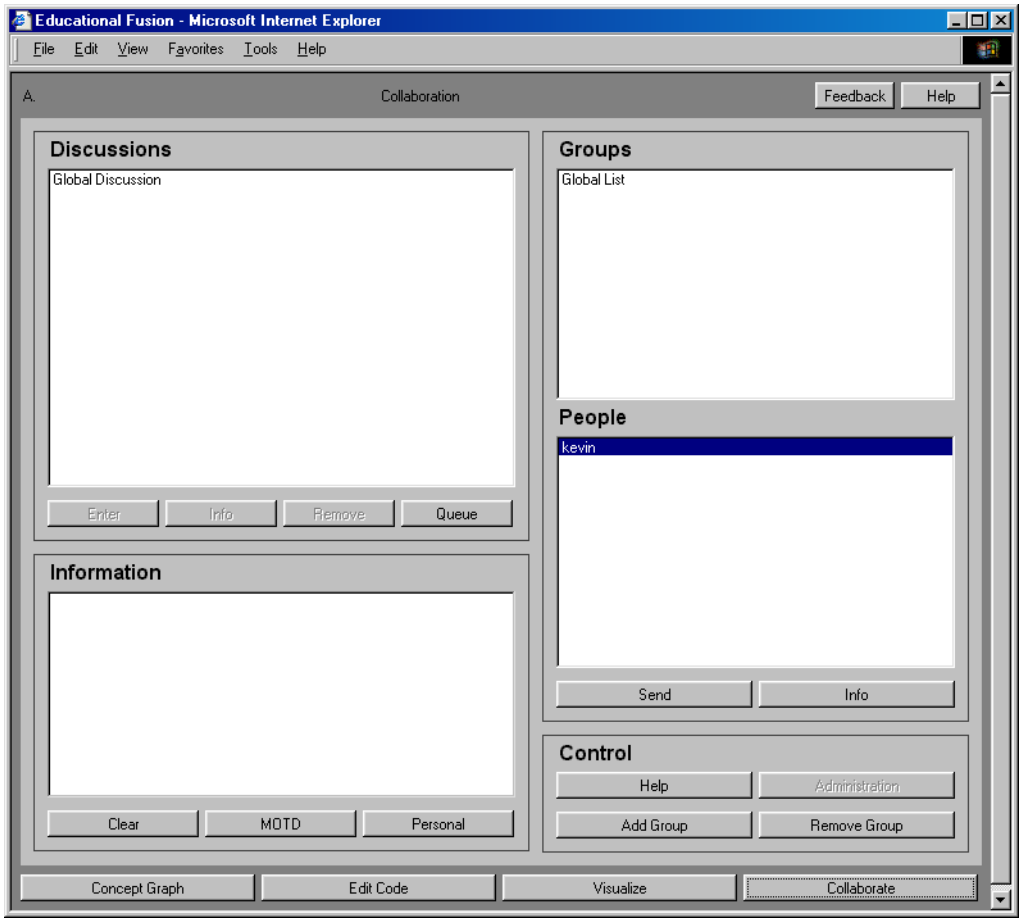


Figure 3. The old interface to Fusion.

The new components of particular interest are menubars, toolbars, and internal frames. Although AWT did have classes for creating toolbars, the ones in swing are far more powerful and allow for images and shortcuts. Toolbars are new to Swing and provide a nice way of making common actions as easy as one click. The most important new component provided by Swing is the internal frame. This is a window which is contained within another window. Fusion consists of four main screens: the concept graph, the editor, the visualization panel, and the collaboration panel. With internal frames the user has the choice of working with these separate screens in different internal frames or working with them maximized so he or she can simply switch between then as they could before. Figures 3 and 4 show the difference between the old and

new high level approaches to the Fusion interface. When viewing these images, note that the old version is being displayed within a browser, and the new one is being displayed as its own window.

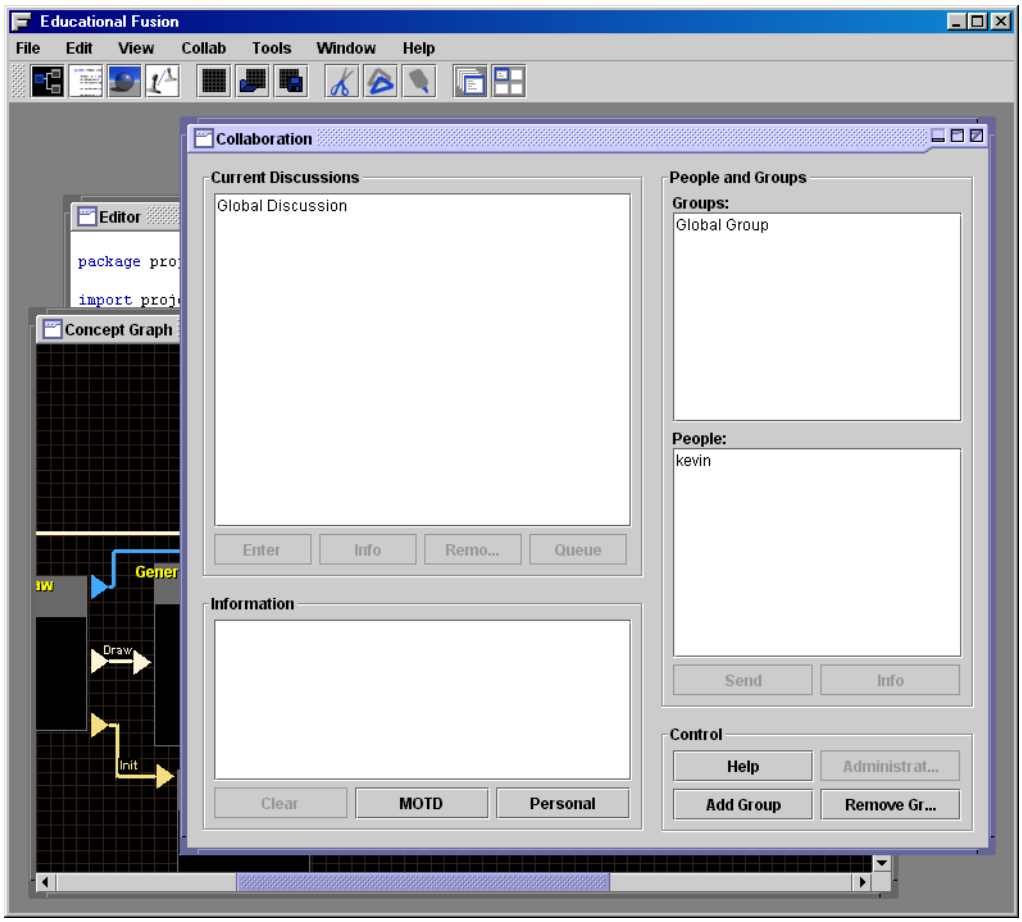


Figure 4. The new interface to Fusion.

## 6. Future Work

Currently, the new high level architecture can't be fully tied into the rest of the system. In order for this to work, the other three panels need to be turned into pure Swing interfaces. This should not be that difficult since I have already created useful classes and layout constants. Also, the concept graph and editor have significantly fewer UI components which should make their translation fairly simple. The only problem is the visualization panel since a large part of the UI that is displayed on the panel is created by third-party developers. Unfortunately, it doesn't make sense to impose the necessity of Swing on these developers. The main problem is that the heavyweight AWT components are rendered by native C code which means that they always appear on top of any Swing components. This is due to the fact that Java has absolutely no control over the way different virtual machines render these components. The most obvious solution to this problem is to have the visualization panel open in a pop-up window. Another possibility would be that while the visualization panel is displayed, it must be on the top level and menus will be disabled. Perhaps further research may turn up another possibility.