Design and Implementation of a Presentation Tool for an Educational delivery platform

Term Project

Nadir Weibel (nad@nadnet.ch)

Prof. Dr. B. Plattner

Supervisor: Georgios Parissidis

Institute for Technical Informatics and Communication Networks

Swiss Federal Institute of Technology - Zurich

Version 1.0
Last Update February 5, 2003

Abstract

Nowadays, information and communication technology is used progressively more for teaching and learning purposes. A lot of universities around the world are opening their campus offering distance learning courses. Distance learning occurs when the lecturer and the students are separated in physical distance, and delivery systems (i.e. voice, video and data) are used to bridge the instructional gap.
The objective of the present thesis is design and implement a *Presentation Tool* as a component of a Synchronous Distance Learning System (SDLS), which will provide learning facilities to students geographically located far away from the lecturer. Such a system must be reliable, platform independent, flexible, scalable and user-friendly.
For satisfying the aforementioned requirements, this thesis presents a Java-based *Presentation Tool* to be used as a collaborative medium by both the tutor and the student. This tool gives the opportunity to the tutor to give a lecture showing a presentation, interacting with the students with an embedded whiteboard and managing the students during the lecture. The tutor can also manage the presentations (upload new presentations, download a presentation) directly from the tool. Moreover students can communicate with the tutor or among them using either the integrated Chat Mechanism or requesting access to the whiteboard or the pointer.
Such an environment satisfies both the requirements imposed by an educational lecture and a conferencing system.

Heutzutage, werden Information und Kommunikation auch in Bereich der universitären Bildung immer wichtiger. Universitäten der ganzen Welt versuchen ihren Campus zu öffnen und Online-kurse anzubieten. Den Studenten wird die Möglichkeit gegeben standortunabhängig Vorlesungen zu folgen. Um das Problem der physische Trennung zu lösen existieren verschiedene technische Möglichkeiten (Sprache, Video und Data).
Die Projektabsicht ist die Entwicklung, Implementierung und Aufbau eines *Presentation Tools*, als Komponent eines Synchronen Distanz Lern System (SDLS). Diese Applikation soll zuverlässig, betriebssystem-unabhängig, flexibel, skalierbar und benutzer-freundlich sein.
Um diese Anforderungen zu erfüllen, wird ein Java-basiertes *Presentation Tool* entwickelt , das, sowohl von einem Lehrer als auch von den Studenten, als Medium zur Zusammenarbeit benutzt werden kann. Dieses Tool erlaubt den Lehrern eine Vorlesung zu leiten, in dem er eine Präsentation durchführt, und, dank dem integrierten Whiteboard, mit den Studenten interagiert. Der Lehrer kann damit auch die Präsentationen administrieren: er kann neue Präsentationen aufladen oder existierende herunterladen und zeigen. Zusätzlich können die Studenten mit dem Lehrer oder untereinander kommunizieren in dem sie den integriertes Chat-Mechanismus benutzen oder den Zugriff auf das Whiteboard beantragen.
Diese Umgebung erfüllt sowohl die *educational-lecture-* als auch die *conferencing-system-*Anforderungen.

Acknowledgments

My thanks to the Department of Information Technology and Electrical Engineering specifically to the Institute of Technical Informatics and Communication Networks (TIK), Prof. Bernhard Plattner and my supervisor Georgios Parissidis for the great support and the excellent working conditions during the whole project, for giving me the possibility to gain more experience with the Java GUI implementation and with such an interesting topic.

— Nadir Weibel, D-Infk, ETH Zurich, 2003

Contents

List of Figures

Chapter 1

Introduction

Distance Learning Systems (DLS) are a known alternative solution for learning when the tutor and the students are separated by geographical distance. In the past, such systems have been realized delivering educational content with email's, cd's, video-tapes or web-based learning management systems. All these methods present a common disadvantage: there is no direct feedback and collaboration between the lecturer and the students.

Nowadays with the large expansion of the internet and the advance of the information and communication technologies, is possible to develop a system for Collaborative Distance Learning. What does this mean? We don't need anymore a standard lecture with physical presence of the students? One obvious answer could be *no*! DLS are thought only as an help for those people which could not be present in a specified location at a specific time to follow the course. Anyway this solution could be also the basis for a complete online diploma program, but this is more an ethical and organizational question and it won't be answered in this thesis.

Distance Learning Systems consist usually of three interworking components:

- **Brokerage System:** used to resolve queries about the services provided by a DLS, as for example available courses, lecturers participating, etc.
- **Learning Management System:** used by the lecturer to design course modules or generally speaking to "wrap knowledge into an appropriate form so as for it to be delivered to students in an educational manner and to assure reusability, ease of maintenance and portability to heterogeneous delivery platforms".
- **Delivery Platform:** used to deliver the input from the Learning Management System to the students and to enable interaction between the participants. Typical examples are real-time delivery platforms (conferencing systems adapted to educational scenarios) and web-based platforms.

There are two categories of DLS, asynchronous and synchronous, depending on whether the participants are separated in time or not, respectively. Asynchronous DLS does not require the simultaneous participation of students and instructors and the instructional material can be delivered with e-mails, cds, video-tapes or web-based learning management systems

(LMS). On the contrary, synchronous instruction requires the simultaneous participation of all students and instructors, creating a virtual classroom where the participants of the lecture are able to interact in "real-time". In that case, high-quality audio, video, presentation and annotation tools constitute the standards for a synchronous educational delivery platform (fig 1.1), imitating and duplicating in a way the classic methods for tutoring.
In this synchronous educational delivery platform we will be focusing on the design and development of a *Presentation Tool* that will be used as a medium to deliver educational content from the tutor to students. Using this *Presentation Tool* as an embedded component of an educational delivery platform, the tutor will be able to present a slide-show synchronously to all the participants [13].

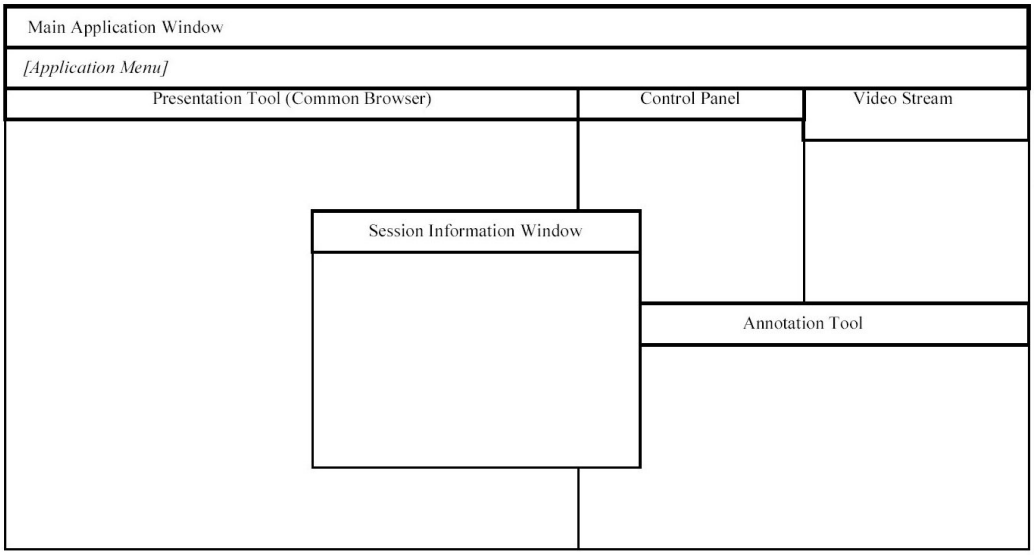

Figure 1.1: Graphical User Interface of the synchronous educational delivery platform

A synchronous educational delivery platform can be seen as an educational content multimedia conferencing system, enhanced with additional features in order to satisfy the requirements imposed by an educational lecture. The participants of such a conferencing system are divided in two categories:

1. the tutor
2. the students

As described also above, the tutor is using the *Presentation Tool* as a medium to distribute the educational content to the students separated in physical distance. The developed application is conforming to the requirements of a conferencing system and an educational lecture.

Chapter 2

Prototype

As first step, in this chapter the base prototype for the *Presentation Tool* will be defined. This prototype is the base of the following design and implementation phases and contains all the requirements to be satisfied.

2.1 Programming Language

The whole project is written in the *Java Programming Language* [14]. This allows portability among different operating systems and usage over the internet (Java applets). Java is also known to be robust, secure, scalable, distributed and dynamic, all important features for this project. Moreover the Java Programming Language allows to easily embed already written code and libraries.

2.2 Functionality and Architecture

How would the *Presentation Tool* really look like and work? There are some important features which define its functionality:

- The educational content is provided and guided by the tutor. The tutor lets the course be accessible by setting the IP address of the running course into the common access-point file, located on a known server, and by uploading on the system's server the presentation material. Thus, when a student wants to participate in the offered course, the application (during the initialization phase) checks the status of the course (online/offline), downloads and saves the content of the presentation on the workstation of the participant. As the flow of the presentation is operated by the lecturer, the advancement of slides on the students' side is synchronized. Furthermore, the application is synchronizing students, who join the conference later than its start, to the actual status of the presentation.
- Students are accessing the presentation slides by downloading them from a known server. The server's address is transmitted to the students by the tutor at the time

they connect to the selected course. The location (IP address) of the course, can be obtained by looking in the common access-point file. For being able to download the slides and thus to follow the presentation, each student must provide a login and a password to access the given server. This trivial mechanism, is used as a sort of student's authentication protocol.

- The application essentially consists of a web-browser embedded in the application, that must be capable of supporting cross-platform widely accepted content format, as HTML and XML. Furthermore, for those tutors using Microsoft's *PowerPoint* as their presentation production tool and that want to maintain animations or sound effects in their presentation material, the use of a special conversion tool [1], that transforms files in PowerPoint format to a browser-independent and thus cross-platform format should be taken into consideration.

- The general educational conference is controlled by a custom *Conference Control Protocol* (CCP), currently being developed in TIK institute based on requirements posed by an educational content multimedia conferencing system. The aforementioned protocol tries to combine and unify in a common interface of conference management two distinct tasks: conference setup (including conference discovery) and conference course control. The *Presentation Tool* as a component of the conferencing system is developed comparable to the architectural aspects of the CCP protocol.

- The application contains an indicator (pointer) offering to the tutor the ability to point into the presentation material. The indicator is shown only in the moments of pointing by the tutor or the student. The students can make use of the indicator under the appropriate tutor's permission. Furthermore, a whiteboard is provided: is placed over the presentation slides as a transparent component, so that the tutor (or the students if allowed) can draw over them for underlying some important aspects.

- Above all, the distributed application is scalable and reliable and fulfils certain temporal synchronization constraints among all the participants. Therefore, the application is not depending on the number of participants and supports the registration and deregistration during the conference, without implications to the flow of the presentation. Reliability is achieved using a reliable transport protocol, i.e. the Transmission Control Protocol (TCP).

[1] Examples of such technologies can be found in [15] [16] [17]

Chapter 3

Research

Before starting the real design and implementation phases, is useful to have an overview of the existing technologies that can be used as underlaying base for the prototype described in chapter 2.

3.1 Web Browsing Technologies

As specified in chapter 2, the essential part of the *Presentation Tool* is a web-browser embedded into the application, capable of displaying HTML and XHTML documents.
Actually, the implementation of such a browser is out of the scope of the present thesis, so a through research of existing Java browsers has been done so as to satisfy the posed requirements of the application.

In the following sections some of the most important embeddable Java browser alternatives are analyzed and, as a result of the research, the advantages and the drawbacks of each technology are referred.

3.1.1 HotJava Browser 3.0 and HTML Components[1] [2]

HotJava is a product developed by the *Sun Microsystems Laboratories* and is intended as an highly-customizable modular solution for creating and deploying web-enabled applications across a wide array of environments and devices.

This solution offers a modern browser with HTML capabilities, comparable with Netscape Navigator 3.0. The HTML rendering is achieved by using the Hotjava *HTML Components* (a *JavaBean* component).
This technology offers a "modern" look-and-feel user interface and supports most of the general purpose Java applets. JavaScript (Full ECMA 1.4 standard), the HTTP 1.1 protocol and the HTTP authentication mechanism are supported by HotJava.
HTML is supported up to version 3.2++ with persistent cookies mechanism.

Advantages

- JavaScript Support
- HTTP 1.1 Compliant
- HTTP Authentication
- Cookies and Session Management Capabilities

Drawbacks

- Not HTML 4.0 Compliant
- Based on JDK 1.1.6 (HTML Components do not run with Java2)
- HTML Components are end-of-lifed (No further development)
- No XML support

3.1.2 Mozilla and WebClient [3]

Mozilla is a known and widely-used browser. For the Java environment, Mozilla has developed a product called *WebClient* which can be embedded in a Java application allowing the application to use all the capabilities of the major browsers by means of a multi-browser implementation (Mozilla Milestone 9, Internet Explorer 5, HotJava Browser Bean). It provides a browser-neutral Java API that enables generic web-browsing capability like web-content rendering, navigation, a history mechanism, and progress notification.
WebClient can be programmatically included in a Java application by using it as a library (jar archive) or as a JavaBean (by embedding it using a bean enabled development tool). It supports JDK 1.1.7, JDK 1.2.1 and greater from *Sun Microsystems*.
This product has Java world-class HTML rendering capabilities: super-fast, fully-functional, fully-standard-supporting, well supported, widely used, well tested, etc. It can be seen as a "thin layer" of software on top of a browser supporting Bookmarks, Caching, Cookies, Copy/Paste, DOM Access, Event Handling, History, HTML 4.0, JavaScript, various protocols, Window manipulation and XML (if native browser supports it).

Advantages

- Supports all features of most recent browsers
- Easy-to-use interface
- Java world-class HTML rendering capabilities
- Latest JDK

Drawbacks

- Not Browser independent (requires native browser installation)
- Difficult to build

3.1.3 BrowserG! [4]

BrowserG! is a Java-based desktop application that provides important functionality and enhances the existing habits of the common web surfer. It is essentially an extension of the *Mozilla WebClient* (is based on WebClient 1.1 Technology, see section 3.1.2) and therefore provides the same characteristics of this product.

Advantages

- Look at *WebClient* (section 3.1.2)

Drawbacks

- Look at *WebClient* (section 3.1.2)

3.1.4 IceSoft IceBrowser [5]

IceBrowser is a fully functional, 100% Java-based web-browser supporting all of the latest standards. It can act as a stand-alone Java application, or be integrated into an application or product as a "browser window". Its architecture is designed to handle any content, such as HTML or XML, with its proprietary interface known as the "pilot interface". IceBrowser supports JavaScript 1.5 as scripting language (through Rhino [32], the Mozilla ECMAScript implementation) and implements the DOM API provided by W3C[18] for direct Java access to the document (W3C standard DOM Level 2 Core). It also supports XML Namespaces and XSLT, CSS and XHTML 1.0.
This software is intended to be used with Java2 and supports Java Applets. Other important features are the HTTP 1.1 and HTTPS support and the Session Management capacity (Cookies).

Advantages

- Full Java Integration
- Java2 support
- HTML/XHTML/XML/XSLT support
- JavaScript + CSS + Applets support
- HTTPS

Drawbacks

- Commercial Product
- No support for external plugins

3.1.5 Jazilla and JRenderer [6][7]

Jazilla, thanks to the *JRenderer* Module, implements a Java version of Netscape's Navigator 5.0 source code (committed to free software and the public availability of source code). The *Jazilla* Source code is released under the Mozilla Public License (MPL), but still no final or beta release-date is planned: the product is still in development (actually it is a defunct and reborn project). Theoretically *Jazilla* should support HTML 4.01, XHTML, CSS1 and CSS2. The content representation should follow a defined data-flow: HTML-XML-XHTML (JTidy, SAX).
This product is meant to look like Mozilla, but it has no support for external interfaces (as *WebClient*); *Jazilla* supports only a very simple interface called "IRenderer". Rendering is achieved by using a hierarchy of Swing Components and utilizing only Java2.
The objective of the project is to build a Fast, Simple Design, Lightweight, 100% pure Java, Open-Source product.

Advantages

- Full Java
- XML/XHTML support
- Full HTML 4.01
- Java2
- Open-Source

Drawbacks

- Defunct Project
- Not usable
- No support for external web client

3.1.6 XBrowser [8]

XBrowser is a totally free and open-source Java application for browsing the web supporting Java2. It is Multithreaded and implements an History Mechanism, a Bookmarks and Page Contents mechanism, Import/Export Facility, Auto-Complete + Domain Completion Plugins. It supports Plugins and Extension, JavaScript and the Cookies. The whole configuration is done in XML and the interface is Multi-Language.

Advantages

- Free and Open-Source
- Supports Java2
- Supports JavaScript

- Configuration in XML
- Cookies and session management

Drawbacks

- No XML support
- Too sophisticated (History, bookmarks, printing, ...)

3.1.7 NetBrowser [9]

NetBrowser is a fully functional stand-alone Java browser which merges the *JRenderer* platform(section 3.1.5) and the *XBrowser* interface (section 3.1.6)

Advantages

- All the advantages of *JRenderer* and *XBrowser*

Drawbacks

- The project is still on work (No available software)

3.1.8 DocZilla [10]

DocZilla is a Mozilla-based SGML/XML/HTML- browser. It handles SGML, HTML, XML, CSS, DTD, XSLT and supports the use of JavaScript and XLinks . The software is free for personal, non-commercial use, the sources of Mozilla's components are available, the sources for *DocZilla* components not.

Advantages

- SGML/XML support
- JavaScript support

Drawbacks

- No sources available
- Not fully in Java developed

3.1.9 NetClue Browser [11]

NetClue is a fully-integrated, pure Java solution. Its components can be easily integrated to provide browser capabilities and rendering of XML, XHTML and HTML to any application. NetClue components support XML Namespaces, CSS, XSLT, JavaScript, DOM and SSL. This software also integrates a cache mechanism, printing, navigation and history facilities, Cookies and Session Management and is meant for using HTTP 1.1 with proxy capability. The product is presented with 3 different packages (Basic, Professional and X Edition) each one with its own supported technology.

Advantages

- XML, XHTML, XSLT support (X Edition)
- CSS, JavaScript and DOM support
- Plugins support
- Easy and extended Configuration capability
- Developer-friendly software solution
- Simple jar file to be included in the libraries
- Extended API

Drawbacks

- Commercial Product
- Full support only with the best package (X edition)
- No source code available

3.1.10 Calpa CalHTMLPane Java Component [12]

CalHTMLPane is a sub-component of a Java development package named *Calpa*, and is built to provide instant HTML access without the need to start up a native web-browser. *CalHTMLPane* allows Java application builders to incorporate HTML documents into a GUI display. Based on the Java Foundation Classes (better known as Swing), the *CalPane* supports a wide range of HTML3.2 and HTML4.0 tags so that documents can be authored with standard HTML editors and immediately displayed within a Java application.
The *CalHTMLPane* provides built-in document history and caching, fast parser and programmable API for customization.

Advantages

- Free to use and lightweight
- Easy to integrate (importable jar file)

- HTML 4.0 and partially XHTML support
- Easily extendable
- Fast rendering
- Extended API

Drawbacks

- No JavaScript support
- No CSS support
- No XML/XSLT support

3.1.11 Technology choice

After the analyze of the characteristics, advantages and drawbacks of the described technologies, it's now possible to take a decision about the browsing technology to be used. As we saw, there are a lot of candidate technologies that fulfil the requirements of the project, but there are also others which can not be taken into consideration.
On one hand, *HotJava* is "obsolete" and cannot be used in an environment using the recent versions of Java, the *WebClient* of *Mozilla* and *BrowserG!* are not browser independent (require a native browser installation), *Jazilla* is not working properly and is a defunct project, *XBrowser* is too sophisticated and doesn't fulfil the imposed requirements, *NetBrowser* is still in development and doesn't provide a stable release and *DocZilla* is not developed entirely in Java and doesn't provide sources or API. On the other hand *IceBrowser*, *NetClue* and *Calpa CalHTMLPane* seem to be a good choice for the project. *IceBrowser* and *NetClue* are more complete and provide a more enhanced environment, but are both commercial product; *Calpa*, instead, provides a more restricted interface, but is free-to-use and the provided interface can be easily extended to support i.e. XML.
Calpa CalHTMLPane is therefore the best choice among the current technologies, for displaying HTML and browsing between the slides of the implemented *Presentation Tool*.

3.2 Data Exchange

A very important issue in a distributed application is the exchange of data among the users of the application. The *Presentation Tool* is essentially based on the known internetworking paradigm, providing *unicast*, *multicast* and *broadcast* communication capabilities.
The JSDT toolkit for collaborative computing, provided by *Sun Microsystems*, has been selected so as to carry out the underlying interconnection between the users. The JSDT has been defined to support highly interactive collaborative applications, written in the Java programming language, and thus, it constitutes the appropriate solution.

3.2.1 Java Shared Data Toolkit

The ***Java Shared Data Toolkit***[1] software is a development library allowing developers to easily add collaboration features to applets and applications written in the Java programming language [19]. With the help of this tool a developer can integrate in network-centric applications (shared whiteboards, chat environments, presentation tools) a mechanism for easily communicating between the users.
For achieving its purpose, JSDT implements a multipoint data delivery service, providing the basic abstraction of a *Session* and supporting *full-duplex multipoint communication* among an arbitrary number of connected application entities, all over a variety of different networks [20]. JSDT provide also a token-based distributed synchronization mechanism, to ensure mutually exclusive access or atomic signaling.
Moreover JSDT provides also the ability for sharing byte arrays between the users connected to the same session.

A JSDT network needs a unique JSDT server in some known location (access-point) for managing the *Registry*. The JSDT Registry contains a transient database that maps names to JSDT objects: it is used to register new sessions and new users within a session and to retrieve the information about the connected users.
Each session can provide more communication channels, and each user logged into the session can subscribe one or more channels as consumer (it actually receives all the broadcast messages sent over this channel, all the messages directly addressed to him and can send uni-/multi-/broadcast messages to the users subscribed to the same channel).

A JSDT session can be initialized by using 3 different implementations each of them using its own protocol:

socket - uses TCP/IP Sockets

http - uses HTTP connections

lrmp - uses a Lightweight Reliable Multicast Protocol (LRMP)

The use of one of this protocols depends on the requirements of the users; i.e. generally a TCP/IP socket can not pass through most of the common firewalls, but an HTTP connection can usually do it.

Two different type of data can be sent over a channel: *Strings* and *JSDT Data Objects*; a JSDT Data Object is simply a Java object, serialized into the JSDT `Data` class.

The *Presentation Tool* utilizes JSDT as basis for the whole communication paradigm and a series of extensions of the basis JSDT classes for managing the different situations. For more detailed description of the used classes, please refer to Chapter 4

[1] For more information about JSDT, refer to [19] and [20]

Chapter 4

Design and Implementation

As defined in the Prototype (chapter 2), the *Presentation Tool* is entirely written in Java [14]. Java is an object-oriented (O-O) programming language and this implies an approach to programming in which the program is created from a series of items interacting with each other in pre-determined ways. The analysis and design of object-oriented software systems involves the identification of self-contained components which can be linked together to produce a complete application. These components achieve the encapsulation of data and function, the ability to inherit things from other components, and can communicate with each other by sending messages back and forth.
The entire design of the *Presentation Tool* is based on a predefined model, representing the functionality of the whole system. As a Graphics User Interface (GUI) application, it is actually utilizing some specific classes defined in the Java Core: the *Java Foundation Classes*, specifically the *Swing* Toolkit.
The *Java Foundation Classes* (JFC) are a set of Java class libraries provided as part of *J2SE* to support building GUI and graphics functionality for Java technology-based client applications. JFC include an extensive set of technologies that enable developers to create a rich, interactive user experience for client applications [21]. The *Swing* toolkit is a fully-featured GUI component library, implemented entirely in the Java programming language. J2SE's *javax.swing* package uses the windowing functionality of *AWT* (the *Abstract Window Toolkit* [22]) and the rendering capabilities of *Java 2D* [23] to provide sophisticated and highly extensible GUI components.

The most general class-structure of the *Presentation Tool* is represented in the diagram of Figure 4.1. In this diagram is possible to see how all the different classes interact among them.

4.1 Starting Point

The starting point of the whole application is the `startApplication` class. This class allows to start the application as a tutor or as a student, by specifying it at the command-line. `startApplication` constructs a new `myClient` instance: depending on the type of client to be constructed (`teacher` or `student`) the relative constructor is called.

Figure 4.1: General Class Structure Overview (UML)

The aforementioned process is very important and is one of the first statement of the `startApplication` class:

```
if(user.equals("Teacher")){
      client=new teacher(rendererType, null, hostname,
        port, connection, jsdtname, frame);
      ...
 }
 else{ //user is a student
      client=new student(rendererType, null, port,
        connection, jsdtname, frame);
      ...
}
```

From the UML diagram of Figure 4.1, should be clear that both classes `teacher` and `student` extend `myClient`. This means they both have a common core, but they are extended with some distinct features. Examples of these features on the teacher side can be the `setAccessPoint` method (sets the access point of the relative lecture), the slide navigation methods (`restart`, `getNext`, `getPrevious`) or the slide publishing method (`publish`)[1].

The `myClient` class implements 4 interfaces:

- `myBrowser`: describes the required methods for using the Browser (see section 4.2)
- `myChat`: describes the required methods for using the ChatLine (see section 4.2)
- `myWhiteBoard`: describes the required methods for using the WhiteBoard (see section 4.2)
- `Client`: required for the JSDT communication (see section 4.3)

The implementation of these classes allows the `myClient` class to build a platform on which initiate a *Distance Learning Session*.

4.2 The 3 main Components (ContenRenderer, ChatLine, WhiteBoard)

The *Presentation Tool* architecture is based on three components: a **ContentRenderer** (section 4.2.1), rendering the content of the tutor's presentation slides, a **ChatLine** (section 4.2.2), allowing the student and the teacher to communicate exchanging messages in a textual form, and a **WhiteBoard** (section 4.2.3), offering drawing capabilities, over the presentation content. These 3 components interact, by implementing the 3 relative interfaces (`myBrowser`, `myChat` and `myWhiteBoard`), defining the methods to be used for constructing a *Presentation Tool*. The base class is `myClient`, which implements all three interfaces, constructing an object able to use these three components.

[1]The detailed description of these and other methods can be found directly in the source code

Beyond the functionality definition of the components (the methods defined in the interfaces), `myClient`, using the three components, creates real objects to be integrated into the application:

- The `contentRenderer` interface
- The `drawingArea` class
- The `chatLine` class

These objects are therefore the GUI components which are inserted in the main application (`startApplication`), by recalling them from the `myClient` class (where they were actually created):

```
//Get the constructed Renderer from the client
JLayeredPane renderer=client.getLayeredPane();

//Create a scroller to contain the renderer
JScrollPane scroller= new JScrollPane(renderer);

//Get the Drawing Area for the Whiteboard
drawingArea dArea=client.getDrawingArea();

//Get the Chat for the Whiteboard
chatLine chat=client.getChat();

//Construct a new Panel containing the overlaying contentRenderer and drawingArea
JLayeredPane contentPanel=new JLayeredPane();
OverlayLayout layout=new OverlayLayout(contentPanel);

contentPanel.setLayout(layout);
contentPanel.setLayer(scroller, -1);
contentPanel.setLayer(dArea, 0);
contentPanel.add(scroller, -1);
contentPanel.add(dArea, 0);

JScrollPane mainScroller=new JScrollPane(contentPanel);

//Add the components to the main frame
frame.getContentPane().add(mainscroller, "Center");
frame.getContentPane().add(chat, "South");
```

A more detailed explanation of these elements is done in the relative sections (4.2.1, 4.2.2, 4.2.3).

4.2.1 ContentRenderer

The ContentRenderer is responsible for rendering the content of the presentation slides. The ContentRenderer component is a Java HTML browser reading the HTML version of the presentation's slides and presenting them by adding navigation functionality and presentation management (opening, uploading, downloading, ...). The whole functionality is essentially defined in the `myBrowser` interface:

```
package presentation;

import com.sun.media.jsdt.Data;
import javax.swing.JLayeredPane;
import myGui.presentation.manageButtons;
import myGui.presentation.navButtons;


public interface myBrowser{

  /**
   * Send data to the defined user over the Browser channel
   * @param to  The destination user(s)
   * @param data    The data to be sent
   */
  public void useBrowserChannel(String to, Data data);

  /**
   * Return the content Renderer
   * @return    The renderer
   */
  public contentRenderer getContentRenderer();

  /**
   * Return the renderer as a JLayeredPane
   * @return    The renderer (as a JLayeredPane)
   */
  public JLayeredPane getLayeredPane();

  /**
   * Return the navigation buttons
   * @return    The navigation buttons
   */
  public navButtons getNavButtons();

  /**
   * Return the management buttons
   * @return    The management buttons
   */
  public manageButtons getManButtons();
```

```
  /**
   * Open the next presentation's slide
   */
  public void getNext();

  /**
   * Open the previous presentation's slide
   */
  public void getPrevious();

  /**
   * Reload the current slide
   */
  public void reload();

  /**
   * Set the current presentation's slide
   * @param slide    The current slide number to be set
   */
  public void setCurrentSlide(int slide);

  /**
   * Get the current presentation's slide number
   * @return The current slide number
   */
  public int getCurrentSlide();

  /**
   * Open a new Presentation
   * @param path  The path of the presentation's directory
   * @param slide The slide number to be opened
   */
  public void openDoc(String path, int slide);

  /**
   * Set the presentation's name
   */
  public void setPresName(String presName);

  /**
   * Return the presentation's status
   * @return    True if the presentation is opened, False otherwise
   */
  public boolean isOpen();
}
```

A class which wants to implement the *Presentation Tool* have first to implement this interface.

After having defined the main functionality of the content renderer, as described before, we need a class defining a new object to be integrated into the application. The `contentRenderer` interface plays this role by allowing the use of different Content Renderers for displaying the presentation's slides. The real implementation of the methods which are using this interface is defined in another class implementing this interface.
The `contentRenderer` interface is defined as follows:

```
package presentation;

import javax.swing.JLabel;


public interface contentRenderer{
  /**
   * Open a new presentation
   */
  public String openDoc();

  /**
   * Open one of the slide of the presentation
   * @param path     The path where to find the presentation
   * @param slide    The slide number to be opened
   * @return         The presentation Name
   */
  public String openDoc(String path, int slide);

  /**
   * Set the current presentation's slide
   */
  public void setCurrentSlide(int slide);

  /**
   * Get the current presentation's slide
   */
  public int getCurrentSlide();

  /**
   * Show the previous slide
   */
  public void getPrevious(JLabel label);

  /**
   * Show the next slide
   */
  public void getNext(JLabel label);
```

```
  /**
   * Restart the presentation from the beginning
   */
  public void restart();

  /**
   * Reload the current slide
   */
  public void reload();

}
```

The methods just described are the essential methods to be implemented for running a presentation; all classes featuring this capability should implement this interface. In this version of the *Presentation Tool*, only one Content Renderer has been defined: the **CalpaHTMLPane**. The `myCalHTMLPane` class is extending the given rendering class (`CalHTMLPane`) and implementing the `contentRenderer` interface. In practice it acts as a bridge for translating the methods defined in the *Calpa* class to be used by the `contentRenderer` interface.
The architecture, anyway, is thought to be used in the future with other types of Content Renderers: a new Content Renderer class should only implement the `contentRenderer` interface and is ready to be used into the application.

The content renderer to be used is then selected, through a name-based selection mechanism, into the client creation phase (`teacher` / `student`), by passing to the constructor the `rendererType` String:

```
//Constructing the renderer, based on the browser's type
if(renderer.equals("Calpa")){
   CalHTMLPreferences prefs=new CalHTMLPreferences();
   prefs.setOptimizeDisplay(CalHTMLPreferences.OPTIMIZE_ALL);
   if(this.pane==null)this.pane =
          new myCalHTMLPane(prefs, new myCalHTMLObserver(label), null);
   if (url != null) {
      ((myCalHTMLPane)pane).showHTMLDocument(url);
   }
}
else if(renderer.equals("..."){

}

...

else{
   System.out.println("Renderer " + renderer + " not recognized! Exiting ...");
   System.exit(1);
}
```

4.2.2 ChatLine

The ChatLine is a text-based communication environment. It is giving the opportunity to the *Presentation Tool* users (tutor and students), to exchange messages in a textual form.

The core functionality of the ChatLine is described in the `myChat` interface:

```
package chat;

import com.sun.media.jsdt.Data;


public interface myChat{

  /**
   * Send data to the defined user over the Chat channel
   */
  public void useChatChannel(String to, Data data);


  /**
   * Return the chatLine
   * @return The chatLine
   */
  public chatLine getChat();

}
```

As evident from the source code, the only defined method is `getChat`, which just returns the newly created object.

The real functionality of this class is implemented in the more extended `chatLine` class: this class initializes all the GUI elements (Swing) and, by implementing the well-known Java `actionListener` interface, gives the users the possibility to select font colors, types and sizes and send the desired message, as a broadcast message, over the JSDT channel (section 4.3 for more details about this process).

This class defines two important methods. One of them is the `sendButton_mousePressed` method, which sends over the JSDT channel the written message by encapsulating it in a JSDT `Data` object and using the chat channel (`useChatChannel`) for the transmission to the destination (the String "all" is defined as an alias for broadcasting the message over the channel).
The class defines also a `receiveDataChat` method, called by the chatConsumer (see section 4.4), for allowing the textArea of the chat panel to receive and display the messages sent by other users:

```
package chat;

import client.myClient;
...

public class chatLine extends JPanel{

...

 /**
   * MouseListener, called when the send button is pressed:
   * sends the message broadcast
   */
  void sendButton_mousePressed(MouseEvent e) {
   int temp;
    if (wbu.getDrawtype() == "Text"){
        //The text will be displayed into the drawingArea of the Whiteboard object
    }
    else if ( modeSelected.equals("ChatMode") ){
    sendString = editorPane.getText();
    if(!sendString.endsWith("\n")) sendString=sendString + "\n";
    String action=String.valueOf(wbu.getDrawcolor().getRGB()) + " "
                          + editorPane.getFont().getStyle() + " "
                          + editorPane.getFont().getName()+ " "
                          + editorPane.getFont().getSize() + " "
                          + editorPane.getText() + " ";
    //sends the chat message to all subscribed users
    wbu.useChatChannel("all", new Data(action));
        try {//Display the chatUserName in the appropriate color.
                if(wbu.getName().equals("Teacher"))
                   StyleConstants.setForeground(userStyle, Color.RED);
                else StyleConstants.setForeground(userStyle, Color.blue);
                StyleConstants.setFontSize(userStyle, 10);
                doc.insertString(0, wbu.getName() + ":> ", userStyle);

                //Display the text.
                defaultStyle = getFontStyle(wbu.getDrawcolor(),
                editorPane.getFont().getStyle(), editorPane.getFont().getName(),
                    editorPane.getFont().getSize());
                doc.insertString(wbu.getName().length()+3 , sendString, defaultStyle);
                editorPane.setText("");
                chatTextArea.setCaretPosition(0);
        } catch (BadLocationException ble) {
            System.err.println("Couldn't insert chat text.");
        }
    }

  }
```

```
...

  /**
   * Called by the chatConsumer class:
   * receives the data to be used for the chat,
   * displays and formats them in the right way
   */
  public synchronized void receiveDataChat(String receivedata) {
      StringTokenizer tok = null;
      String clientName = null;
      String chatString = null;
      String drawcommand = null;
      String temp = null;

      while (receivedata != null) { //Create the tokenizer
          tok = new StringTokenizer(receivedata, " \r\t");
          receivedata = null;
          clientName = tok.nextToken();
      }
      try {//Display the chatUserName in the appropriate color.
                if(clientName.equals("Teacher"))
                   StyleConstants.setForeground(userStyle, Color.RED);
                else StyleConstants.setForeground(userStyle, Color.blue);
                StyleConstants.setFontSize(userStyle, 10);
                doc.insertString(0, clientName + ":> ", userStyle);

                //Display the text.
                receivedStyle = getFontStyle(Color.decode(tok.nextToken()),
                                Integer.parseInt(tok.nextToken()),tok.nextToken(),
                                Integer.parseInt(tok.nextToken()));
                String text="";
                while(tok.hasMoreTokens()){
                  text=text + " " + tok.nextToken();
                }
                doc.insertString((String.valueOf(clientName)).length()+3 ,
                text, receivedStyle);

                editorPane.setText("");
                chatTextArea.setCaretPosition(0);
        } catch (BadLocationException ble) {
            System.err.println("Couldn't insert chat text.");
      }
    }

 ...

}
```

This object, as the other GUI-based objects, will be inserted as a Swing `JPanel` into the *Presentation Tool* GUI.
The aforementioned method is the most important of the whole class, because it really enables the communication between the users connected to the JSDT channel.

4.2.3 WhiteBoard

The WhiteBoard is an extension of the presentation's slides, in the sense it can extend them by adding some extra-features: the possibility to trace a line, to draw, to write some comments or to move a pointer over the slides. This is done (in the `startApplication` class) by placing a transparent layer over the presentation slide (actually by using the AWT `OverlayLayout` class).

```
// Construct a new Panel containing the overlaying
// contentRenderer and the drawingArea

JLayeredPane contentPanel=new JLayeredPane();
OverlayLayout layout=new OverlayLayout(contentPanel);
contentPanel.setLayout(layout);
contentPanel.setLayer(scroller, -1);
contentPanel.setLayer(dArea, 0);
contentPanel.add(scroller, -1);
contentPanel.add(dArea, 0);

JScrollPane mainScroller=new JScrollPane(contentPanel);
```

This transparent layer is always highlighted and has no actual interaction with the below-layered presentation: they are 2 distinct objects, each of them working alone. The result of merging these two objects is surprisingly: it simulates a real interaction between the whiteboard and the slides.

From a more technical point of view, the WhiteBoard functionality is described (as for the chatLine and the contentRenderer) in an interface, specifically the `myWhiteBoard` interface:

```
package whiteboard;

import com.sun.media.jsdt.Data;
import java.awt.Color;
import myGui.whiteboard.whiteBoardButtons;

public interface myWhiteBoard{

 /**
   * Send data to the defined user over the Whiteboard channel
   * @param to  The destination user(s)
   * @param data    The data to be sent
```

```
    */
    public void useWBChannel(String to, Data data);

    /**
     * Get the whiteboard management buttons
     */
    public whiteBoardButtons getWButtons();

    /**
     * Return the whole drawing area
     */
    public drawingArea getDrawingArea();

    /**
     * Set the sender mode (use of the token)
     */
    public void setSenderMode(boolean sender);

    /**
     * Return the senderMode status
     */
    public boolean isSenderMode();

    /**
     * Get the color set for the drawing area
     */
    public Color getDrawcolor();

    /**
     * Set a new color for the drawing area
     */
    public void setDrawcolor(Color color);

    /**
     * Get the last used drawingType
     */
    public String getDrawtype();

    /**
     * Set the drawingType to be used
     */
    public void setDrawtype(String type);

    /**
     * Format and write the jsdt messages on the Whiteboard channel
     */
    public void writeLine(String line);
}
```

This interface defines the core methods to be implemented in order to use the WhiteBoard into a real application.
The `writeLine` method, in particular, is very important in this class: in the same way as the `sendButton_mousePressed` method of the chatLine, this method, which is used in the `drawingArea` class and implemented in the `myClient` class, allows the whiteboard events to be sent as a line (as a String), over the whiteboard channel to the subscribed users (broadcast over the channel):

```
/**
* Write a message on the Whiteboard channel
* (invoked by the whiteboardConsumer class)
*/
public void writeLine(String line) {
  if (isConnected) {
    String message=this.jsdtName + " " + line;
      try {
          data = new Data(message);
          data.setPriority(Channel.HIGH_PRIORITY);
          useWBChannel("all", data);
      } catch (Exception e) {
          System.err.print("Caught exception in ");
          System.err.println("WhiteBoardUser.writeLine(): " + e);
          System.err.flush();
          e.printStackTrace();
      }
  }
}
```

The `drawingArea` class constructs a new object to be incorporated into the application as a whiteboard. It initializes all the GUI components and listens for actions over it (by implementing the `MouseListener` and the `MouseMotionListener` interfaces). This class implements a lot of important methods, but for obvious reasons, it is impossible to present all of them[2]. Three of them are anyway very important: the `mouseReleased`, the `receiveData` and the `paintComponent` methods.

```
package whiteboard;

import client.myClient;
...

public class drawingArea extends JPanel
   implements MouseListener, MouseMotionListener{

...
    //Implementing the interface: action on mouse release
```

[2]For a better explanation take a look to the documentation in the Javadoc

```
    public void mouseReleased(MouseEvent event) {
      if(wbu.isSenderMode()){
       mouseReleased=true;
       String drawtype=wbu.getDrawtype();
       System.out.println(drawtype);
       int shapeCount = shapeList.size();
       if(offScreenGraphics!=null && s!=null) s.draw(offScreenGraphics);

       if (drawtype== "Rect") {
          wbu.writeLine("Rect " + String.valueOf(drawcolor.getRGB()) +
                " " + anchorPoint.x + " " + anchorPoint.y +
                " " + event.getX() +  " " + event.getY() + "\n");
          repaint();
       }
       else if (drawtype == "Oval") {
          wbu.writeLine("Oval " + String.valueOf(drawcolor.getRGB()) + " " +
                 anchorPoint.x + " " + anchorPoint.y + " " +
                 event.getX() + " " + event.getY() + "\n");
          repaint();
       }

       else if(drawtype == "..."){
         ...
       }

       ...

      }
    }

...

   // Processing of received data
    public synchronized void receiveData(String receivedata) {
      StringTokenizer tok = null;
      String clientName = null;
      String textString = null, fontTextName=null;
      int fontTextStyle=0, fontTextSize=10;
      int shapeListIndex;

      while (receivedata != null) {
            tok = new StringTokenizer(receivedata, " \n\r\t");
            receivedata = null;
            clientName = tok.nextToken();
            drawcommand = tok.nextToken();
            receivedColor = Color.decode(tok.nextToken());
            px1 = Integer.parseInt(tok.nextToken());
            py1 = Integer.parseInt(tok.nextToken());
```

```
            if(drawcommand.equals("Text")){
                  textString = tok.nextToken();
                  fontTextStyle = Integer.parseInt(tok.nextToken());
                  fontTextName = tok.nextToken();
                  fontTextSize = Integer.parseInt(tok.nextToken());
            }
            else{
                  px2 = Integer.parseInt(tok.nextToken());
                  py2 = Integer.parseInt(tok.nextToken());
            }
      }

      receivedData = true;

      if (drawcommand.equals("Rect")) {
              s = new HollowRectangle();
              s.color = receivedColor;
              s.boundsBox = new DragRect();
              dragRect = s.boundsBox;
              dragRect.setBounds(px1,py1,px2-px1,py2-py1);
              dragRect.normalize();
              shapeListIndex= shapeList.size();
              shapeList.insertElementAt(s, shapeListIndex);
              ((Shapes)shapeList.get(shapeListIndex)).color = receivedColor;
              repaint();
      }

      else if(drawcommand.equals("...")){
        ...
      }
      ...
    }

...

    //Method called automatically to refresh the drawing area
    public synchronized void paintComponent(Graphics g) {

      super.paintComponent(g);

      Graphics2D g2d = (Graphics2D)g;

      int j=0;
      while(j<shapeList.size()){
            j++;
      }
     //System.out.println(j + " elements in the Shape List");
```

```
Dimension d = getSize();

if((offScreenGraphics == null) ||
   (d.width != offScreenDimension.width) ||
   (d.height != offScreenDimension.height)) {

     offScreenDimension = d;

     offScreenImage = new BufferedImage(d.width, d.height,
       BufferedImage.TYPE_INT_ARGB);
     offScreenGraphics=offScreenImage.createGraphics();
     offScreenGraphics.setRenderingHint(
       RenderingHints.KEY_ALPHA_INTERPOLATION,
       RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY);
     offScreenGraphics.setComposite(
       AlphaComposite.getInstance(AlphaComposite.CLEAR, 0.0f));
     Rectangle2D.Double rect =
       new Rectangle2D.Double(0,0,d.width, d.height);
     offScreenGraphics.fill(rect);
     offScreenGraphics.setComposite(AlphaComposite.SrcOver);
     offScreenGraphics.setStroke(new BasicStroke(5.5f));
     g2d.setStroke(new BasicStroke(5.5f));
 }

 if (offScreenImage != null) {
    g2d.drawImage(offScreenImage, 0, 0, this);
 }

 if (receivedData){
     System.out.println("Drawing " + drawcommand);

     if (drawcommand.equals("Rect") ||
         drawcommand.equals("Oval") ||
         drawcommand.equals("Line") ||
         drawcommand.equals("FilledRect") ||
         drawcommand.equals("FilledOval") ||
         drawcommand.equals("Text") ){
           receivedData = false;
           s.draw(offScreenGraphics);
     }
     g2d.drawImage(offScreenImage, 0, 0, this);
     if (drawcommand.equals("Pointer"))
       g.drawImage(image1,currentPointX-5,currentPointY-5,this);
     if (drawcommand.equals("Delete")){
     for(int i=0; i<shapeList.size(); i++) {
        s = (Shapes) shapeList.elementAt(i);
        drawcolor = null;
        s.draw(offScreenGraphics);
```

```
            }
           }
      }

      if (!receivedData){
           String type=wbu.getDrawtype();
           if (type == "Rect"  || type == "Oval" || type == "Line" ||
               type == "FilledRect" || type == "FilledOval"){
              g2d.setStroke(new BasicStroke(5.5f));
              s.draw(g2d);
           }
           if (type == "Pointer"){
               g2d.drawImage(image1,currentPointX-5,currentPointY-5,this);
           }
           if (type == "Clear"){
              g2d.drawImage(offScreenImage, 0, 0, this);
           }
           if (type == "Text"){
              s.font = fontText;
              s.color = drawcolor;
              s.draw(g2d);
           }
           if (type == "Delete"){
             for(int i=0; i<shapeList.size(); i++) {
                s = (Shapes) shapeList.elementAt(i);
                s.draw(offScreenGraphics);
             }
           }

           if (mouseReleased) g2d.drawImage(offScreenImage, 0, 0, this);
      }
    }
}
```

The `mouseReleased` method is called when the mouse is released, that is when a new object has been designed onto the drawing area. At this time, the method constructs a new String and sends it out on the whiteboard channel by using the `writeLine` method described above. The newly constructed string is different for every selected type of drawing, so that the destination drawingArea can recompose the drawn element and draw it also locally. The recomposing process is done by the `receiveData` method: this method is synchronized for keeping the right sequence of events and constructing a new object (for example an `HollowRectangle`) for each different received type of drawing, when invoked by the `whiteBoardConsumer` (see later in section 4.4). The color, the size and all attributes of the figure sent as a String by the `mouseReleased` class are here extracted and assigned to the newly constructed object.
The last method, `paintComponent`, is called automatically by Swing each time the method `repaint()` is called; it just repaints the whole drawing area, by actualizing the view to the current state (adding new drawings and deleting the deleted ones). This process has been

implemented by using a technique called *Double Buffering*: composing the image off-screen and then drawing the buffered image on the screen. This technique avoids annoying flicker that results from drawing and erasing everything on-screen. This is accomplished by using the `BufferedImage` and the `Graphics2d` classes of AWT.

4.3 JSDT

The Java Shared Data Toolkit is a very important component of the whole application, because it constitutes the medium for intercommunication among the users.
As mentioned in chapter 3, thanks to JSDT is possible to join sessions and subscribe to channels. Sessions and channels are first initialized by the `myClient` class through the `jsdtServer` class:

```
/**
* Start a new jsdt registry server (if the client is a teacher),
* create/joinsa jsdt session, start/join the Browser, Whiteboard
* and Chat channel and subscribe (addComsumer) to these channels.
*
*/
public void connect(){

  //Setting all the action done to false
  boolean registryStarted=false;
  boolean sessionStarted=false;
  boolean bConsumerAdded=false;
  boolean wConsumerAdded=false;
  boolean cConsumerAdded=false;

  if(this.isConnected) {
    String label="You are already connected!";
    warningDialog dialog=new warningDialog("Warning", label);
  }
  else{
    //Create a new Connection dialog to write the messages in
    connectionDialog dialog=new connectionDialog(this);

    //Create a new jsdt server to initialize the connection
    server = new jsdtServer(this,dialog);

    //Start the registry
    if(isTeacher) registryStarted = server.startRegistry();
    else registryStarted=true;

    //Start/Join the session
    if(registryStarted)
      sessionStarted=server.startSession("TikSession");
```

```
    //Create/join the 3 channels
    if(sessionStarted){
      bChannel=server.createChannel("browser");
      wChannel=server.createChannel("whiteboard");
      cChannel=server.createChannel("chat");
    }

    //Add the client to the 3 channels as consumer
    if(bChannel!=null && wChannel!=null && cChannel !=null){
      jsdtClient bClient=new jsdtClient(this, bChannel, "browser", dialog);
      bConsumerAdded=bClient.addConsumer();
      jsdtClient wClient=new jsdtClient(this, wChannel, "whiteboard", dialog);
      wConsumerAdded=wClient.addConsumer();
      jsdtClient cClient=new jsdtClient(this, cChannel, "chat", dialog);
      cConsumerAdded=cClient.addConsumer();
    }
    if(bConsumerAdded && bConsumerAdded && cConsumerAdded){
      this.isConnected=true;
      disableButtons();
    }
    dialog.enable(true);
  }
}
```

The called `jsdtServer` class implements essentially three distinct tasks with three distinct methods:

- `startRegistry`: Starts the JSDT registry
- `startSession`: Starts a new session into the JSDT community
- `createChannel`: Creates a new channel based on the type given as a parameter

```
package jsdt;

import client.myClient;
...

public class jsdtServer{

...

  /**
   * Start the registry for the jsdt session
   * @return True/False depending if the registration is ok
   */
  public boolean startRegistry(){
```

```
  try {
    RegistryFactory.startRegistry(connection);
    String label="Registry started";
    dialog.add(true, label);
    return true;
  }catch (NoRegistryException nre) {
    String label="Couldn't start a Registry of this type.";
    dialog.add(false, label);
    System.out.println("Registry exception: " + nre.getMessage());
    return false;
  }catch (RegistryExistsException ree) {
    String label="The Registry was already running.";
    dialog.add(true, label);
    return true;
  }
}


/**
 * Start a new session and register it on the Registry server
 * @return  True/False depending if the session starting is ok
 */
public boolean startSession(String sessionName){
  try {/* Resolve the session. */
     URLString url =
       URLString.createSessionURL(hostname, port, connection, sessionName);
     session = SessionFactory.createSession(this.client, url, true);

     /*  Need to setup a unique number for this client.
      *  A simple cheap way is to determine how many clients
      *  are joined to the session, and use that.
      */

    clientNames = session.listClientNames();
    clientNo    = clientNames.length;
    String label="JSDT Session opened";
    dialog.add(true, label);
    return true;
  }catch (Exception ex) {
    String label="Unable to connect the client! Session not yet Opened";
    dialog.add(false, label);
    client.getNavButtons().resetCourse();
    System.out.println("Session exception: " + ex.getMessage());
    return false;
  }
}
```

```
  /**
   * Create a new channel of a defined type to be used for sending jsdt messages
   * @param type  The type of channel to be created (Name)
   * @return True/False depending if the channel creation is ok
   */
  public Channel createChannel(String type){
    try{
      Channel channel = session.createChannel(this.client, type, true, true, true);
      String label=type + " channel opened";
      dialog.add(true, label);
      this.channels.add(channel);
      return channel;
    }
    catch(Exception ex){
      String label="Unable to open " + type + " channel";
      dialog.add(false, label);
      System.out.println(type + " channel exception: " + ex.getMessage());
      ex.printStackTrace();
      return null;
    }
  }

...

}
```

After the initialization done by the `jsdtServer`, each user can now connect itself to the session and subscribe to the channels, by calling the same methods (`startSession` and `createChannel`); these methods, if a session with the same name already exists in the given registry, just join the existing session. The same mechanism applies to the channels.

4.4 JSDT Channel Consumers and Data Flow

After joining a channel, a user should subscribe itself to this channel to receive notifications (messages) sent over it. This is done in the `jsdtClient` class, using the *Channel Consumers*.

```
package jsdt;

import chat.chatConsumer;
...

public class jsdtClient{

  public jsdtClient(myClient client, Channel channel,
       String type, connectionDialog dialog) {
    this.client=client;
```

```
    this.channel=channel;
    this.dialog=dialog;
    this.type=type;

    if(type.equals("browser"))
       this.consumer=new browserConsumer(client);
    else if(type.equals("whiteboard"))
       this.consumer=new whiteBoardConsumer(client);
    else if(type.equals("chat"))
       this.consumer=new chatConsumer(client);
  }

  /**
   * Add a new consumer of the given type to the defined channel
   */
  public boolean addConsumer(){
    try{
      channel.addConsumer(client, consumer);
      String label=this.type + " consumer added";
      dialog.add(true, label);
      return true;
    }
    catch(Exception ex){
      String label="Adding " + this.type + " consumer failed";
      dialog.add(false, label);
      System.out.println(this.type + " consumer exception: " + ex.getMessage());
      return false;
    }
  }
}
```

The `jsdtClient` creates a new consumer for each of the existing channels (browser, chat and whiteBoard channels), by constructing the relative object:

- `browserConsumer`
- `chatConsumer`
- `whiteBoardConsumer`

These three consumers implement the JSDT `ChannelConsumer` interface, by defining the `dataReceived` method. This method is called each time a new JSDT `Data` object has been sent over the channel. The three consumers have different behavior in relation to the Data received from the channel; anyway all of them make a first distinction between the user receiving the message: is it a teacher or a student?
As an example an extract of the `browserConsumer` class:

```
package presentation;

import client.myClient;
...

public class browserConsumer implements ChannelConsumer{

  /**
   * Construct a new consumer and set the global variables
   * @param client   The client using this consumer
   */
  public browserConsumer(myClient client){
    this.client=client;
    if(client instanceof teacher) this.isTeacher=true;
  }

 /**
   * Invoked automatically when a user subscribed to this consumer
   * get a new data over the channel
   */
  public synchronized void dataReceived(Data data){
      if(isTeacher){//teacher defined actions
       ...
      }
      else{//student defined actions
        ...
     }
  }
}
```

4.5 GUI and Utilities

Apart from the core application classes there are a number of utility classes defined for developing the GUI and the numerous specialities of the *Presentation Tool*. An overview of these classes can be seen in figure 4.2.

All the listeners are just an implementation of the AWT class `MouseListener`: in practice they are listener for a specific type of objects (menu, dialog, ...).

The dialogs extend the `JDialog` Swing class for having more structured dialog windows for some frequent events (errors, warnings, ...).

The buttons and panel classes extend the `JPanel` Swing class for creating different button panels for the various part of the GUI.

The 2 distinct menu classes (one for the teacher, one for the student) extend the Swing `JMenuBar` class by specifying the correct menu items for the two type of users.

Figure 4.2: GUI and Utilities Class Structure Overview (UML)

Finally there are some utilities like the `myFTPClient` extending the `FTPClient` class provided by *enterprisedt.net.ftp* (to add ftp facilities to the application), the `httpReader` class (performing an HTTP connection and reading the output), the `fileParser` class (for reading the configuration files by parsing their content) and the `desEncrypter` class (used to encrypt the ftp password saved into the configuration files). These classes help to build the whole application, but are just utilities and will not be further discussed here[3].

[3]For more information look at the Javadoc documentation of the single classes

Chapter 5

Conclusion and Future Work

The main goal of this thesis was to develop a *Presentation Tool* to be integrated in a *Synchronous Distance Learning System* (SDLS). The developed application represents such a tool, and particularly it is a stand-alone Java application which can be used as a Tool for presentations in an online distance lecture.
The tool allows the management of the presentation by the tutor, the interaction with the connected users and focuses on the intercommunication capabilities between tutor and students.
The innovative idea of an overlaying, common whiteboard, improves the interaction, transforming the course in a new collaborative experience for building a real learning environment.

The offered functionalities fulfil the requirements imposed by a distance online lecture and all the main tasks have successfully been implemented. But, of course, there are some small details which are still in development, or needs to be implemented. These tasks can be seen as future stimulus for extending and improving the implemented application:

- At the current status, the integrated renderer is only capable of showing HTML documents and partially XHTML files. But it is very easy to integrate into the *Presentation Tool* a new renderer. For these reasons an essential improvement is the integration of a new renderer capable to show XML documents (by transforming them through XSLT) or a Flash Player for animations and dynamical presentations.

- By inserting new renderers it will be possible to implement a multi-renderer mechanism, for allowing the different users to select the preferred renderer. The basis of this mechanism is already implemented into the application and should just be extended with the real implementation of the various rendering possibilities.

- The overlaying mechanism through which the whiteboard is working is not optimal. In some distinct situations the interaction with the Swing refreshing mechanism is not working. A new mechanism, or an improvement of the current one, is required.

- The authentication mechanism is currently very trivial and the FTP login is not satisfying strictly all the security aspects. The integration of the *Presentation Tool* into the extended *Conference Control Protocol* (CCP), is essential. In this way the

students will have real accounts and their personal information will be transfered in a secure way (i.e. Secure FTP).

- The user management on the tutor side is very important. At the current status, the tutor can only control the interaction of the students with the whiteboard. A more extensive controlling and management mechanism is important and should be developed in the future.
- The *Presentation Tool* is based, by definition, on the presentation of a sequence of files exported from a presentation software like Microsoft's *PowerPoint*. An interesting feature is to integrate into the tool the possibility to open directly a *PowerPoint* file, which will be then transformed in a sequence of HTML files, XML files or in an animation (Flash) depending on the used renderer.
- The implemented courses' status publication via the access-point and the IP addresses is just a proposal. A more detailed, extended and secure mechanism for manage the various courses is advisable.

As final remark, the aim of this thesis has to be analyzed: why do we need such a tool? Is really important to build an application like this?
In the last years everything is getting faster and the world is getting smaller. This means that there are people that either are moving from their home location frequently, or they are not wishing to change their home location for educational reasons. In both cases, the need of a *Presentation Tool* as a component in an educational delivery platform is therefore clear: why do we have to move people when is possible to move information?
Unfortunately the present technologies are not yet presenting a complete working product for offering the experience of a real course, being away. This lack has to be solved.
The implemented *Presentation Tool* is not meant to fulfil all the needs of such a situation, but is just a first step, from which we can start building a new educational environment.

Appendix A

User Guide

In this small chapter we will focus on the requirements, the installation, the configuration and the use of the *Presentation Tool*.
This is not the final version of the application, so remember that it is possible to find some small bugs or errors.

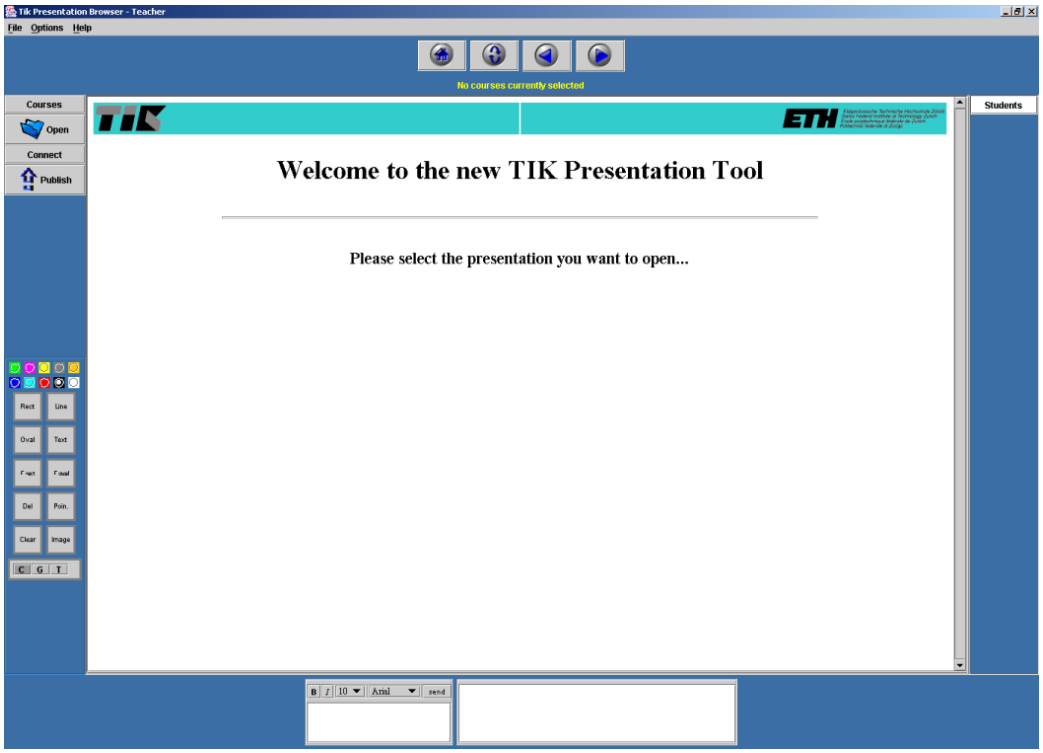

Figure A.1: The GUI of the Presentation Tool

A.1 Requirements

General Requirements

- Running Java Virtual Machine (JVM) implementing the **j2sdk1.4** Java version [24]
- Internet connection
- Some space on the hard disk to save the downloaded presentations

Technichal Prerequisites

- Running Web Server and FTP Server to be used for the course registering:
  - The student should have access to the web server: the server name and the path to the access-point's index page should be saved into the *config/accessPoint.xml* file of the student application (it can be done through the GUI).
  - The teacher should have access to the web server and to the ftp server to change the location (IP Address) of the given course: the server name, the web path, the ftp path, the ftp server's login and the ftp server's password should be saved in the *config/accessPoint.xml* file of the teacher application (it can be done through the GUI).
  - For each course, on the ftp server you must provide a file called *NameOfThe-Course.text* which will contain the current IP address of the given course.
  - The *index.php* file in this directory shows all the given courses followed by the current IP address or *null* if the course is off-line.
- Running FTP Server to publish/download the presentation files: each user (teacher and student) should have access to this server on the same account with different login and password (should be provided to the student when subscribing to the course). This server acts therefore also as an authentication service. The presentations can be published through the GUI to the selected FTP Server, but they must be encapsulated in a directory named as the presentation.
- Mechanism for exporting presentations to HTML files. The subsequent files should be ordered alphabetically, so that each slide can be automatically loaded when pressing the next (previous) button.

A.2 Installing, configuring and starting the Presentation Tool

To start the presentation tool as a user, you have to be connected to the internet and install Java on your system. The Java installation process is described in [25].

After that let the class *startApplication* run (`java startApplication`), specify the class-path (by including the "classes" directory, the "lib/calpahtml.jar", the "lib/ftp.jar" and the "lib/jsdt.jar" libraries) and the type of user you want to run (-Student/-Teacher). Alternatively you can use the included scripts for starting a Teacher or a Student on a Windows or Linux environment.

A.3 Using the Presentation Tool

When starting the *Presentation Tool*, an application like the one represented in Figure A.1 will be opened. This is the starting point for managing a presentation (as a teacher) or following a lecture (as a student).
Keep in mind that you are not the only person using this tool at the same time, so try to be as fair as possible in using the whiteboard and the chat.

In Figure A.2, is possible to see all the basic application's components.

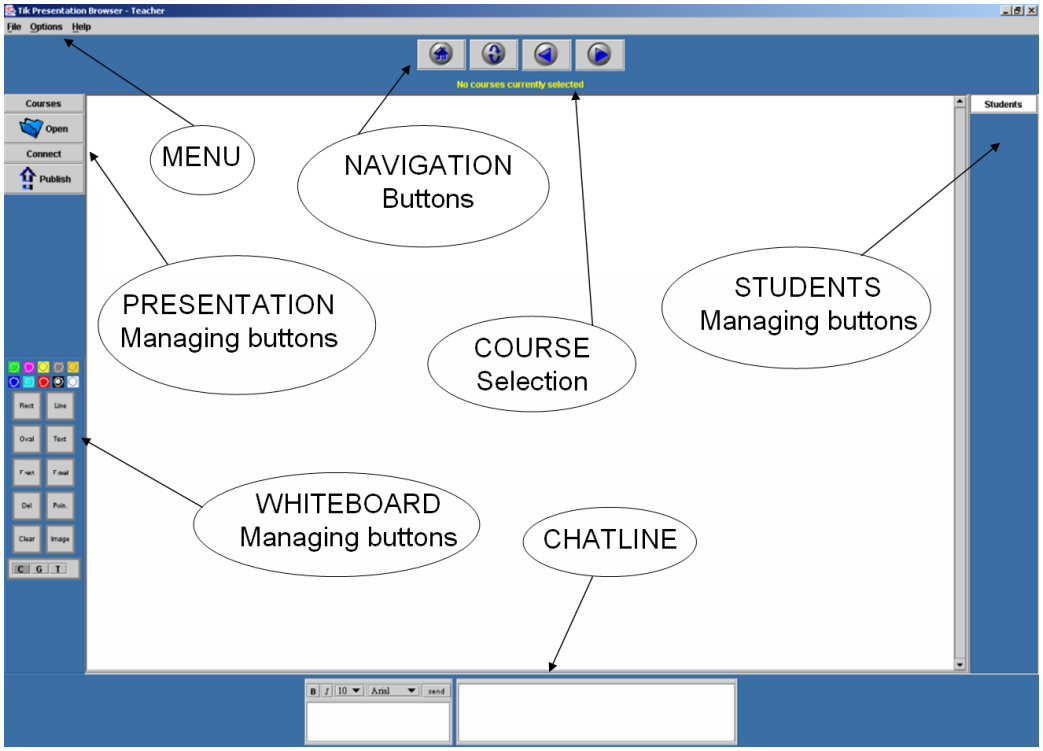

Figure A.2: The Components of the Presentation Tool

Depending on how you use the application (as a Tutor or as a Student) you should follow a different sequence of steps.

A.3.1 Tutor

1. **Selecting a course to be given**

   To select a course you have basically to click on the COURSES button on the top left of the application window. You also have the choice to select the course by going in the menu to *File - Select Course* (see Figure A.3).

   - Click the *Courses* button on the top-left of the window
   - The default access-point is contacted and the course list is retrieved
   - A new dialog window with a list of courses will appear

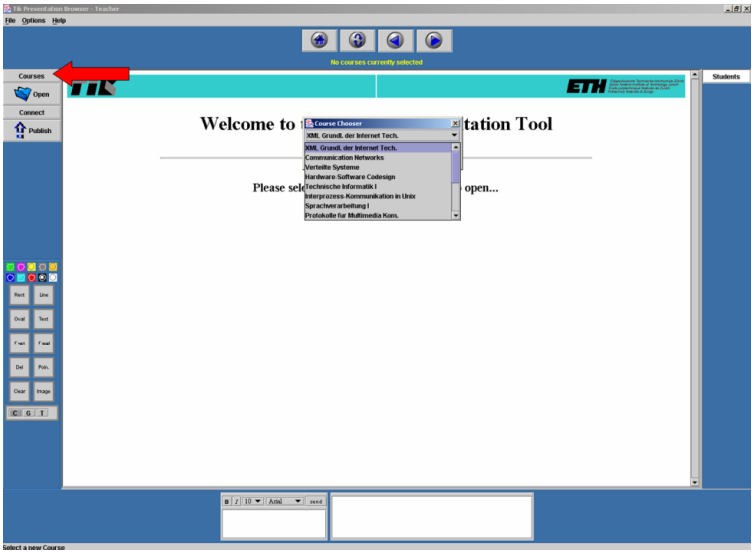

Figure A.3: Selecting a course (teacher side)

- Select the course you want to give and click *choose*
- The selected courses will be updated at the default access-point and the current IP address will be set
- On the center of the upper panel (on the navigation bar) appears the name of the selected course

2. **Publishing a new Presentation**

   To publish a presentation click the PUBLISH button in the button list on the left size of the window or go to the File menu and click *Publish* (see Figure A.4).

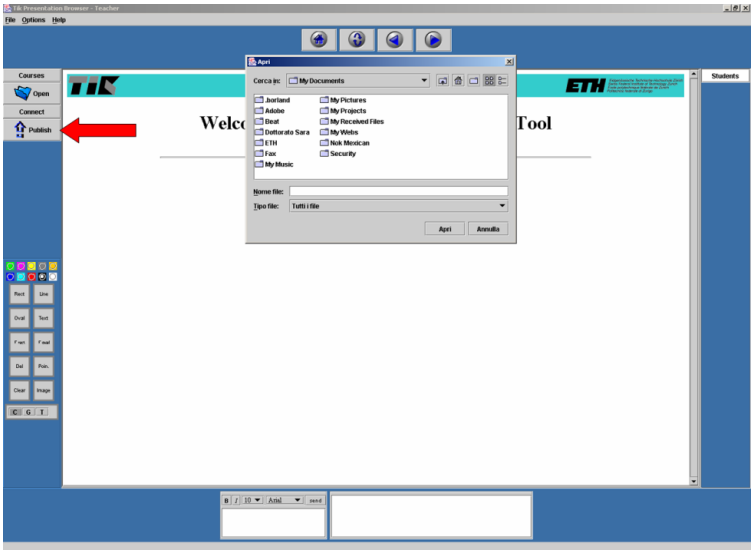

Figure A.4: Publishing a new presentation (teacher side)

- Click the *Publish* button on the top-left side of the window
- An ftp dialog window appears; select the ftp server where do you want to put your presentation or write a new one in the field. To add or delete servers from this list go to the Options menu and select *FTP Servers*. To log into the FTP server, in the appeared dialog window you also have to provide your username and password
- Insert all values and click *Connect*
- The connection is in progress, if everything is ok, a message will advise you. If you are logged in, click *Publish* to publish your presentation
- A new dialog window appears inviting you to select the directory containing your presentation; the presentation must be saved in a sequence of files ordered alphabetically and saved in a directory with the name of the presentation
- Select the PRESENTATION DIRECTORY you want to publish and click *Open*
- If a presentation with the same name already exists on the server, it will be overwritten by this new one. A message will advise you if everything is going well

3. **Opening a new Presentation**

   To open a new presentation press the OPEN button on the top-left of the window, just below the "Courses" button. Alternatively you can go to the File menu and select *Open presentation* (see Figure A.5).

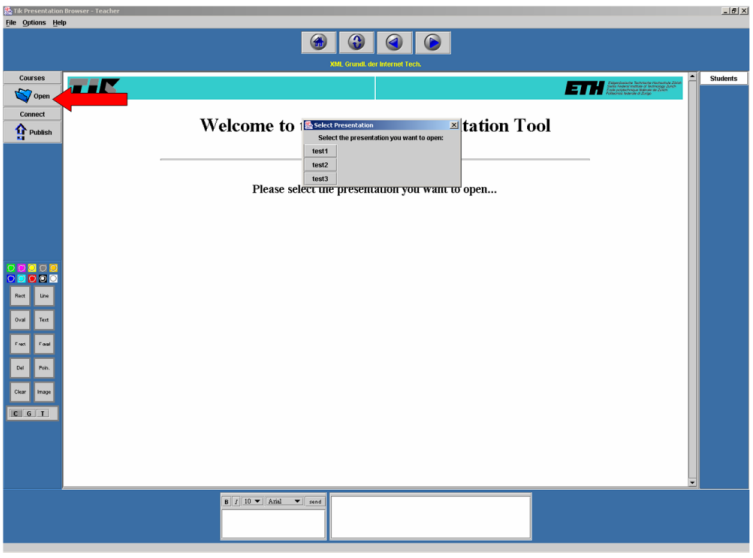

Figure A.5: Opening a new presentation (teacher side)

- Click the *Open* button
- A new dialog Window will appear; here you can choose the ftp server from where you want to download your presentation or writing a new one just inside. To add or delete servers from this list go to the Options menu and select *FTP Servers*. To log into the FTP server, in the appeared dialog window you also have to provide your username and password

- Insert all values and click *Connect*
- The connection is in progress, if everything is ok, a message will advise you. If you are logged in, click *Open new Presentation* to open your presentation
- A new dialog will appear with the list of presentation saved on this server. Select one of them.
- The presentation is loaded and the first slide will be shown in the browser's main window

4. **Opening a new Connection**

   To open a new connection and begin giving a course click the CONNECT button in the button list on the left size of the window or go to the File menu and click *Connect* (see Figure A.6).

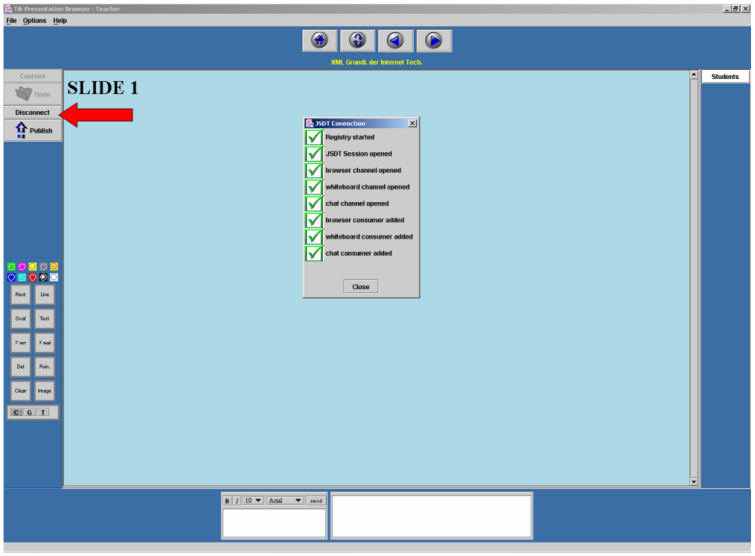

Figure A.6: Opening a new connection (teacher side)

- Click the *Connect* button to initialize a new connection
- After a while will appear a new dialog window confirming you that you are successfully connected
- Close the window and look at the "Connect" button: now a button click will Disconnect you from the session
- If you disconnect, the presentation will be closed and all participants will be removed from the session

5. **Managing the users**

   The panel on the right side, contains the list of all students currently connected to this session.
   By clicking on the selected student you can manage him (see Figure A.7).

   When a new user is connected to the session, its name will be displayed on the panel with a Cyan Background. Different background colors have different meanings:

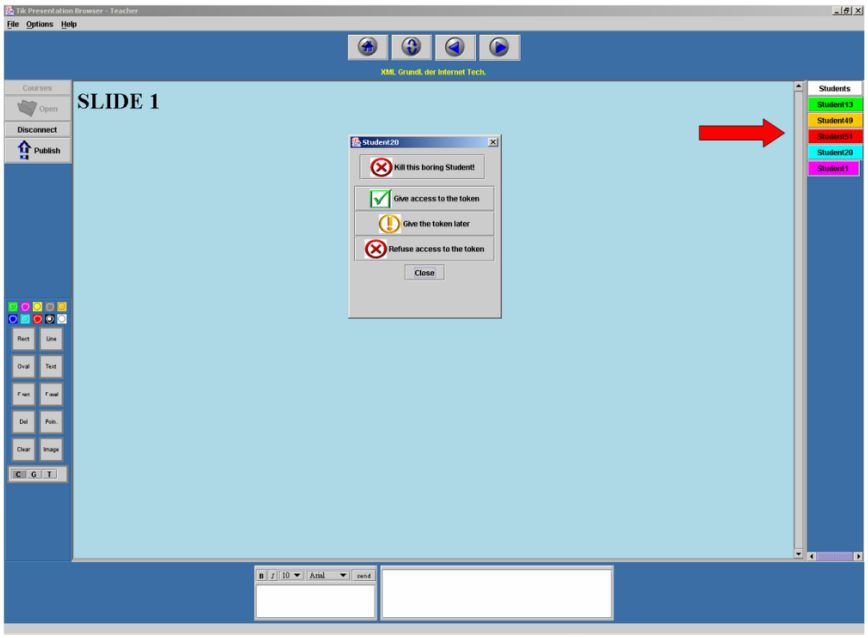

Figure A.7: Managing the connected students (teacher side)

- CYAN: The color of a new user
- PINK: The user is requesting the access to the whiteboard
- GREEN: The user has access to the whiteboard
- ORANGE: You delayed the access to the whiteboard for this user
- RED: The access to the whiteboard has been refused for this user

If you click on a student you can perform different tasks:

- Killing him if you don't want to see him in your application anymore
- Giving him access to the whiteboard
- Refusing him access to the whiteboard
- Let the user waiting for the whiteboard access

As soon as a student will disconnect (or will be killed), its name disappear from the users' list.

6. **Adding/Removing FTP Servers**

   The application provides some predefined ftp servers to log on for publishing or downloading presentations. However you can define your own default servers by going to the menu Options and clicking *FTP servers*. Here you can add new servers by clicking *new...* or removing the already defined ftp servers by picking them up from the list under "delete".

A.3.2 Student

1. **Selecting a course to follow**

   To select a course you have basically to click on the COURSES button on the top left of the application window. You also have the choice to select the course by going in the menu to *File - Select Course* (see Figure A.3).

   - Click the *Courses* button on the top-left of the window
   - The default access-point is contacted and the course list is retrieved
   - A new dialog window with a list of courses will appear
   - Select the course you want to follow and click *choose*
   - The selected course's IP Address will be set from the retrieved list
   - On the center of the upper panel (on the navigation bar) appears the name of the selected course

2. **Connecting to a presentation**

   To resolve an existing connection and following a course click the CONNECT button in the button list on the left size of the window or go to the File menu and click *Connect* (see Figure A.6).

   - Click the *Connect* button to resolve the connection
   - After a while a new dialog window appears, confirming you that you are successfully connected
   - Click the *Close* button; a new dialog window will appear, asking you to login into an ftp server for downloading the presentation
   - Insert your login and your password for the server indicated on the dialog window and click *Connect*
   - The presentation is loaded at the current running status and you can begin to follow the course
   - Look at the Connect button: now a button click will disconnect you from the session
   - If you disconnect, the presentation will be closed and the tutor will be informed of that

A.3.3 Access-point, Whiteboard and Chat

- **Updating the access-point**

  The access-point is the only fix point of the whole application: it has to be known to all participants, so keep in mind that changing it can cause potentially fails of the whole application. A teacher has a double access to the access-point: on one side he's accessing the Web Server for reading the courses IP addresses, on the other he's accessing the FTP server to update the current IP address. The FTP server provides a list of text files, named as the courses and containing the IP addresses of the teacher applications teaching the selected course.

  As teacher you can modify 5 parameters:

- HOSTNAME: The name of the server (both FTP and HTTP)
- WEB PATH: The path of the file providing access-point information for the users
- FTP PATH: The internal path of the FTP server where to find/put the modified IP files
- LOGIN: The login needed to access the FTP server
- PASSWORD: The password needed to access the FTP server

As student you can only modify the parameter you need (HOSTNAME and WEB Path).
To modify these parameters go to the *Options* menu and click *Access Point configuration*. By opening this menu a dialog window containing the current value of these 5 parameters will be displayed. You can change the parameters and save them by clicking on *Change*.

- **Whiteboard and Chat**

  The presentation tool provides a Whiteboard, placed over the presentation slides, and a Chat line. These 2 components let the teacher interact with the students. The teacher has always access to the whiteboard, the student per default not.

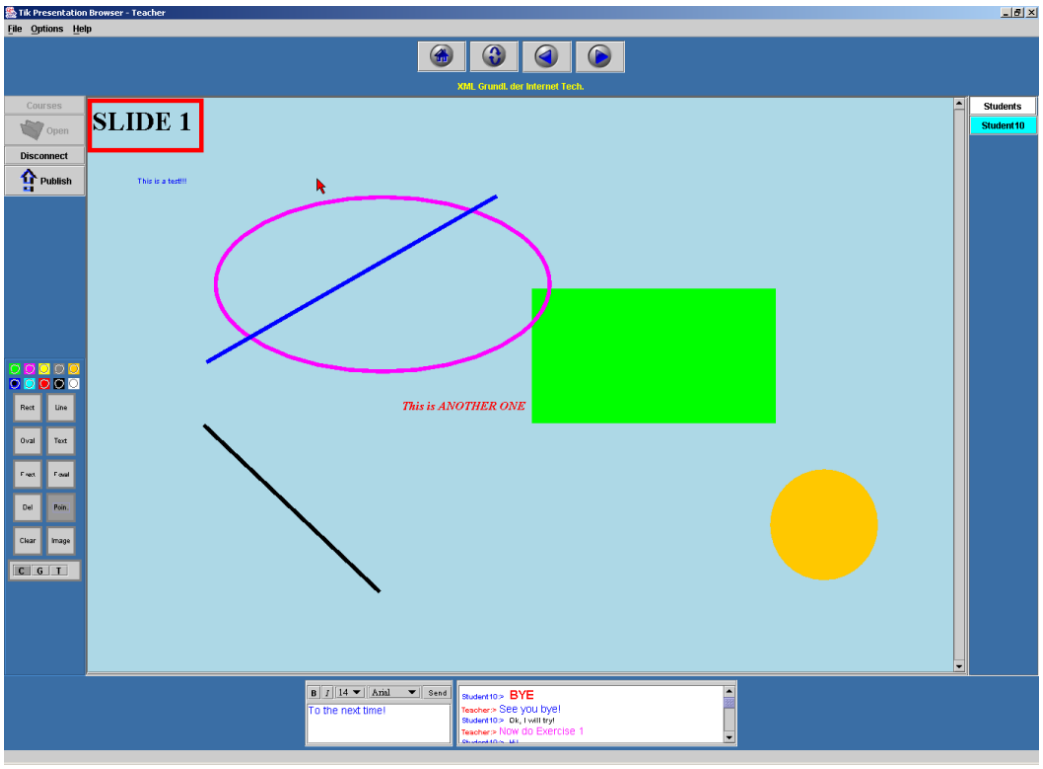

Figure A.8: Using the whiteboard

Using the users' panel, the teacher can manage the students and give them access to the whiteboard. For requesting such an access the students should click the token button in the whiteboard button's panel. The color of this button indicates whether the student has access to the whiteboard (green) or not (red). The other buttons

allow the users to draw different drawings with different colors on the drawing area placed over the presentation slides (see Figure A.8 and Figure A.9).

The chat line is placed at the bottom of the window and can be used by any user connected to the session to send messages to all other users (see Figure A.9). It is possible to change the size, the font or the style (bold/italic) of the written text. In the Chat panel all the sent messages will be displayed indicating the respective sender (teacher always in red, students always in blue).

For writing text in the whiteboard, a user has first to select the T button in the whiteboard button's panel, then can write and format the text in the chat panel and finally can click with the mouse on the whiteboard, at the place where he wants to insert the text.

A user can also point something on the slides: for doing that just select the pointer button in the button's panel and hold the mouse button pressed dragging the mouse over the drawing area.

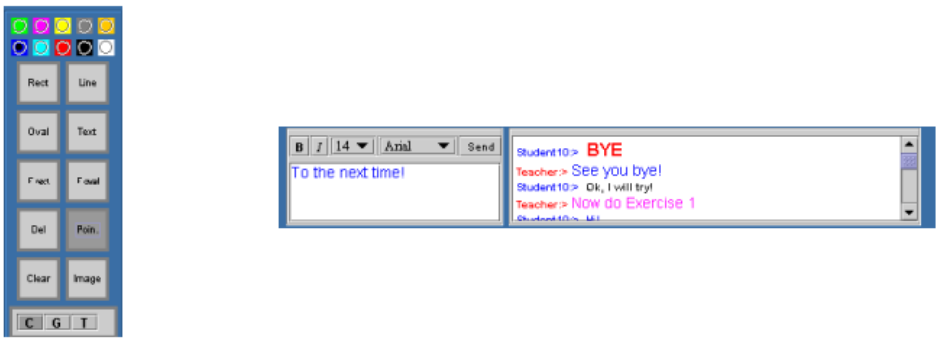

Figure A.9: Whiteboard Buttons and Chat Panel

Bibliography

[1] HotJava Browser Product Family (at Sun Microsystems). *http://java.sun.com/products/hotjava/*.

[2] HotJava HTML Component (at Sun Microsystems). *http://wwws.sun.com/software/htmlcomponent/*.

[3] Mozilla.org Web Client. *http://www.mozilla.org/projects/blackwood/webclient/*.

[4] BrowserG! Version 1.3. *http://browserg.mozdev.org/*.

[5] Java Browser Technology, Components for Enterprise, and Embedded Application Developers. *http://www.icesoft.no/*.

[6] The Jazilla Project. *http://jazilla.sourceforge.net/*.

[7] JRenderer Design Document. *http://jazilla.sourceforge.net/products/jrenderer/*.

[8] XBrowser Extended Web Browser. *http://xbrowser.sourceforge.net*.

[9] Netbeans Module Netbrowser. *http://netbrowser.netbeans.org*.

[10] Citec DocZilla. *http://www.doczilla.com*.

[11] Clue Browser for Desktop/Server Applications. *http://www.netcluesoft.com*.

[12] Calpa Java Package Homepage. *http://www.netcomuk.co.uk/∼offshore/index.html*.

[13] S. Shirmohammadi, A. El Saddik, N.D. Georganas, and R. Steinmetz. Web-based multimedia tools for sharing educational resources. *ACM J. of Educational Resources in Computing (JERIC)*, 1(1):13, March 2001.

[14] The Source for Java Technology. *http://java.sun.com*.

[15] The PPT2HTML PowerPoint to HTML Converter. *http://www.rdpslides.com/ppt2html/*.

[16] CZ-Ppt2Htm V1.0. *http://www.convertzone.com/ppt2htm/*.

[17] Microsoft PowerPoint Web Publishing Accessibility Wizard. *http://www.rehab.uiuc.edu/ppt/*.

[18] World Wide Web Consortium. *http://www.w3.org/*.

[19] Java Shared Data Toolkit Home Page. *http://java.sun.com/products/java-media/jsdt/index.html*.

[20] Rich Burridge. Java Shared Data Toolkit user guide. Technical report, Sun Microsystems, Inc., October 1999.

[21] Java Foundation Classes: Cross-Platform GUIs and Graphics. *http://java.sun.com/products/jfc/*.

[22] Abstract Window Toolkit (AWT). *http://java.sun.com/j2se/1.4.1/docs/guide/awt/*.

[23] Java 2D API. *http://java.sun.com/products/java-media/2D/*.

[24] Standard Edition (J2SE) Java 2 Platform. *http://java.sun.com/j2se/1.4.1/download.html*.

[25] version 1.4.1 Documentation Installation Instructions for Java 2 Platform, Standard Edition. *http://java.sun.com/j2se/1.4.1/install-docs.html*.

[26] Ronan Sorensen. Java sockets push. what's possible. *Visual Basic Programmer's Journal*, pages 111–113, December 1997.

[27] Jane Hunter and Suzanne Little. Building and indexing a distributed multimedia presentation archive using SMIL. Master's thesis, University of Queensland, Australia, 2002.

[28] Klaus H. Wolf, Konrad Froitzheim, and Schulthess Peter. Multimedia application sharing in a heterogeneous environment. *ACM Multimedia 95*, November 5-9 1995. San Francisco, California.

[29] The Application Sharing Technology. *http://www.motifzone.com/tmd/articles/XpleXer/XpleXer.html*.

[30] Collaborative browsing in Information Resources. *http://www.tik.ee.ethz.ch/∼cobrow/home.html*.

[31] Cay S. Horstmann and Gray Cornell. *Core Java 1.2*. Prentice Hall, 1999.

[32] Rhino: JavaScript for Java. *http://www.mozilla.org/rhino/*.