UNICAMP
Departamento de Engenharia de Computação e Automação Industrial
Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas

Relatório Técnico
Technical Report
DCA-005/00

Print Facilities for Report Editor in the CoLab Environment

Bostjan Kolar
Paulo Henrique Fisch de Brito
Wu Shin-Ting

State University of Campinas)
FEEC (School of Electrical and Computer Engineering)
DCA (Dept. of Computer Engineering and Industrial Automation)
{bostjan,fisch,ting}@dca.fee.unicamp.br

December 2000

Abstract

This work aims at providing print facilities to the Colab system, a collaborative learning environment under development by the Group of Image Computing in the Laboratory of Computer Engineering and Industrial Automation. It is performed during the trainee program supported by IAESTE at the Department of Computer Engineering and Industrial Automation of The State University of Campinas - Unicamp, Brazil. CoLab is implemented in Java for taking advantages of reusing the available Java scripts, such as Java Notepad and Java Plotter. In the Colab environment, geographically dispersed students and teachers can synchronously solve a problem through textual (chat) and graphics communication (plotter) and the solution is editable as a report via a Java Notepad, which is desirable to be printable from the CoLab environment if necessary. Our proposal is to convert the report file into a HTML file an print it through a HTML browser.

1 Introduction

Colab is an environment that is intended to be useful to teachers and students in the Calculus Courses. To make the learning process more efficient, different ways of distribution and exchange of information are provided in its conception:

- internet navigator for search and presentation of the matter related to the course and the usage of the CoLab environment,
- a window with the information about the users connected at the moment,
- forum, where students can exchange their problems (and doubts) and solutions of those problems in an asynchronous mode,
- e-mail tool, as the other form of asynchronous communication (besides the forum),
- a chat tool, synchronous way of communication between the students,
- a plotter of functions, as either a synchronous tool for visualizing the functions of two variables and manipulating its parameters or an independent tool for a student to test or explore some features of functions,
- a report editor, with possibility of including text and images, to annotate ideas or remarks or the edit a report to be hand-out to the teachers (if necessary).

The problem that we had to solve was to print the report edited by a student or edited collaboratively by a group of geographically dispersed students with the use of the report editor. The task was not so easy as it seems initially, since, to be runable in any platform, Colab is implemented in a multiplatform language, the Java. This means that the compiler compiles the source code to a code that is interpreted by the virtual machine. The virtual machine then interprets this code to a machine code of the platform on which it is running. Therefore, the access to the hardware, such as a printer, is not direct.

2 Report Editor in CoLab

The report editor is adapted from the `Notpad` [2], which is a module of the Java development Kit JDK 1.2.2 freely distributed by the Sun Microsystem [1]. It provides several commonly used editing facilities, such as undo, redo, cut, copy, paste, new and load.

Class/Interface	Description
Notepad (class)	Main class that is responsible for the management of the report editor, such as the actions of a menu.
JTextPane (class)	Class responsible for the direct interface to a user, including the presentation of a text.
StyledDocument (interface)	Interface that defines the methods for editing text with a specified style.
Document (interface)	Interface responsible for the storage of a text.

Table 1: Classes and interfaces for the report editor.

The Notepad is implemented on the top of the classes belonging to Swing GUI Components. Swing is the package of Java classes that provides a rich, extensible GUI component library with a pluggable look and feel. In the original version, the JTextArea class was used for editing text. A JTextArea is a multi-line area that displays plain text. It can display and edit multiple lines of text, allowing the user to enter unformatted text of any length or to display unformatted help information. Although a text area can display text in any font, all of the text is in the same font [3].

In the CoLab environment, we consider that the students task may concern in analyzing and/or synthesizing the graphs of functions. Hence, we also provide facilities for including these graphs into a report as images. These images are indeed objects in the CoLab environment. They can be generated either by the plotter or imported from an image file. Hence, we replaced the JTextArea class by the JTextPane one that can display and edit styled text with embedded images. JTextPane can be marked up with attributes that are represented graphically. This component models paragraphs that are composed of runs of character level attributes. Each paragraph may have a logical style attached to it which contains the default attributes to use if not overridden by attributes set on the paragraph or character run. Components and images may be embedded in the flow of text [4].

Figure 1 shows the relationship of the principal classes that implement our report editor. Observe that the Notepad employs the class JTextPane for editing a document containing styled text with embedded images. This class requires the object StyledDocument for building a specified document model for a report. StyledDocument is, in its turn, derived from the class Document. Table 1 summarizes the role of each mentioned class/interface.

The Document object has a serialized hierarchical tree structure,

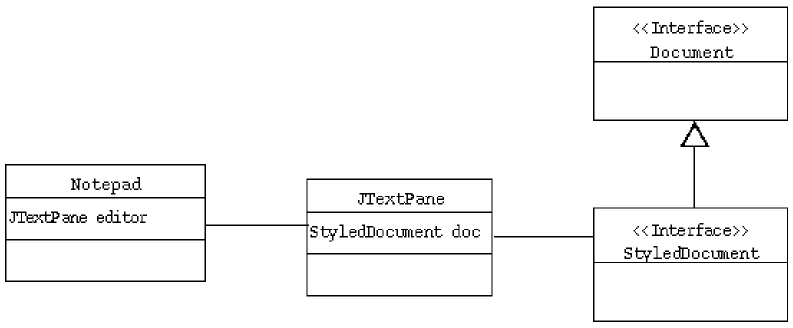


Figure 1: Relationships of the classes/interfaces implementing the report editor.

where each node is called *element*. The first node is always the *Root Element* to which are subordinated every node referring the lines and paragraphs of a document. The lines and paragraphs have, in their turn, as sons the elements of the third level of the tree – the *content elements*. These content elements contain, actually, the references to the part of text (string of characters) in the document that they represent, as shown in Figure 2.

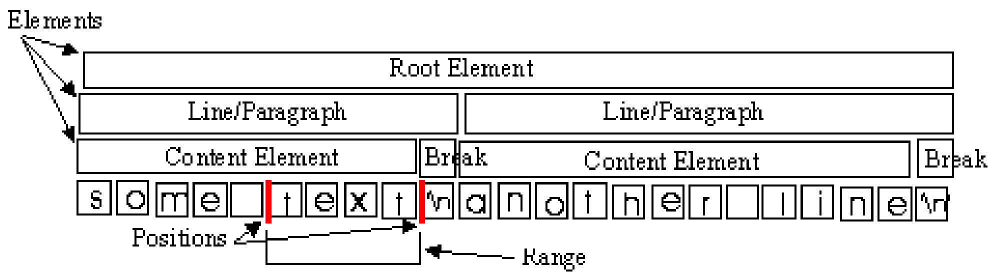


Figure 2: Data structure of a Document object.

Observe that there is no special element for distinguishing between text and images. Indeed, the images are considered attributes of an element, in addition to the attributes like underline, italic, and boldface.

3 Report Print Facilities

On the demand by the teachers of the Calculus Course, that are used to the paper media, print facilities must be provided in the CoLab environment.

As already remarked, the fact that the CoLab is implemented in Java less support can be found for print facilities, once it runs on a virtual machine isolated from the hardware features of a specific real machine. For overcoming this problem, two approaches were investigated:

- implementing a Java class for printing directly the documents in postscript file, or
- converting the document in HTML and using an adequate HTML browser for printing (e.g. Internet Explorer and Netscape).

After a brief review of the existing work and Java scripts, the first approach does not seem realizable in a period of three months. There are not freely distributed Java scripts for converting and printing documents in Postscript files. This leads us to decide for the second approach.

4 Conversion from Document to HTML

In this section we describe the algorithm we implemented to convert the Document object in the StyledDocument style described briefly in Section 2 to the HTML format. To be self-contained, the most important features of the HTML language for our work are given in the Section 4.1.

4.1 HTML

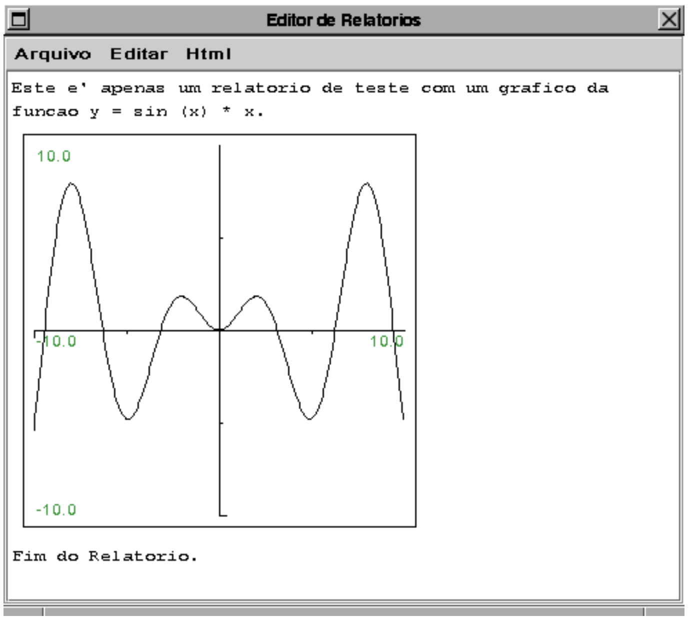
HTML (*HyperText Markup Language*) is a document-layout and hyperlink-specification language. It defines the syntax and placement of special, embedded directions or *tags* that are not displayed by the browser, but tell it how to display the contents of the document, including text, images, and other support media [5].

The first word in a *tag* is its formal name, which usually describes its function. Any additional words in a *tag* are special attributes, sometimes with an associated value after an equal sign (=), which further define or modify the *tag*'s actions.

With the *tags* one may structure the document content without regard to the final appearance yielded by a specific browser, such as section headers, paragraphs, titles, and embedded images. Figure 3 exemplifies a way to structure the report in the report editor with use of HTML.

```
<!doctype html public "-//w3c//dtdhtml 4.0 transitional//en">
<html>
```

```
    <head>
       <title>Report of Colab</title>
    </head>
    <body bgcolor="#FFFFFF">
        Este e' apenas um relatorio de teste com um grafico da
        funcao y = sin (x) * x.
        <center>
            <img SRC="images/Imagem1.jpg">
        </center>
        <br>
        Fim do Relatorio.
    </body>
</html>
```



Este e' apenas um relatorio de teste com um grafico da funcao y = sin (x) * x.

Fim do Relatorio.

Figure 3: Example of a short report.

The content element can be inserted between the <body> and </*body*> *tags* and the figures, after converting into *jpeg* format, are included via the <*img*> *tag* with the special attribute *SRC* for indicating the link to the image file.

4.2 Conversion Algorithm

For converting a Document object into a HTML file, its tree structure is scanned and the attributes of the *content elements* are analyzed. Whenever an image is found, it is converted into a *jpeg* file and included in the HTML file with the <*img*> *tag*. The text in the *content element* is extracted and linked sequentially, as presented in the following Java code (the runElementTree method).

```
    public String runElementTree (Element el) {

      String strH = "";

1     AttributeSet as = el.getAttributes().copyAttributes(); /* 1 */

      if(as != null) {

2         Enumeration        names = as.getAttributeNames(); /* 2 */
          while(names.hasMoreElements()) {

              Object        nextName = names.nextElement();

              if(nextName != StyleConstants.ResolveAttribute) {

                  Object o = as.getAttribute(nextName);

3                 if (o instanceof ImageIcon) { /* 3 */

                      String        asString = "Imagem";
                      BufferedImage bImage;

                      if (((ImageIcon)o).getImage() instanceof BufferedImage) {

                          bImage = (BufferedImage) ((ImageIcon)o).getImage();

                      } else {

                          // need conversion to BufferedImage
                          bImage = Image2BufferedImage
4                             (((ImageIcon)o).getImage()); /* 4 */

                      }

                      try {

5                         //Saving image as JPEG /* 5 */
                          FileOutputStream fos = new FileOutputStream
                              (new File (imageDirectory, asString +
                                         imageNumber+".jpg"));
                          BufferedOutputStream bos = new
                              BufferedOutputStream(fos);
                          JPEGImageEncoder encoder =
                              JPEGCodec.createJPEGEncoder(bos);
```

```
                        encoder.encode(bImage);
                        bos.close();

6                       //Generating HTML code /* 6 */
                        strH = strH.concat("<center>\n<img SRC=¨");
                        strH = strH.concat("images/" + (asString+imageNumber)
                                           + ".jpg");
                        strH = strH.concat("¨>\n</center>\n<br>\n");
                 imageNumber++;

                    } catch (Exception e) {

                        System.out.println ("Colocar um dialogo");
                        e.printStackTrace();

                    }
                }
            }
        }
    }

7   //extracting text to html code /* 7 */
    if (el.isLeaf()) {

        strH = strH.concat("</p> \n <p>");

        try {

            strH = strH.concat (el.getDocument().getText(el.getStartOffset(),
                         el.getEndOffset()-el.getStartOffset()));

        } catch (Exception e) {}

    }

8   //Recursive traverse of the element tree /* 8 */
    int n = el.getElementCount();
    if (n > 0)
       for (int i = 0; i<n; i++)
9         strH = strH.concat(runElementTree (el.getElement(i))); /* 9 */

    // return htmlCode
    return strH;
}
```

In the step **1** the attribute list of the *content element* is checked. If it is not empty, we search in the step **2** the reference for an image in the list. If there is a reference to an image (step **3**), we need to encode it in a *BufferedImage* (step **4**) before converting it into the *jpeg* format using the class JPEGImageEncoder and storing it in the ".images" directory (step **5**). Then, in the step **6** the <img> *tag* is included in the HTML file with the correct image hyperlink. In the step **7** the text in the *content element* is extracted and tagged with the <body> *tag*. The procedure is repeated recursively until all the *content elements* are traversed (step **8**) and their text and images put together (step **9**).

Structurally, the runElementTree method only convert the text and images in a Document object to the content of the body of a HTML file. The outer <*html*> tag *enclosing the HTML document header and body, and the* <head>, <*title*>*, and* <body> *tags* must be generated additionally, such as in the following code:

```
String htmlString = "<!doctype html public ¨-//w3c//dtd"+
  "html 4.0 transitional//en¨>\n<html>\n"+
  "<head>\n<title>Report of Colab</title>\n"+
  "</head>\n<body bgcolor=#̈FFFFFF¨>\n";

Element[] roots = getEditor().getDocument().getRootElements();
htmlString = htmlString.concat(runElementTree (roots[0]));
htmlString = htmlString.concat("<br>&nbsp;</body>\n</html>");

try {
   FileOutputStream outFile = new
      FileOutputStream(chooser.getSelectedFile());
   outFile.write(htmlString.getBytes());
   outFile.close();
} catch (IOException ioe) {}
```

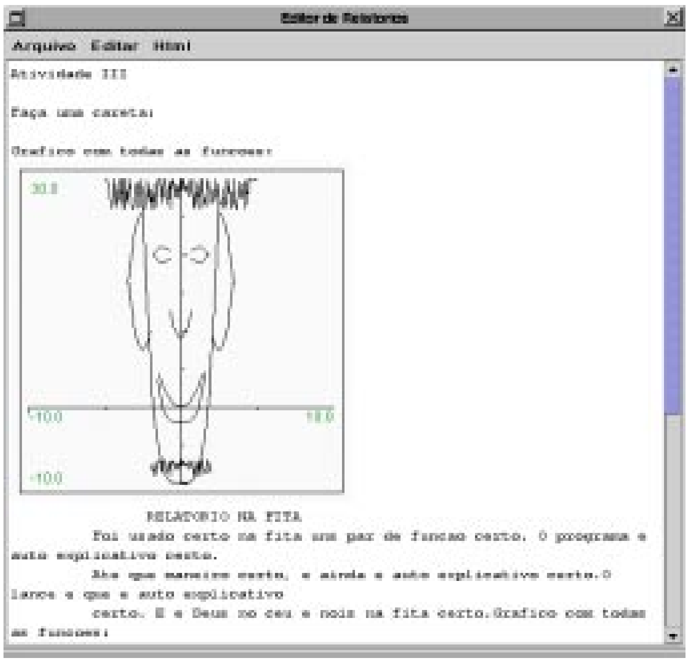
Observe that at the end of the task, the resulting HTML code is written in a file.

5 Experiments and Results

To exemplify how our report editor works, particularly how it prints a document, we present the conversion of a report of a task performed by a student of the Institute of Mathematics at Unicamp.

The task consists in drawing a mask with the use of 2-variable functions and the student should make a brief comment, as depicted in Figure 4.

The report was successfully converted into the following HTML file:



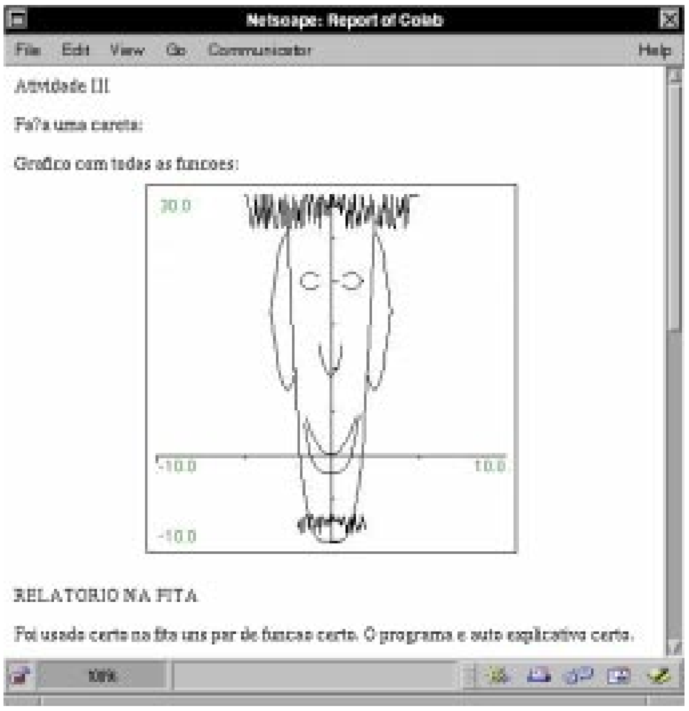
```
Atividade III

Faça uma careta:

Grafico com todas as funcoes:

                 RELATORIO NA FITA
          Foi usado certo na fita uns par de funcao certo. O programa e
auto explicativo certo.
          Aha que maneira certo, e ainda e auto explicativo certo.O
lance e que e auto explicativo
          certo. E e Deus no ceu e nois na fita certo.Grafico com todas
as funcoes:
```

Figure 4: A Report.

```
  <!doctype html public "-//w3c//dtdhtml 4.0 transitional//en">
<html>
<head>
<title>Report of Colab</title>
</head>
<body bgcolor="#FFFFFF">
</p> <p>
Atividade III
</p>  <p>
</p>  <p>
Fa?a uma careta:
</p>  <p>
</p>  <p>
Grafico com todas as funcoes:
<center>
<img SRC="images/Imagem1.jpg">
</center>
<br>
</p>  <p>
</p>  <p>
</p>  <p>              RELATORIO NA FITA
</p>
 <p>         Foi usado certo na fita uns par de funcao certo. O programa e auto
explicativo certo.
```

```
</p>
 <p>         Ate que maneiro certo, e ainda e auto explicativo certo.O lance e que
e auto explicativo
</p>
 <p>         certo. E e Deus no ceu e nois na fita certo.Grafico com todas as funcoes:
</p>
</p>  <p>
<br>&nbsp;
<p>Here comes the Authors Name and Date!!!!!
</body>
</html>
```

The appearance of the document rendered by the Netscape browser is given in Figure 5.



Atividade III

Fa?a uma careta:

Grafico com todas as funcoes:

RELATORIO NA FITA

Foi usada certo na fita uns par de funcao certo. O programa e auto explicativo certo.

Figure 5: Report rendered by the Netscape browser.

6 Conclusions

The conversion algorithm has worked as expected, although it was not the best solution for providing print facilities in the CoLab environment. Our solution demands more on the cognitive process, since more steps are necessary to carry out a task.

It is foreseen that in the JDK Release 1.4 the API for printing will be improved, when we will test the feasibility of printing directly a report in the CoLab environment.

References

[1] -, `http://java.sun.com/products/jdk/1.2/`

[2] -, `http://www-uxsup.csx.cam.ac.uk/java/jdk-1.2.2/demo/jfc/Notepad/`

[3] -, `http://java.sun.com/j2se/1.4.1/docs/api/javax/swing/JTextArea.html`

[4] -, `http://java.sun.com/j2se/1.4.1/docs/api/javax/swing/JTextPane.html`

[5] Chuck Musciano and Bill Kennedy, HTML: The Definitive Guide, O'Reilly & Associates, Inc., 1996 (ISBN:1-56592-175-5)