JAZ: JAVA ALGORITHM VISUALIZER. A MULTI-PLATFORM COLLABORATIVE TOOL FOR TEACHING GRAPH ALGORITHMS

Giancarlo Bongiovanni
Dipartimento di Scienze dell'Informazione
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, 00139 Roma, Italy
bongio@dsi.uniroma1.it

Pierluigi Crescenzi
Dipartimento di Sistemi ed Informatica
Università degli Studi di Firenze
Via Cesare Lombroso 6/17, 50134 Firenze, Italy
piluc@dsi2.dsi.unifi.it

Gabriella Rago
Dipartimento di Scienze dell'Informazione
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, 00139 Roma, Italy
rago@dsi.uniroma1.it

Abstract

In this paper we propose a new graph algorithm visualizer, called JAZ (Java Algorithm visualiZer), whose main features are: (a) JAZ is basically machine-independent; (b) JAZ is a post-mortem SV system; (c) JAZ heavily uses colors and it allows to show multiple views of program executions; (d) JAZ is based on an imperative approach; (e) the interaction between the user and the system is based on the Java Application Programming Interface; (f) the main purpose of JAZ is novice classroom demonstration and algorithm development and debugging. In our opinion, JAZ is very simple to be used. The main characteristic of JAZ, however, is that it can also be used as a multi-platform distributed collaborative tool for teaching graph algorithms (over any network supporting the TCP protocol, such as Internet or any intranet).

Keywords: Graph algorithms; Algorithm animation; Visualization.

1 Introduction

The last years have seen remarkable improvements in the development of frameworks for the so-called area of *algorithm visualization*, which is understood to be the visualization of a high-level description of a piece of software. It is well known to people who develop algorithms that their brain (or, better, its right side [14]) automatically does an imaginative work, consisting in creating mental images that correspond to the main characteristics of the different phases of analysis, development and implementation of an algorithm. Therefore, as it is explained in [18, 19], automatic visualization can become a useful aid for teaching as well as for developing and understanding algorithms (both in the design phase and in the analysis and development ones). A survey on the more general subject of *software visualization* (in short, SV) can be found in [21], where twelve visualization systems (that is, SOS [3], BALSA [9], ZEUS [7], TANGO [24], ANIM [6], PASCAL GENIE [12], UWPI [17], SEE [4], TPM [15], PAVANE [22], LOGOMEDIA [16] and CENTERLINE OBJECTCENTER [11]) are briefly described and compared with respect to six main categories which can be summarized as follows: (a) the *scope* (that is, the kind of hardware, operating system, language and applications that can be handled by a given SV system), (b) the *content* (that is, what subset of information is visualized by the SV system and whether the visualization is produced as a batch job (*post-mortem*) or as the program executes (*live*)), (c) the *form* (that is, the medium and the graphical objects by means of which a visualization is specified), (d) the *method* (that is, the style in which a visualization is specified (procedural or declarative)), (e) the type of *interaction* existing between the user and the SV system, and (f) the *effectiveness* of the system.

We have noticed that the main visualization packages, such as ZEUS, TANGO and PAVANE, suffer of

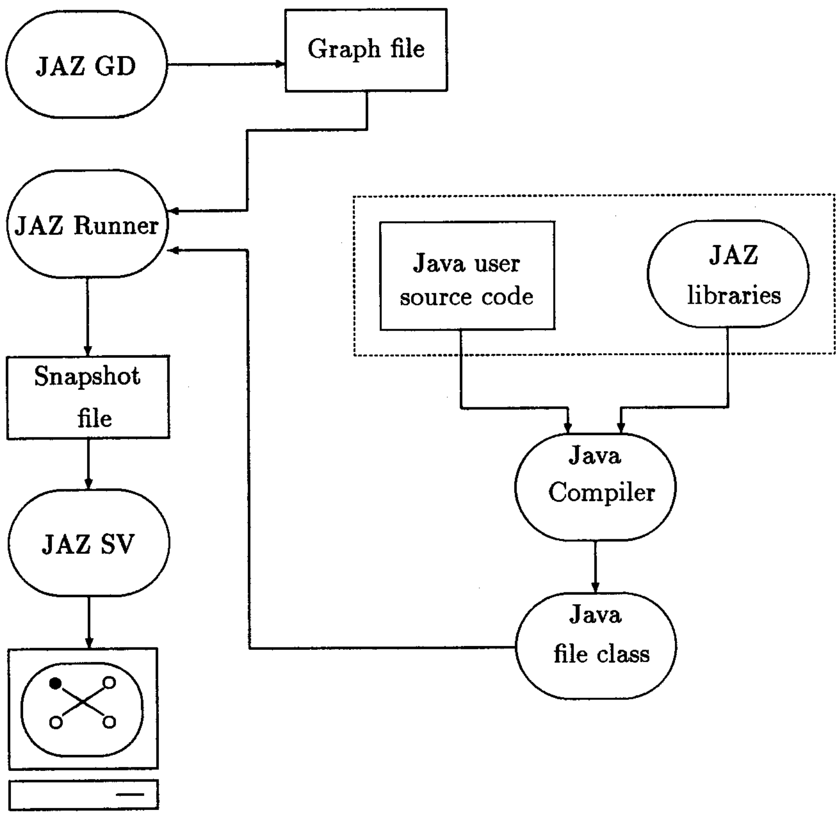


Figure 1: Software architecture of the JAZ SV system

two main drawbacks: they are machine-dependent since they make use of machine-dependent graphic libraries and, being general-purpose algorithm animation systems, they require a very high learning effort for a person who want to visualize a given algorithm. In this paper we propose a new algorithm visualizer, called JAZ (Java Algorithm visualiZer), that overcomes the first drawback, being developed in Java language [2], and is very simple to be used, having restricted the class of algorithms to be animated to the class of graph algorithms.

According to the six categories described above, the main features of JAZ are the following:

1. Being written in Java, JAZ is basically machine-independent. Indeed, the only requirement is that a Java virtual machine is available on the machine and the operating system on which JAZ has to be executed. So far, JAZ has been tested on the following operating systems: MAC-OS, Windows 95, and Unix.

2. JAZ is a post-mortem SV system.

3. JAZ heavily uses colors. Moreover, it allows to show multiple views of program executions (thus allowing, for instance, to compare different algorithms for the same problem).

4. JAZ is based on an imperative approach, that is, the data structures that can be used for the visualization are constrained by the functions contained in the JAZ Library.

5. The interaction between the user and the system is based on the Java Application Programming Interface.

6. The main purpose of JAZ is novice classroom demonstration and algorithm development and debugging. In our opinion, JAZ is very simple to be used.

The main characteristic of JAZ, however, is that, similarly to other SV systems such as GASP-II [23], MOCHA [5], and CAT [8], it can also be used as a *multi-platform distributed collaborative tool for teach-*

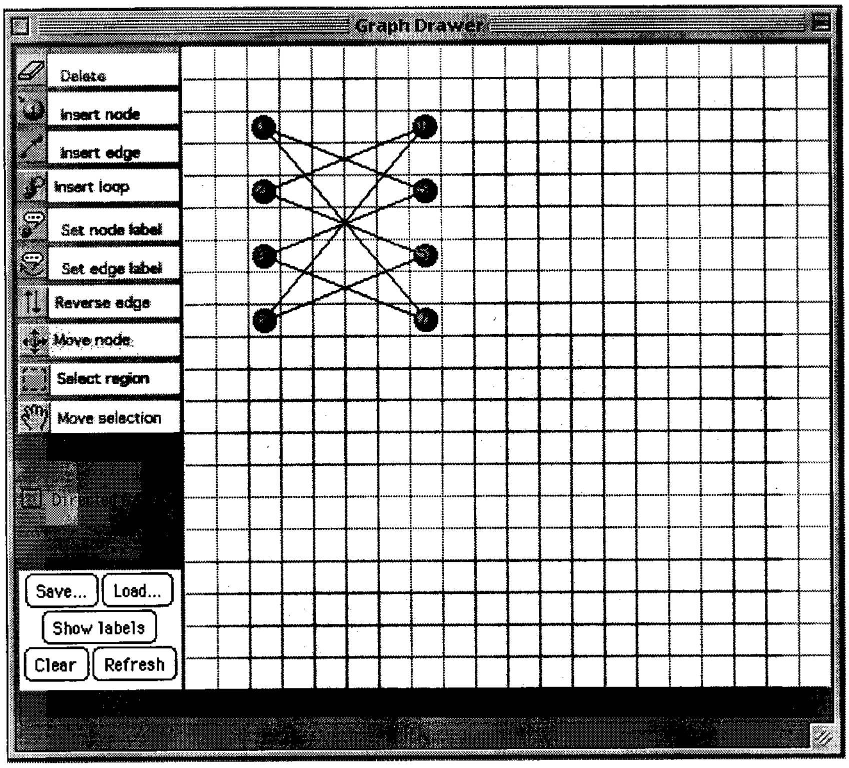


Figure 2: The graphic interface for the JAZ graph drawer

ing graph algorithms (over any network supporting the TCP protocol, such as Internet or any intranet).

The paper is structured as follows. In Section 2 we briefly describe the philosophy behind JAZ and its software architecture when viewed as a single-user SV system. In Section 3, instead, we describe the network deployment of JAZ when used as a multi-platform collaborative tool for teaching algorithms and how JAZ can be used as a teaching tool in an electronic classroom. In Section 4, finally, some directions for future research are briefly outlined.

2 The JAZ SV System

JAZ is a Java SV system which allows to visualize a sequence of slides intended to represent the execution of a graph algorithm. It uses a post-mortem technique for making animations out of the execution of a graph algorithm, similar to the one used by GAIGS [1, 20]. In other words, the visualization can be displayed only after the program executed.

The software architecture of the JAZ SV system is shown in Fig. 1. The user who wants to animate a graph algorithm can use the *JAZ Graph Drawer* to draw a graph and save it into a file with a specific format. Then, in order to load the graph and create a file representing snapshots of the execution of any graph algorithm, the user has to annotate the program, by inserting at interesting points of the source code some Java methods, contained in a set of *JAZ Libraries* (to this aim, the inclusion of these libraries in the source code is needed). This set of libraries contains methods that allow a Java programmer to load a graph, to modify its structure (by either adding or deleting nodes and edges or changing the color of its nodes and edges), to take pictures of the graph at interesting points of the execution of the source code, and to save these slides into a file with a specific format. The code, compiled with any Java compiler, will produce a class file. Therefore, the user can call the *JAZ Runner* that takes this class and the graph file as input and produces as output a file containing an animation of the algorithm. At this point, the user can execute the *JAZ Snapshot Visualizer*, which reads the animation file, interpret it and displays the snapshots representing the execution of the program.

The graph drawer The graph drawer allows the user to either load a graph or draw a new one. To this aim, a simple and almost auto-explaining interface has been implemented (see Fig. 2). At the present

```java
import java.lang.*;
import java.awt.*;
import java.io.*;
import java.util.*;
import jaz.graph.*;
import jaz.files.*;

public class DFSApplication implements AlgInterface{
  boolean[] n;
  boolean BT = false;
  JazAlgGraph g = new JazAlgGraph();
  JazAlgFiles files = new JazAlgFiles();
  PrintStream outputStream = null;

  public DFSApplication() {}

  public void algStart() {
    g.loadGraph();
    outputStream = files.openShowfile();
    g.initNodeColors(Color.black, Color.white);
    files.snapshot("Initial graph", g, outputStream);
    n = new boolean[g.computeNumberOfNodes()];
    for (int j= 0; j < n.length; j++) n[j]=false;
    DFSearch(0);
    files.closeShowfile(outputStream);
  }

  public void DFSearch(int i) {
    n[i] = true;
    g.setColorsOfNodeOfIndex(i, Color.black, Color.green);
    files.snapshot("Visit node"+i, g, outputStream);
    g.setColorsOfNodeOfIndex(i, Color.green, Color.black);
    for (int j = 0; j < n.length; j++) {
        if ((g.existsEdgeBetweenIndices(i, j)) && (n[j] == false)) {
          g.setColorOfEdgeBetweenIndices(i, j, Color.green);
          DFSearch(j);
          if (BT) {
            g.setColorsOfNodeOfIndex(i, Color.black, Color.green);
            files.snapshot("Backtrack to node"+i, g, outputStream);
            g.setColorsOfNodeOfIndex(i, Color.green, Color.black);
            BT = false;
          }
        }
    }
    BT = true;
  }
}
```

Figure 3: The code for the DFS problem

time, the graph drawer is quite basic, especially if compared to more powerful specific software. The interface is composed of three panels: a *palette* panel, a *command* panel and a *drawing* panel. By clicking with the mouse on any of the buttons of the palette panel, the user can perform on the drawing panel one of the possible drawing actions (such as deleting a node or an edge, inserting a node, an edge, or a loop, setting a node or an edge label, reversing an edge direction, moving a node, and selecting and moving a set of nodes). After a graph has been drawn, the user can save it, by clicking on the `Save...` button of the command panel. In the current version of the graph drawer, it is not possible to edit several graphs simultaneously on the same drawing panel neither to perform any kind of graph combination. Moreover, the node or edge label can only be a `float` value. Finally, no graph drawing algorithms are available even though they will be included in the near future.

The libraries JAZ offers a set of Java libraries, that the user must import in the source code in order to animate an algorithm. An example of a Java program, annotated with some JAZ primitives (the ones written in bold), that implements a depth-first search algorithm, is shown in Fig. 3. In order to produce a file of snapshots representing the graph at different times of the execution of the program, the standard `main()` method is substituted by the **`algStart()`** one, that loads the graph and opens an outputstream, where the snapshots will be printed. This method also initializes the colors of the nodes of the graph and takes an initial snapshot of the graph by means of the **`files.snapshot()`** method that causes the description of the graph at that moment to be printed on **`outputStream`**.

The kernel of the program is the recursive method `DFSearch()` which receives as parameter the current visited node `i`. The first visualization step within this method consists of coloring node `i` with green in background and black in foreground by means of the method **`g.setColorsOfNodeOfIndex()`**. A snapshot of the graph is then taken. Successively, when the edge between `i` and `j` is selected, this edge is colored with green by means of the method **`g.setColorOfEdgeBetweenIndices()`** and, once again, a snapshot of the graph is taken. The last visualization step consists of changing again the background and the foreground colors of node `i` in order to emphasize that this node has been already visited.

The snapshot visualizer The JAZ snapshot visualizer is the heart of the toolkit. Its interface is shown in Fig. 4 where different steps of the execution of the program previously described are presented. After having loaded a file of snapshots (through the `Load` button), the user can either show the slides one-by-one both in forward and in backward mode (by clicking on the `Step` or on the `Back` button) or visualize them (by clicking on the `Go` button) until the end of the execution of the program is reached or until the `Pause` button is clicked. Notice that a snapshot description is composed of a text which is visualized in the text area positioned in the upper part of the interface and of a graph specification which is visualized immediately below the text area. The meaning of the `Broadcast` button and of the lower part of the interface (which supports a chatline) will be explained in the next section.

3 The JAZ Collaborative Tool

The network deployment of the JAZ collaborative tool is shown in Fig. 5. Basically, the interaction between the teacher and the students is allowed by means of a server which broadcasts any incoming

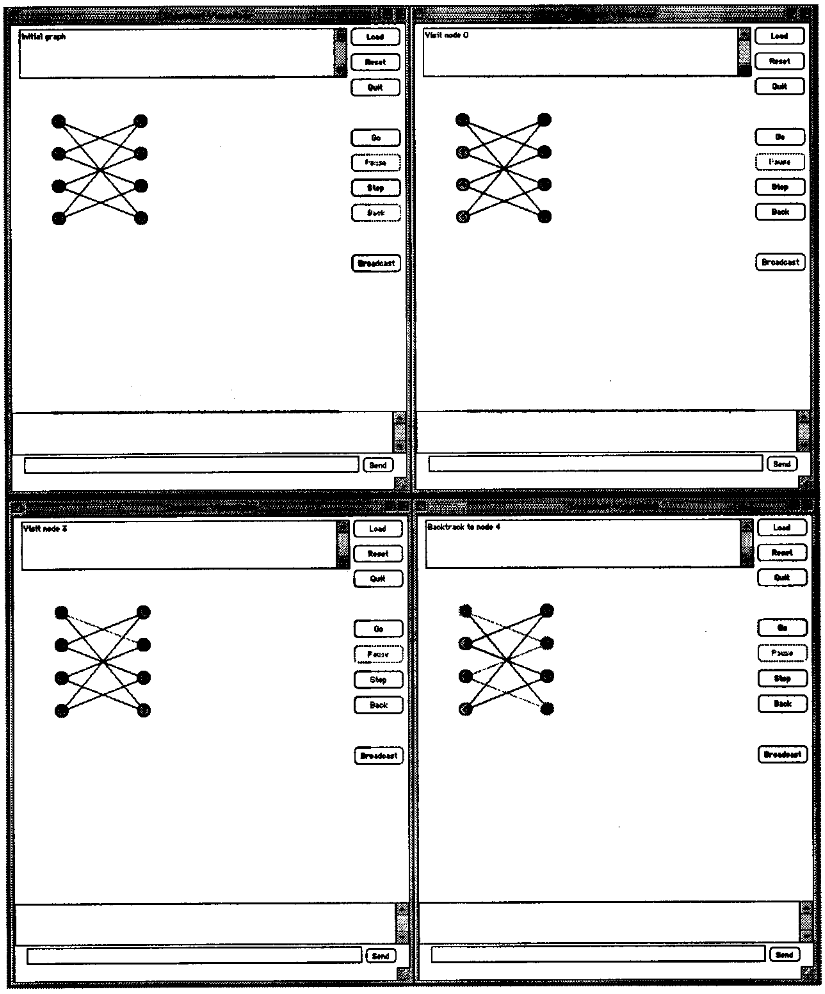


Figure 4: Four screen dumps of the execution of the Jaz Visualizer

message to all connected clients (i.e. the teacher and the connected students). This server is part of the teacher tool: it communicates with the teacher interface through a pipe connection and with the student tools through TCP connections (thus, there are no limits on the "geographic dimension" of the virtual classroom). Since the messages traveling through the network can be either snapshot descriptions or chatline messages, all connections need a multiplexor and a de-multiplexor in order to correctly handle two distinct message streams traveling on a single connection. Observe that, in the current version of JAZ, the visualization of the snapshots is not interactive: that is, the student cannot interfere with the sequence of the visualized snapshots.

The teacher interface The teacher snapshot visualizer (see lower part of Fig. 6) is exactly the same as that described in the previous section. The `Broadcast` button allows the teacher to send the description of the currently visualized snapshot to all connected students (observe that only the teacher can send snapshot descriptions). The lower part of the interface regards the chatline. This part is composed of (a) a text area where all messages traveling through the network are visualized, (b) a text field where a new message can be composed, and (c) a `Send` button that allows to send the message contained in the text field.

The student tool The student interface is similar to the teacher one (see upper part of Fig. 6). There are two main differences: (a) the student interface does not contain the visualization buttons (indeed, the student can only receive snapshot descriptions from the broadcast server), and (b) the interface contains a connection part where the name of the student has to be specified and the connection can be estab-

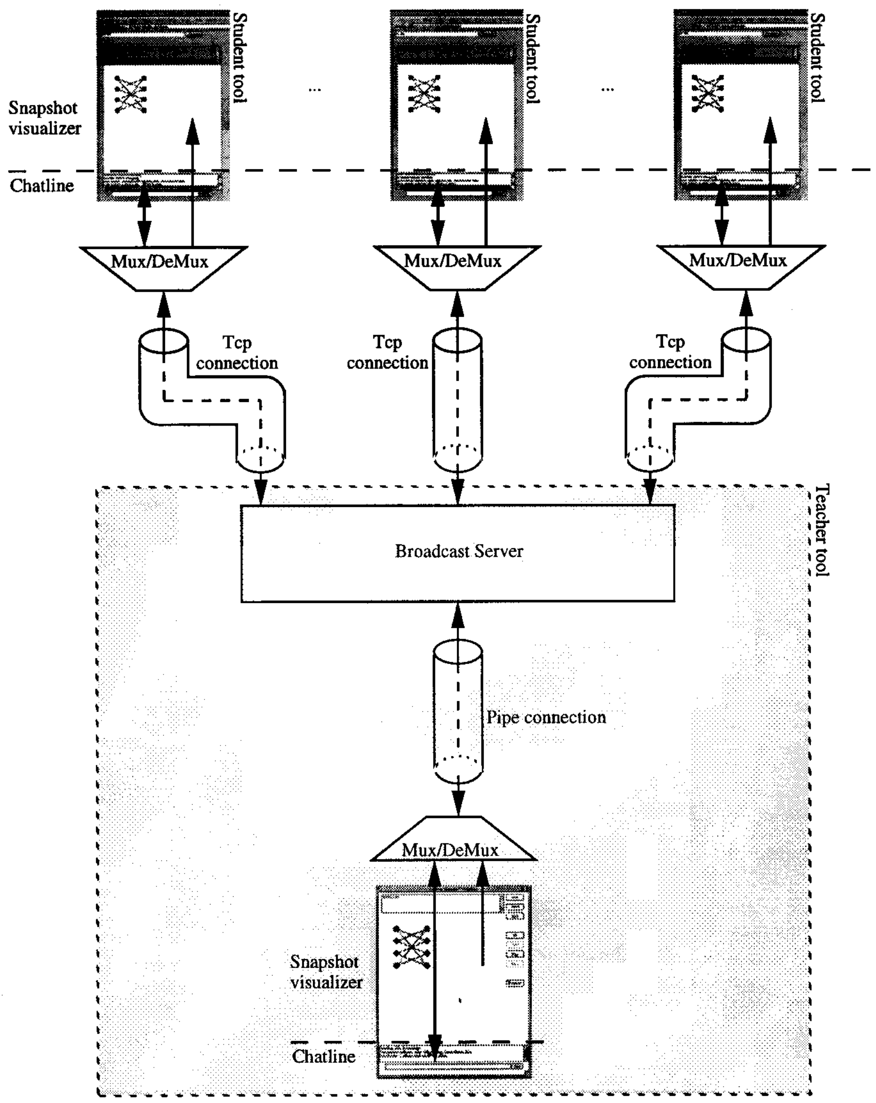


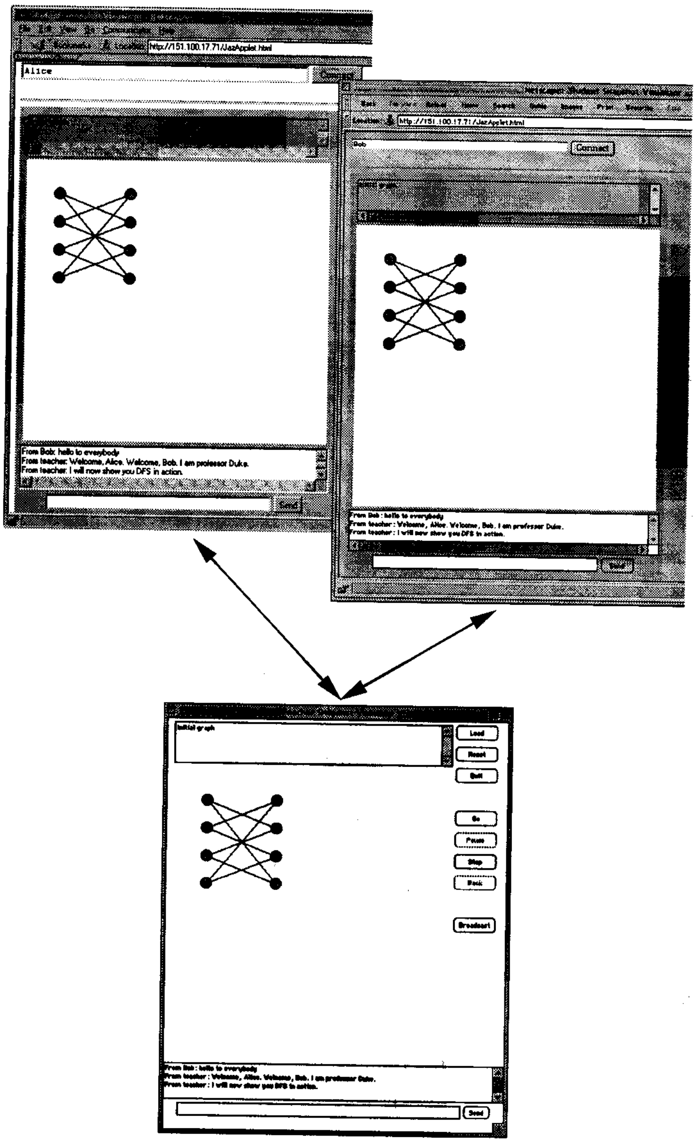
Figure 5: The network deployment of the JAZ collaborative tool

lished by means of the `Connect` button (to be precise, this part is contained in the Html document that the student opens with any browser).

An example In Fig. 6, an example of the beginning of a typical network class is shown. In this case, two students, Alice and Bob, are connected with the teacher, professor Duke (observe that Alice is running Netscape Communicator under Windows 95 while Bob is running it under MAC-OS). Whenever a student starts a new connection, the default message `hello to everybody` is broadcast through the network (observe that any chatline message is shown in the text area preceded by the identification of the sender). In our example, after the connection of Alice and Bob, the teacher sends the two messages `Welcome, Alice. Welcome, Bob. I am professor Duke.` and `I will now show you DFS in action.` Successively, he broadcasts the description of the snapshot previously loaded and currently visualized. The class can now continue: the teacher can alternatively load and, eventually, broadcast new snapshot descriptions or answer to any question that the students pose through the chatline.

4 Conclusion

In this paper we have briefly described a new SV system, called JAZ, which can also be used as multi-



From Bob: hello to everybody
From teacher: Welcome, Alice. Welcome, Bob. I am professor Duke.
From teacher: I will now show you DFS in action.

From Bob : hello to everybody
From teacher : Welcome, Alice. Welcome, Bob. I am professor Duke.
From teacher : I will now show you DFS in action.

From Bob : hello to everybody
From teacher : Welcome, Alice. Welcome, Bob. I am professor Duke.
From teacher : I will now show you DFS in action.

Figure 6: An example of a JAZ multi-platform classroom

platform distributed collaborative tool for teaching graph algorithms. With JAZ programmers can easily obtain animations to support the development, the debugging, and the explanation of graph algorithms. To this aim, they just have to annotate their Java code with JAZ primitives.

Three main research directions will be followed in the near future:

- To experiment the system in actual electronic classrooms. The main goal of these experiments should be to understand whether animations can really aid understanding an algorithm. Similar experiments have already been performed (see, for example, [10]) but, as far as we know, there are no results regarding electronic classrooms.

- To develop a fundamental graph algorithm base to be distributed along with the system. To this aim, we will mainly refer to the graph algorithm part of [13].

- To compare JAZ with other similar SV systems for electronic classrooms, such as GASP-II [23], MOCHA [5], and CAT [8]. Two examples of criteria that will be used to perform such a comparison are the ease-of-use and the delay induced by the visualization object transmission.

References

[1] Anderson J.M. MacGAIGS: General Algorithm Visualization Software for Macintosh, `http://www.uis.edu/%7Egrissom/VISUALS/algorithms.html#MacGAIGS`.

[2] Arnold, K. and Gosling, J. *The Java^{TM} Programming Language.* Addison-Wesley Publishers, 1996.

[3] Baecker, R. M. Sorting Out Sorting. *ACM SIGGRAPH Video Review* **7**, 1983.

[4] Baecker, R. M. and Marcus, A. *Human Factors and Typography for More Readable Programs.* Reading, MA Addison-Wesley, 1990.

[5] Baker, J.E., Cruz, I.F., Liotta, G., and Tamassia, R. Algorithm Animation over the World Wide Web. *Proc. of ACM AVI 96*, 203-212, 1996.

[6] Bentley, J. L. and Kernighan, B. W. A System for Algorithm Animation. *Computing Systems*, 4:5-30, 1991.

[7] Brown, M. H. Zeus: A System for Algorithm Animation and Multi-View Editing. *Proc. of IEEE Workshop on Visual Languages*, 4-9, 1991.

[8] Brown, M. H. and Najork, M.A. Collaborative Active Textbooks: a Web-based Algorithm Animation System for an Electronic Classroom. SRC Research Report 142, 1996.

[9] Brown, M. H. and Sedgewick, R. A System for Algorithm Animation. *Proc. of ACM SIGGRAPH 84*, 177-186, 1984.

[10] Byrne, M. D., Catrambone, R., and Stasko, J.T. Do Algorithm Animations Aid Learning? Technical Report GIT-GVU-96-18, August 1996.

[11] CenterLine Software ObjectCenter Reference. Cambridge, MA: CenterLine Software, Inc, 1991.

[12] Chandhok, R., Garlan, D., Meter, G., Miller, P., and Pane, J. *Pascal Genie*, Chariot Software Group, San Diego, CA, 1991.

[13] Cormen, T.H., Leiserson, C.E., and Rivest, R.L. *Introduction to Algorithms*, MIT Press, 1990.

[14] Edwards B. *Drawing on the right side of the brain.* Fontana Collins Publishers, 1979.

[15] Eisenstadt, M. and Brayshaw, M. The Transparent Prolog Machine (TPM): an execution model and graphical debugger for logic programming. *Journal of Logic Programming*, **5**: 1-66, 1988.

[16] DiGiano, C. J., Owen, R., and Rosenthal, A. J. *LogoMedia*, Department of Computer Science, Toronto, 1992.

[17] Henry, R. R., Whaley, K. M., and Forstall, B. The University of Washington Illustrating Compiler. *Proc. of the ACM SIGPLAN'90*, 223-233, 1990.

[18] Jones, V. *Visualization and Optimization.* Kluwer Academic Publishers, 1996.

[19] McCormick B.H., Defanti T.A., and Brown M.D. Visualization in scientific computing, *Computer Graphics*, **21**: 1-14, 1987.

[20] Naps T. GAIGS: General Algorithm Visualization Software for PC Windows, `http://www.uis.edu/%7Egrissom/VISUALS/algorithms.html#GAIGS`.

[21] Price, B.A., Baecker R. M. and Small I.S. A Principled Taxonomy of Software Visualization, *Journal of Visual Languages and Computing* **4**: 211-266, 1993.

[22] Roman, G.C., Cox, K. C., Wilcox, C. D., and Plun, J. Y. Pavane: a System for Declarative Visualization of Concurrent Computations. *Journal of Visual Languages and Computing*, **3**: 161-193, 1992.

[23] Shneerson M. and Tal A. GASP-II: a Geometric Algorithm Animation System for an Electronic Classroom. WISDOM Technical Report in Computer Science **CS96-21**, 1996.

[24] Stasko, J. T. Tango: A Framework and System for Algorithm Animation. *IEEE Computer*, **23**: 27-39, 1990.