USAGE OF DIGITAL PEN BASED DEVICES IN E-LEARNING

Adrian Neațu, Dan Tuşaliu, Robert Popescu, Ecaterina Manole

University of Craiova, Faculty of Automation, Computers and Electronics,
Computer and Communications Engineering Department

Abstract: In this paper it is presented the possibility of usage of new tools in the learning process. On one hand there is the hardware, the digital pen based devices, such as a tablet PC, and on the other hand new applications able to facilitate the teachers or professors the use of these devices. The goal is to give the possibility to add value (information) to already existing electronic courses, thus improving explanation of concepts, student involvement in the teaching process and also assimilation of knowledge.

Keywords: e-learning, tablet PC, digital pen, open source.

1. INTRODUCTION AND MOTIVATIONS

Tablet PCs represent a challenging new technology that can offer multiple facilities. In the e-learning environment, tablet PCs or other digital-pen based equipments can allow for visible improvements through development of active and collaborative learning systems. These types of systems are considered to be efficient and flexible from the teaching point of view and have the tendency to encourage the involvement of students in the teaching act, as well as encouraging the student – teacher interaction.

The vast majority of existing digital pen based annotation systems, or other applications using digital ink have been developed in the C# programming language, utilizing the Microsoft.NET framework. The novelty and the challenge in the presented application is that it is developed using only the Java technology.

Although controversial, the slide based presentations present a series of advantages, such as: possibility of preparing in advance the materials to be presented, possibility of envisaging quality examples and illustrations, ease of reuse and facilitation of distance learning.

The main idea behind the application is to offer the possibility to annotate presentation slides using a digital pen. Unfortunately, the Java programming language does not offer support for such implementation, such that the solution adopted is to use an intermediate stage, namely transform the slides into images. Later, these images can be annotated. For this, the graphical libraries found in Java were used, which offer support for digital pen stroke recognition.

Ultimately, the goal for a teacher is to progressively annotate the images, also having the possibility of organizing these images into projects, thus being able to reconstruct the teaching process evolution. Annotation will be done by means of a pen onto a tablet PC or simply on any graphical tablet. The teacher may modify the images by using handwriting this way.

It is desired the maximization of the teaching process efficiency, by trying to add in a natural way supplemental information to electronic courses. This information is added with the goal of better explanation of concepts, more involvement of students in the teaching process and therefore a better assimilation of the presented knowledge.

For portability reasons, the application has been developed entirely using the Java programming

language and a free integrated development environment.

2. GENERAL PRESENTATION OF THE APPLICATION

2.1 The interface

The graphical interface comprises the drawing area or blackboard, where the actual annotation is realized, based on a certain document or which can be used as a board to edit the course.
Besides the working area one can find a menu formed of the buttons **File, Edit, View, Project** and **About** whose functionalities are later detailed.
In the left side there is the line selection menu, the color menu and the buttons for the movie mode.
In the right side of the application window there is a tree component, used to display the projects and their components. Modifications can be applied with the aid of the **Add, Remove, Up** and **Down** buttons. Navigating the slides composing a project is possible by means of the **Previous** and **Next** buttons**,** as shown in the below figure.

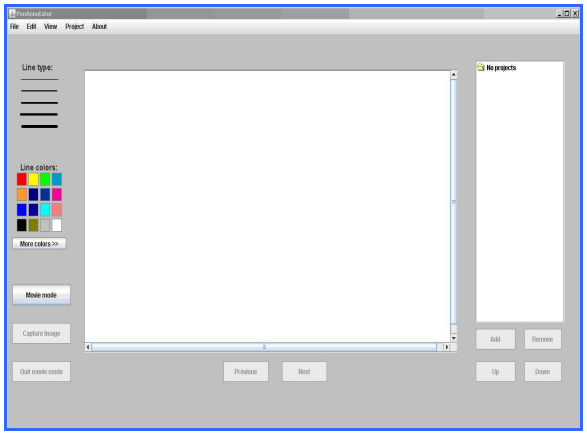

Fig. 1. Main window. The drawing area, the menus: upper and left menu, and the right panel presenting the projects

Upper menu consists of the submenus File, Edit, View, Project and About.
The **File** submenu implements mainly the classic functions of any text editor: creating a new blank page, loading an image, saving the image with a certain extension and exiting the program.
The **Edit** button implements the **undo** function, which is deleting the last annotation that is the modification made in the interval from the moment when the pen was pressed on the drawing area until it is lifted from the drawing area.
View submenu implements the modality of displaying the images, navigating through them in the direct manner (Next Image) or indirect (Previous Image) and also deleting an image. The order of the slides in the project can also be modified. The slides will be noted with "*" in order to keep track of the unsaved changes.

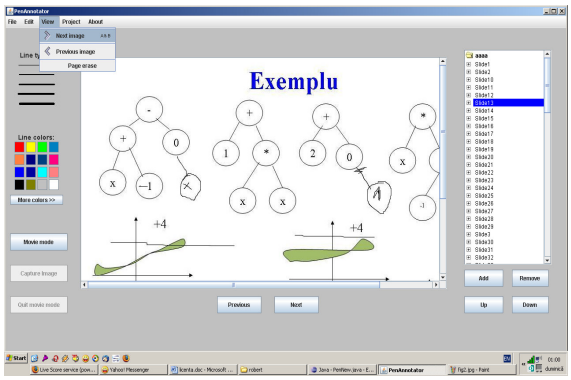

Fig. 2. Main window. The **View** submenu is dropped, the components of a project are listed on the right side of the screen.

Project submenu administrates the working mode of the project. New Project, Open Project, Save Project and Add directory are implemented in it, also as Add file or Delete File. In the **Save Project** function there is a feature to choose which slides one wants to save, as shown in the next figure.
In the **Add directory** function, a filter is added, in order to import only one folder in a project.

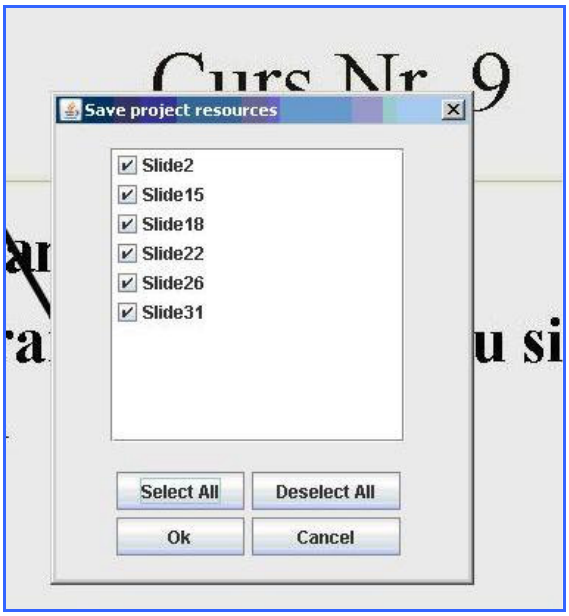

Fig. 3. Saving different slides in a project. The **select all** button will save all the slides, and the **deselect all** will ignore any modification.

Line type represents the menu for choosing the favorite line type. There can be used 5 different line types.

Line colors implements the selection menu of the favourite color. The **More Colors** button will open a **JcolorChooser** to be able to choose more color on

RGB (Red, Green, Blue) or HSB (Hue, Saturation, Brightness).

2.2 The implementation

The main window is an extension of **JFrame from javax.swing**. On this window there is a panel from **JPanel** where the other components are added. The menu bar is a **JMenuBar** with the 5 menus added.
All these menus are implemented with **JMenu** components.
Each menu has a list of options to select. The options have been implemented with the aid of **JMenuItem** component. Constructing this component implies an option to select and **ImageIcon** object that has as parameter the URL of the picture. The names of the images are retained using **Reflection** from the **images** directory of the application. The menu that presents the width of the lines that are used to add notes is made of a **JLabel** that has set the title to "Line Type**"** and 5 **JToggleButton** buttons customized to apply images on them for de 5 line types. It is followed to activate the border of the button when pressed, and the border should disappear when other button is pressed. The menu for colors is made of a **JLabel** with the title "Line Colors" and 16 **JButton** customized buttons for which there is a set background with the most representative colors from RGB. Pressing such a button will set the selected color for drawing.
To select other colors, the **JButton** button **More Colors** is used. By pressing it a **JColorChooser** will be opened and the user can chose colors based on **HSB** (Hue, Saturation, Brightness), or **RGB** (Red, Green, Blue).

The drawing area is a **JLabel**. The class that implements this component is **ScrollablePicture.** A **ScrollablePicture** component is mainly a JLabel that permits scrolling, by implementing the Scrollable interface. The image on the JLabel will be setted as follows:
- the image is loaded on the disc in an **ImageIcon** object
- the object image will be extracted with the **getImage()** method
- the **Image** object will be drawn on an object member of **BufferedImage**. An **antialising** is used for a better drawing quality.
- The BufferedImage object is set as icon for the ScrollablePicture object.

There is also a control implemented by **MediaTracker** class. So, the image will be shown only if it has loaded completely from the hard disk, else a blank image will be shown.

An undo operation is also implemented, by creating an undo directory each time the **ScrollablePicture** class is instantiated. In this folder, different stages of the picture will be stored, in order to be able to iterate between them, loading finally the one needed. The name of the files in the undo directory will be the timestamp of the system. At the beginning the directory will hold only the initial image. It is used the hard disk saving to save the virtual memory. After each intermediary saving, the directory is emptied, and after exiting the application, the directory is erased from the disk. So there will be a directory for each session. The directory will be placed at the same path as the original image, and will have its name. This method also detects the pen when not being on the tablet, and saves the intermediary image in the undo directory.

The graphical part of the project is implemented with a **JList** in which are added elements the name of the application and the images.

Behind the logical part of the application there are 3 classes:

- **Project** class extending **AbstractListModel** from **javax.swing** where the organizing of the project is realized. Three homolog list are used: one to retain the paths to the pictures in the project, one for the names of these images, and one containing ScrollablePicture objects:

```
paths = new ArrayList<String>();
names = new ArrayList<String>();
pictures = new ArrayList<ScrollablePicture>();
```

In the project will appear a list of the images' names. In this class are implemented the add file, delete file, empty project, save and load project:

```
public boolean add(Object object) {
……
}

public void remove(int i) {
……
}

public void clear(){
……
}

public void readFile(){
……
}

public void writeFile(){
……
}
```

This class implements the moving of the files in the project, on different positions:

```
public void moveUp(int i){
……
}
```

```
public void moveDown(int i){
……
}
```
and I is the index of the image in the list.

- **CustomCellRenderer** class will set the form for the list of the project. This class implements **ListCellRenderer** from **javax.swing.** The elements are JLabel with an icon and names attached, and the name will be the same as the pictures name. There are two types of icons:
  - One for the element of type project, Will contain an icon and the name of the project
  - One for the image type element. Will contain an icon and the name of the image.

```
projectIcon = new
        ImageIcon("./images/project.jpg");
itemIcon = new
        ImageIcon("./images/plus.jpg");
```

In this class, the color is set, blue for background and white for writing color, for a selected element, or white background and black writing for a unselected element.

- **Selector** class indicates the actions that are made when an image is selected in the project. As follows**:**

  - If the current image is not in the project but was modified after loading, and an image from the project is selected, then a pop-up save window will appear for the current iamge. If the user decides not to save the image, the new selected image from the project will take its place on the drawing area.

```
if(window.getPathLabel().getText().equals("
") == false)
        {
int filenamePos =
window.getPathLabel().getText().lastIndexOf(Syste
m.getProperty("file.separator"));
ArrayList<String> fileNames = new
ArrayList<String>();

fileNames.add(Utils.getName(window.getPathLa
bel().get    Text().substring(filenamePos + 1)));

        ArrayList<String> paths = new
ArrayList<String>();

paths.add(window.getPathLabel().getText());

//if saving is asked

if(window.getPicture().isModified() == true)
        {
SaveWindow sw = new SaveWindow(window,
fileNames, paths, false, "Save file");
                sw.initComponents();
        }

//deleting undo files associated with this image

Utils.removeDirectory(window.getPicture().getU
ndoFile());

window.getPathLabel().setText("");
}
```

  - If the current image is in the project, and another image in the project is selected, the second image will be shown in the drawing area:

```
window.setPicture(this.displayedPicture);
window.drawWindow();
```

For the Movie Mode part there are utilized the following 3 classes:

- **MovieModeOptionsDialog** extending **JDialog** and interrogates the user about the way to use the movie mode saving: by click or after some seconds indicated by the user. By saving after a click, it is understood saving by clicking the **Capture button.**
- **MovieModeListener** implements saving the images in movie mode. Whether it is click saving or auto-saving, first a project for movie mode is created. Second, a directory to save the pictures resulted from movie mode. This directory will be created in the same folder as the image selected for movie mode and will have the image's name.
  - For click saving, each time the **Capture** button is acted, the current state of the picture will be saved in this directory, and will be named as follows: projectName + currentNumber + "."+Utils.*png* where currentNumber cand be between 0 and 1000
  - for auto-saving, the class **MovieModeAutoSaving** is used.
- **MovieModeAutoSaving** is actually a thread which for a certain number of seconds given by the user will save the current state of the picture.

```
public void run()
{
        Project p =
penWindow.getActiveProject();
        String path =
Utils.getName(p.getFile().getPath());
        File movieDir = new File(path);
        movieDir.mkdir();
        while(keepGoing)
        {
                String next =
getNextPictureName();

Utils.saveTo(penWindow.getPicture(),
movieDir.getPath()+"\\"+next);

p.add(movieDir.getPath()+"\\"+next);
                try{
```

```
                    Thread.sleep(seconds *
1000);
                }
                catch(InterruptedException e)
                {
                    System.err.println("Sleep
exception");
                }
            }
        }
```

Exiting the movie mode is realized by pressing the **Quit movie mode** button**.**

3. USAGE INSTRUCTIONS

3.1 Project-oriented mode

This mode permits organizing files (these files currently are pictures in any form of extension known at the moment) in projects, for a better logical structure and also for easier further access of the application.
To create a project, **File -> New Project** has to be selected. This option will create a new and empty project.
For adding files in this application, it is followed the following sequence of events:

- **Project -> Add File** to add only one file to the project
- **Project -> Add directory** to add the contents of a directory in the project

After the selection, the files added can be viewed in the right side of the application window. By selecting one file in the project, its content will be displayed in the drawing area. In this way the user can add notes to the specified file, as shown in Fig.1. Viewing files on the drawing area.

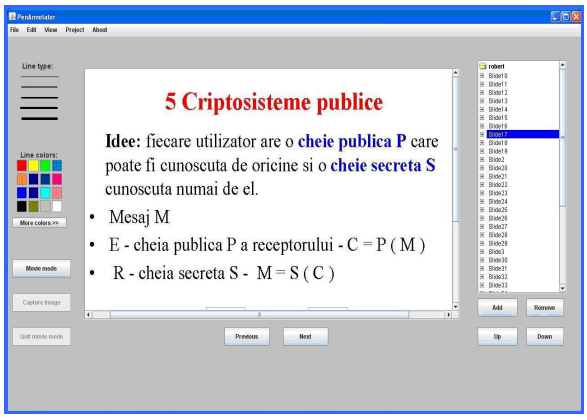

Fig. 4. Viewing files on the drawing area. This feature offers to the user the possibility to see the contents of files

In order to keep the file on the disc for further usage, the **Project -> Save Project** option will be chosen. Only in this way the project will be available in a future work session. This option is shown in the below figure.

Fig. 5. Saving a project. Using this feature the project will be available for further usage.

It is also provided the feature to travel between the file projects, compliant with the drawing area. The traveling is possible using **Next and Previous** buttons in the main windows of the application or in the **View -> Next Page** and **View -> Previous Page** options.
The user can also change the order of the files in the project using **Up and Down** buttons.
When adding notes, the user has the undo possibility by **Edit -> Undo.**
To open an older project, there is the **Open Project** option in **Project** menu.

3.2 File-oriented mode

This mode permits changing a certain file loaded by the user with the command **File -> Load Image**. In the upper part of the window the absolute path of the file will be displayed.
In this work mode the option **File->Page Erase** is available, which will replace the drawing area with an empty page.
After the user has finished adding notes, the saving feature to save the modified file is made possible by using:

- **File -> Save image** to save the file with the current name
- **File -> Save images** to save the file with a different name

The user also has the possibility to work in parallel with these two modes, the project-oriented mode and

the file-oriented mode, as presented in Fig. 3. Parallel usage of project and file – oriented modes.

Fig. 6. Parallel usage of project and file – oriented modes. The file has been loaded in a project and then saved within the project.

3.3 Movie mode

In this mode the user has the possibility to view the evolution by modifying a file from the moment it was open until it is saved, after making the modifications. This actually means that a new project is associated to each modified file, and in this project will be kept as separate files (pictures), the intermediary states of the file.

To enter movie mode, the **Movie Mode** button from the main window is used. It will appear a dialog to ask the user for the mode to save the intermediary pictures:

- For a **click** (mainly the user decides what intermediary states to be saved in the movie-mode project; these states are saved with the button **Capture Image** in the main window)
- **Self-acting**, for a number of seconds specified by the user.

This option is shown below.

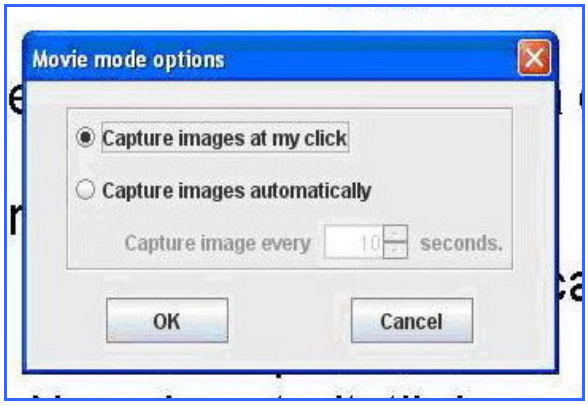

Fig. 7. Movie mode options. In this dialog box the user selects the mode to save de states of the files

To exit movie mode the **Quit movie mode button** is used.

Saving the movie-mode project is made as with a usual project.

Movie-mode projects are thought to facilitate the learning process by better distinguishing the phases in which the concepts are presented.

REFERENCES

Bloch, J.and A. Wesley. (2002). *Effective Java*

Bruce, E. (2002). *Thinking in Java,* Prentice Hall Ptr.

Flanagan, D. (2001). *Java in a Nutshell.* Oreilly & Associates Inc.

Marinacci, J. and C. Adamson (2005). *Swing Hacks*. Oreilly & Associates Inc.

http://www.cs.usna.edu/~lmcdowel/pubs/fie2005.pdf

http://www.cs.washington.edu/homes/jonsu/ITICSE_2004.pdf

http://www.cs.washington.edu/education/dl/presenter/papers/2006/RA_SIGCSE_2006.pdf

http://www.cs.washington.edu/homes/fred/wace-2003-pres.pdf

http://people.depauw.edu/dberque/ccsc_ne_2006_berque.pdf

http://people.csail.mit.edu/albert/pubs/2003-ashuang-ugrad-thesis.pdf

http://www.formatex.org/micte2005/206.pdf

http://ils.unc.edu/annotation/publication/Fu_etal_ASIST05.pdf

http://csdl2.computer.org/comp/proceedings/hicss/1999/0001/02/00012012.pdf

http://www.orcca.on.ca/PenMath/materials/papers/2005-msc-pencontext.pdf

http://www.cs.washington.edu/homes/jonsu/CG_2005.pdf

http://www.carle.ws/PACT.pdf

http://www.cs.clemson.edu/~pargas/tabletpc/PargasTabletPCProposal.pdf

http://www.math.uaa.alaska.edu/~afkjm/papers/mock-ccsc2004.pdf

http://www.cs.usna.edu/~lmcdowel/pubs/fie2005.pdf