Chapter 3

The E-Chalk Application

This chapter describes the system architecture and implementation of the main E-Chalk server application. The user front-end of this main part is the setup dialog described in Section 2.4.1. It handles the settings for the E-Chalk session and controls the subcomponents board, audio, and video, as well as the subcomponent for creating the PDF transcript.

The setup of the main application is represented as `java.util.Properties`[1] and is stored in the Java `Properties` file format. Two `Properties` files are of central relevance to the system. The file for the configuration setup described in Section 3.1 controls the whole E-Chalk application. The other file contains the settings for a lecture recording, changed by the setup dialog and read by the above-mentioned subcomponents.

The entire application was written in Java [GM96] and requires the libraries of Java version 1.3 or later. The audio and video server components rely on the Oscar [70] implementation of the OSGi component framework. When using the video component, the Java Media Framework (JMF) [44] library is also needed. See Chapter 5 for details.

3.1 Main Configuration

The general setup `Properties` are read from the standard Java system `Properties` and the E-Chalk startup configuration file <*echalk install dir*>`/conf/-main.conf`, stored in the Java `Properties` file format. The entries define, among others, the locations of secondary files, like bookmarks and online documentation, may activate the debug output mode, and give class names of components to load dynamically.

The property entries in the startup configuration do not need to define the entries as literal strings. The entries may use other property entries[2] which will be evaluated recursively when the application accesses the property entry, where faulty definitions, which are due to circular references, will be detected. A property is referenced as `${`<*property key*>`}`, for example `${echalk.conf.dir}/audiowizard.conf`, the default entry for the audio wiz-

[1] Java `Properties` objects are hash tables which map `String` keys to `String` entries.

[2] Referenced properties may include the startup configuration properties as well as some special E-Chalk properties made available at runtime.

`main.conf`	E-Chalk's startup configuration
`echalk.conf`	lecture settings
`prefs.conf`	user's preferences on suppressing warnings
`streamer.conf`	configuration for audio/video streaming
`console.rc`	console startup command file
`console.dat`	console command history file
`echalk.log`	session log file (error messages, etc.)
`streamer.log`	log file for audio/video streamer system
`audio/audiowizard.conf`	configuration for audio profile wizard
`audio/converter.conf`	configuration of audio format converter
`audio/profiles.xml`	audio profile list
`audio/wwr3.xml`	audio components graph definition
`board/texthistory.dat`	board text history file
`bookmarks/applets.xml`	bookmarks of Applets
`bookmarks/cgis.xml`	bookmarks of CGI calls
`bookmarks/chalklets.xml`	bookmarks of chalklets
`bookmarks/images.xml`	bookmarks of images
`bookmarks/macros.xml`	bookmarks of board macros
`profiles/`	data directory containing audio profiles
`templates/templates.conf`	HTML template handler setup file
`templates/archived.def`	localized template for replay
`templates/live.def`	localized template for live transmission
`templates/recording.def`	localized template for ongoing recording
`video/wwv.xml`	video components graph definition

Listing 3.1: Files installed in the configuration directory.

ard's configuration file path, defining it locally to the entry for the configuration directory. The character `$` as a literal can be represented in entries by the empty variable `${}`. For properties with no definitions in the startup configuration file, default values are used.

The settings for a lecture recording which are changed by the setup dialog are stored in a second `Properties` file, referenced by the startup property `echalk.lecture.conf`.

After reading the properties from the main configuration file, the application's command-line parameters are evaluated in order of appearance. Three types of command-line parameters are recognized. Parameters in the form of `-X`<*key*>`=`<*value*> are used to override setup dialog property entries stored in the settings file. This may be used to start E-Chalk from a script auto-generating the lecture's output directory, e. g. using a combination of user name and date. Parameters with the syntax <*key*>`=`<*value*> without a leading `-X` are used to override entries from the startup configuration file.[3] Entries without a `=` character or in the form `-f`<*path*> are interpreted as property file, from which startup configuration properties are loaded (with the ability to override the entries from the standard startup configuration file). See Listing 3.1 for the list of E-Chalk configuration files.

[3]Because all of E-Chalk's configuration properties start with a lowercase letter and not with a dash, there is no problem in distinguishing the two parameter types.

3.1.1 Dynamically Loaded Classes

Several components of the system are realized as dynamically loaded classes. The boards connection to computer algebra system[4] and the handwriting recognition system[5] belong to these, as well as a number of components implementing the interface `de.echalk.util.Launchable`.[6] The `Launchable`s include the audio, video and board subsystems as well as the PDF converter and some debugging tools. The classes to be loaded are defined as properties in the configuration files.

The system delegates the class loading to the class `de.echalk.util.Resources`.[7] Its loading mechanism does not only search the classes from the Java VM's `classpath`, but from classes stored in the directory <*echalk install dir*>`/addons`, either residing directly in it or stored in a Jar archive file. This allows developers of E-Chalk components to quickly test their components by adding them to the `addons` directory without the need to compile the classes into the E-Chalk system core libraries or to change the classpath.

3.1.2 Multiuser Configuration

In the standard setup, all configuration files of E-Chalk reside in the <*echalk install dir*>`/conf` directory. The startup configuration file can redefine the property `echalk.conf.dir` to point to an alternative directory. All secondary configuration files are referenced by using paths relative to this property.

To manage several users working with a single E-Chalk installation but individual configurations, the `echalk.conf.dir` can be set[8] to a user-defined directory. For example it can be changed to `${user.home}/.echalk`, using the Java system property `user.home`, pointing to the user's home directory. This also avoids write-permission problems, when E-Chalk configuration files are installed write protected against non-owners on multi-user platforms.

To avoid requiring first-time users to manually copy these configuration files, this is handled automatically by E-Chalk: When the `echalk.conf.dir` directory does not exist or does not contain the configuration files, the files are copied from the original configuration directory and its subdirectories. The original configuration directory in turn is defined by the property entry `echalk.conf.src.dir`. A property named `echalk.conf.src.filter` defines a file-name filter for the files to be copied to prevent the copying of system files like the error log file.

3.2 Data Model of the Settings

The class `echalk.main.Settings` serves as a data model for the collected input elements of the setup dialog, which is stored in the lecture settings file, with its file path defined in the configuration entry `echalk.lecture.conf`. It is saved in the Java `Properties` file format.

[4]Described in Section 4.6.2.

[5]Described in Section 4.8.

[6]The `Launchable` interface id described in Section 3.10.1.

[7]Loading the Java resources like images for image buttons is also handled by this class.

[8]It can be redefined either by editing the `conf/main.conf` file or using command-line arguments, as described in Section 3.1.

All GUI elements of the startup dialog that accept user input are registered in the dialog's `Settings` object. The associated `Property` key as well as a mapping between localized GUI values and their internal `Property` values may be accessed via the `Settings` object. It also allows to change a registered input component's state by setting its corresponding `Property` value via its key. This mechanism is used to change settings by commands using the command-line console, see Section 3.10.1.

Structure Preserving Property Parsing

A Java `Properties` file can contain both key-value pairs and comments.[9] They are processed line by line, while lines end at the end of the file or with one of the line terminators `\n`, `\r`, or `\n\r`. Comments are lines that start with one of the characters `#` or `!`. Except for empty lines, all lines are interpreted as key-value pairs. Key and value are separated by a whitespace, a `:`, or a `=`. If these characters appear in the key, they must be escaped by a \. Whitespaces directly after the separator are also ignored; if the entry starts with a whitespace, it must be escaped. Entries may also be distributed over several lines, as escaped line terminators are ignored in entry parsing.

In the standard E-Chalk installation, the `echalk.lecture.conf` file has extensive comments and a semantic ordering to assist novice user changing the settings using a text editor, see Section 2.4. If the setup dialog would simply use the standard Java `Properties` `load` and `store` methods to make the key-value pairs of changed settings persistent, comments and ordering would get lost. For this reason, the E-Chalk application represents the documents structure in an instance of `echalk.util.PropertyDoc`. It reads in the complete string representation of a `Property` file and determines the character intervals of the keys and the corresponding values in this string. A `PropertyDoc` object provides methods to replace the entry for a key, to add new key-value pairs and to write the document back, all while preserving the complete structure.

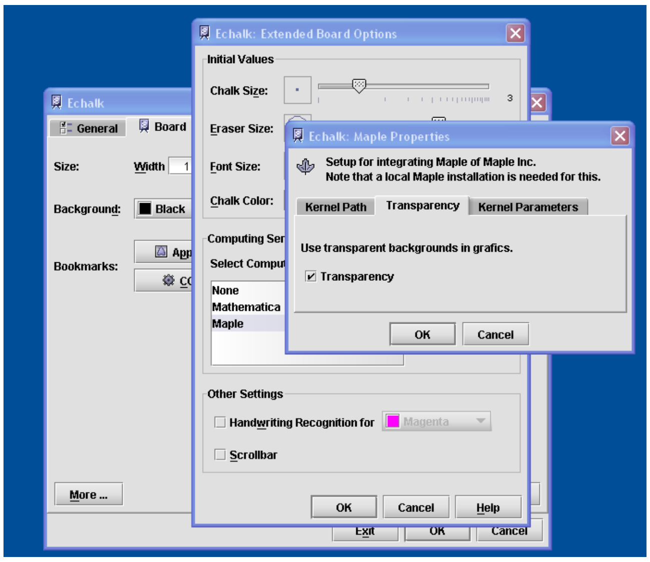
Expert Mode

For *expert-mode editing*[10], the `PropertyDoc`'s string representation is loaded into an `echalk.util.TextEdit` instance, which implements a graphical text editor with all the standard features. The setup dialog is hidden and the text editor is shown instead. When the text editor is closed, it executes a callback method to show the setup dialog again. The `PropertyDoc` for the setup dialogs settings is updated, if the `echalk.lecture.conf` file has changed.

3.3 Connecting Computer Algebra Systems

The list of computer algebra system available in the setup[11] is defined by the configuration entry `echalk.computingserver.add`. It is a comma separated list of `de.echalk.util.EvaluatorKit` classes. These classes provide static builder methods that returns a `de.echalk.util.Evaluator` instance to serve as

[9]The format is given in the API documentation of the `load` and `store` methods of `Properties`, see [40].
[10]See Section 2.4.
[11]See part on board settings in Section 2.4.



Setup for integrating Maple of Maple Inc.
Note that a local Maple installation is needed for this.

Use transparent backgrounds in grafics.

[✓] Transparency

[ ] Handwriting Recognition for: Magenta

[ ] Scrollbar

Figure 3.1: Setup dialog generated for a computing server setup info with three parameters, showing the tabbed panel of the boolean parameter *Transparency*.

actual connection to a computer algebra system like Mathematica. Other static methods return a localized name and an optional icon for the `EvaluatorKit` as well as a `de.echalk.util.KitSetupInfo`, which describes the parameters needed by the builder method, and an overall (localized) description of the kit. For each of these parameters, a `KitSetupInfo.ParamInfo` object is given by `KitSetupInfo`. `ParamInfo` itself is an abstract class, with the available concrete subclasses defining the type of the parameter. Allowed parameter types are strings, strings without line breaks, passwords, files, booleans, enumerations (list of possible choices), and whole numbers (for a declared interval, up to the long integer range). Each `ParamInfo` instance defines a parameter name and may give a description and a default value, with all texts being localized. For example the `echalk.util.cserver.MathematicaKit` needs the path of the Mathematica kernel file as a parameter.[12] `ParamInfo` objects for files may also provide a customized file chooser using appropriate file filters to help with file selection.

When the user selects to configure an `EvaluatorKit`, a dialog automatically generated from the `KitSetupInfo` is shown with input fields adequate for the type of parameter: check boxes are used for boolean parameters, combo boxes for enumeration parameters, text areas and text fields for strings (depending on whether these are allowed to contain line breaks), password input fields for password strings, text fields with file choosers for file parameters, and number spinner objects for whole numbers. See Figure 3.1 for an example.

[12]For convenience, the Mathematica kit provides an intelligent default value by scanning the file system's standard installation paths of Mathematica for a kernel executable, preferring the most recent version if several installations are found.

```
<!DOCTYPE E-Chalk:bookmarks [
  <!ELEMENT E-Chalk:bookmarks (entry*)>
  <!ATTLIST E-Chalk:bookmarks title CDATA #REQUIRED>
  <!ELEMENT entry (name, url, thumbnail?, arg*, property*)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT url (#PCDATA)>
  <!ELEMENT thumbnail (ziprgb | href)>
  <!ELEMENT ziprgb (#PCDATA)>
  <!ATTLIST ziprgb width CDATA "16">
  <!ATTLIST ziprgb height CDATA "16">
  <!ELEMENT href (#PCDATA)>
  <!ATTLIST href width CDATA "16">
  <!ATTLIST href height CDATA "16">
  <!ELEMENT arg (prompt, id)>
  <!ATTLIST arg post (yes|no) "yes">
  <!ELEMENT prompt (#PCDATA)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT property EMPTY>
  <!ATTLIST property key CDATA #REQUIRED>
  <!ATTLIST property value CDATA #REQUIRED>
]>
```

Listing 3.2: DTD of bookmark files and the audio profile list.

3.4 Bookmark Files

All bookmarks are stored in XML files with the inline DTD[13] shown in Listing 3.2. Each entry has an name to identify it and a URL to load the data from. As an optional[14] field, an icon entry named `thumbnail` is provided for visual identification in the dialogs. An icon is stored as a URL reference to an installed image resource or as raw image data. A reference is used for an icon image supplied with the E-Chalk system[15], accessible in the bookmarks editors with the *Set Icon>Choose ...* menu. Raw images, as produced from the *Thumbnail* and *Import ...* sub-menus of *Set Icon*, are stored as hexadecimal zipped RGB data. For the icon size of 16*16 used in the bookmarks, even uncompressed raw RGB pixel data is often smaller than standard image formats like GIF or PNG due to the size overhead from format headers.

For image and Applet bookmarks, these are the only informations that are stored. CGI bookmarks additionally provide a list of CGI parameters. A parameter is denoted by the key `arg` and has entries for the parameter's name, the HTTP method to send it for the CGI request (GET or POST) and a string to prompt the user for input. See Listing 3.3 for an example entry.

Macro bookmarks use the key-value pair entries to store the board width and board background color used in the macro to enable the board to decide

[13]The document type definition, or DTD, provides the grammar for a class of XML documents [BPSM+04].

[14]The bookmark dialogs of the board show a default icon for bookmark entries that do not define their own icons.

[15]The supplied images are installed as Jar bundles, so the URL uses the Java specific `jar` protocol with the syntax `jar:`<*URL of jar file*>`!`{<*entry in Jar archive*>}.

```
<entry>
  <name>Calendar</name>
  <url>http://www.online-lectures.org/cgi-bin/cal.cgi</url>
  <thumbnail>
     <href width="16" height="16">
        jar:file:lib/icons.jar!/icons/cgi16.gif
    </href>
  </thumbnail>
  <arg post="false">
     <prompt>[[month] year] </prompt>
     <id>date</id>
  </arg>
</entry>
```

Listing 3.3: Example CGI bookmark entry for a calendar resembling Unix's `cal`.

which macros are compatible with the current board setup without having to load the macro itself. A third key-value pair defines the speed used for replaying the macro.

For chalklet bookmarks, the URL entry is the classpath to the chalklet classes to be loaded. Additional key-value pairs define the minimum height and width of the chalklet (enabling the board to sort out chalklet bookmarks that would not fit on the board), the name of the `ChalkletKit` class to be used to build the chalklet[16], and the parameter values to be passed to the chalklet kit's build method as well as the localized parameter names, enabling the board to show a user-friendly tooltip, see Section 4.5. For providing the user-guided interface to set the parameter, a `ChalkletKit` has to define the description of expected parameters in the bookmark editor, like an `EvaluatorKit` does for the setup. See Section 3.3 for an description of the `KitSetupInfo` provided by `ChalkletKit`.

An excerpt from a chalklet bookmark file is shown in Listing 3.4.

3.5 Audio Profiles

The list of available audio profiles[17] is stored as an XML file, using the same DTD as the bookmarks shown in Listing 3.2. The URLs define the location of the profile data identified by the name entry, the thumbnail entry is unused. Additional information is stored as key-value pairs with the entries. These informations are shown in the *Properties* dialog for the profiles to help the user identifying the profiles. While most of this information is stored explicitly or implicitly in the profiles, storing them in the XML list makes it available to the setup dialog without having to load the relatively big audio profile data files. As a side effect, this makes the setup dialog independent from format changes in the profile-data files.

[16]See Section 4.6.4 for more on the abstract `ChalkletKit` class and `Chalklet` interface.
[17]See part on audio settings in Section 2.4.

```
<entry>
  <name>Logicrecognation</name>
   <url>file:etc/logicrecognition.jar</url>
   <thumbnail>...</thumbnail>
    <property
       key="classname"
       value="de.echalk.logicrecognition.LogicAnimationKit"
    />
    <property key="minheight" value="300"/>
    <property key="minwidth"  value="300"/>
    <property key="param.no" value="3"/>
    <property key="param.0.name" value="feedback"/>
    <property key="param.0"      value="true"/>
    <property key="param.1.name" value="ini file"/>
    <property key="param.1"      value="etc/logicrec.conf"/>
    <property key="param.2.name" value="diagram"/>
    <property key="param.2"      value="false"/>
</entry>
```

Listing 3.4: Example chalklet bookmark entry referencing the logic-circuits-simulating chalklet described in Section 6.12.4. Thumbnail data is omitted in this example.

Audio Profile Wizard Launch

For creating a new audio profile in the setup dialog, the audio profile wizard application is started.[18] See Section 5.2.3 for a description of how the profile wizard works.

The wizard directly changes the audio profile list when started as a standalone application. Started as a component of the E-Chalk setup dialog, profiles should only become permanent when the setup dialog is exited with *OK* or *Exit* and not with *Cancel*. To this end, the setup dialog creates a temporary file with a copy of the XML audio profile list. This copy is given to the wizard for modifying. When the setup dialog exits with *Cancel*, the original XML file remains unchanged and any newly created audio profile, identified by being referenced in the temporary copy but not in the original file, is removed. If the setup dialog is closed with *Exit* or if a recording is started with *OK*, the temporary XML file is copied to the path of the audio profile list XML file and any user delete action on the profile list by users is made permanent by removing any now unreferenced profile data file.

3.6 Starting a Lecture Recording

When a recording is started, the output directory is generated, and the necessary files for remote access are written, like the HTML index file and the Jar archive containing client classes. See section 3.7 for possibilities to customize these files.

[18]Starting the profile wizard from the setup dialog relies on the wizard implementing the `Launchable` interface, described in Section 3.10.1.

```
echalk.board.class=\
  echalk.board.Launcher
echalk.board.class.args=\
  board.overwrite=${board.overwrite},${echalk.lecture.conf}
echalk.audio.class=\
  de.echalk.audio.EchalkAudio
echalk.audio.class.args=\
  ${echalk.conf.dir}/streamer.conf
echalk.video.class=\
  WWV.Launcher
echalk.video.class.args=\
  ${echalk.lecture.conf}
```

Listing 3.5: Recording components properties in file `main.conf`.

If the lecture is recorded in append mode and with audio recording data in the old audio format, it is transformed into the new format.[19]

Components for the streams to record, board, audio, and/or video are loaded and started dynamically. The names and parameters of the components are defined in the startup configuration files as properties, see Listing 3.5. The loading process relies on the components implementing the interface `Launchable` described in Section 3.10.1. Using the interface methods, the initialization phase and the execution phase are put into different methods because the time needed for initializing the components may differ considerably. For example, the time for setting up the board component for append mode depends on the size of the previous recording, as the board must load the old content.

After hiding the setup dialog, each component is started in a different thread and execution blocks until the board thread returns, as the board component is the only one that allows the user to end the session. If the board is not among the recording components started, an indeterminate progress dialog with a stop button is shown instead, see the part on recording in Section 2.4.

If starting the recording fails, for example when there is no write permission on the output directory, a localized error message is presented to the user and the setup dialog reopens, enabling the user to change any erroneous settings and retry to record.

When the recording was successful, the output directory is set up for archived access. First, the PDF version(s) of recorded board data is/are generated, again relying on the board-to-PDF converter being realized as a `Launchable`. See Section 6.1 for details. The length of the recorded lecture streams is determined and with this information the HTML index file for the archived replay is generated, as described in Section 3.7.

[19]Like the recording components, the converter is started by using the `Launchable` interface. To be more specific, the conversion is implemented using the `Progressible` sub-interface of `Launchable`, allowing to show a feedback on the conversion progress. See Section 3.10.1 for a description of the interfaces. For a description of the old WWR2 and the new WWR3 audio format, see Section 5.1.

3.7 HTML Templates

The E-Chalk server creates three different kinds of HTML index files in recording directories. First, there is an index file for an ongoing recording without live transmission available. The HTML file serves to inform remote users about the lecture not being available yet. Then there is an HTML file for a live transmission that embeds the Applets for receiving the transmitted stream (board, audio, and/or video) and the relevant transmission parameters, like which TCP port to listen to. Finally, there is a page for archived replay, embedding the replay Applets of the recorded streams, which are the same Applets as those for the live transmission, but with different parameters, plus an Applet similar to a VCR control.[20] Also, PDF transcripts of the lecture are linked from this page.

For these pages to be customizable in advance by the user depending on the lecture parameters[21], they are generated from templates interpreted by the `echalk.main.util.TemplateProcessor` class. The configuration file `conf/-template/template.conf` determines which of the template files is used for which of the three HTML files, the character encoding used by the template, and the names of the HTML files to produce. The character encoding property allows the use of localized templates with managed character sets, for example when using a Korean locale. Also, source and target name of the Applet's client Jar file and optional skin data for the VCR control Applet are specified. Optionally, a list of files to be copied to the lecture recording directory can be specified, too, for example for images to be embedded by the HTML files.

The templates are a mixture of literate HTML and interpreted code. The code parts are enclosed in {} brackets. Within the code area, parts can be aggregated to blocks by enclosing them in {} brackets. Literate strings in the interpreted section are marked by enclosing pairs of quotes or double quotes. Tokens, which are not literals, predefined keywords, brackets, or operators are interpreted as variables. For a variable, the value of the `Property` with the variable name defined in the E-Chalk system is used, or the empty string if it is undefined. For property lookup, the lecture settings and the startup configuration properties are used. In addition, a few runtime properties are defined, like `time`, `date`, and (in templates for replay from archive) the values of the recorded lecture's length.

Supported statements in the template code are `if-else` statements with a syntax similar to Java and a statement for setting a property variable in the context of the template, with syntax being `set` <*varname*> <*value*>. All other statements are terms formed by literates, variables, or operators on sub-terms and are interpreted as output of the corresponding values.[22]

Supported operators are the comparison operators `!=` (not equal), `==` (equal), `<`, `<=`, `>`, `>=`, and the boolean operators `&&` (and), `||` (or) and `!` (not). Comparison operators work with lexical string order. Possible return values of boolean operators are the strings `"true"` and `"false"`. As boolean operand, only the

[20]The VCR control Applet is described in Section 7.1.

[21]For example, the default pages use the lecture title as page title.

[22]The syntax could easily be extended by additional statements like loops or arithmetic operators to make the template syntax computationally complete. Also, allowing an operator for indirect addressing might be introduced then. For the templates produced here, the power of the provided statement is enough and the simple syntax makes the template definition less error prone. When targeting at a full-fledged programming language, it would make more sense to use a standard script language like Perl or Python.

```
<HTML><HEAD>
  <TITLE>E-Chalk: {echalk.title}</TITLE>
  <META name="author" content="{user.name}">
  <META name="date" content="{date}">
</HEAD>
<BODY bgcolor="#000000" text="#FFFFFF">
<DIV align="center"><H1>{echalk.title}</H1><P>
{ if (echalk.pdf) {
    if (pdf.create=="both") {
      'The session as PDF files: '
      '<A href="pdf/lect_col.pdf" target="_TOP">color</A> and '
      '<A href="pdf/lect_bw.pdf" target="_TOP">b/w</A>.\n</P>'
      '<P>\n'
    } else if (pdf.create=="color") {
      'The session as <A href="pdf/lect_col.pdf" target="_TOP">'
      'Portable Document Format</A> file.\n</P><P>\n'
    } else {
      'The session as <A href="pdf/lect_bw.pdf" target="_TOP">'
      'Portable Document Format</A> file.\n</P><P>\n'
    }
  }
  if (echalk.audio) {
    '<APPLET archive="'template.archives'" code='
    '"wwr.wwrclient.class" width="1" height="1" name="audio">\n'
    if (echalk.audio.sync.alpha!=null) {
      '  <PARAM name="syncalpha"'
      ' value="'echalk.audio.sync.alpha'">\n'
    }
    '  <PARAM name="archivemode" value="audio/">\n'
    '  <PARAM name="loopmode" value="on">\n'
    '</APPLET>\n'
  }
  if (echalk.video)
    [...]
  if (echalk.board)
    [...]
}
<APPLET archive="{template.archives}" code="console.Console"'
  ' width="1" height="1" name="console">
  <PARAM name="title" value="{echalk.title}">
  <PARAM name="seconds" value="{echalk.recordingtime.seconds}">
{ if (template.skin.src!=null) {
   '  <PARAM name="initfile" value="skin.ini">\n'
  }
' Please activate Java!'
}</APPLET></P></DIV></BODY></HTML>
```

Listing 3.6: Default English template for archived replay. Video and board Applet parts are omitted due to space constraints.

string `"true"` is interpreted as *true*. Bracketed sequences of values are interpreted as the concatenation of their string values. See Listing 3.6 for an example template.

Besides user-defined text and layout changes, the templates can be used to add other elements like a chat line in the live page, to communicate with the lecturer or a teaching assistant. It can also be used to adjust the E-Chalk client's behavior, for example to set a client parameter that results in the client board being displayed embedded in the browser window instead of opening a new frame.[23] For another usage example see the setup with three live board clients described for the FU data wall in Section 8.1.1.

3.8 Help System

E-Chalk's user manual is installed in HTML format. When the user accesses the help, the system displays the manual in a simple browser which is part of E-Chalk. Early versions of E-Chalk used an external browser application instead of its own, which uses the Java Swing HTML rendering engine. In E-Chalk's early history, the browser model could be guessed with high accuracy from the operating system, simply assuming the Netscape browser for Unix platforms and the Internet Explorer on Windows. Nowadays, it is no longer that easy to determine a browser to use automatically, as the HTML viewers have become to manifold on Linux systems for such a simplistic approach and E-Chalk provides its own one instead. For historical reasons there is still the `echalk.htmlviewer` entry in the `conf/echalk.conf`, which can be changed to use another browser for displaying the manual when it is started from the boards tool dialog.

When the help is started from within E-Chalk, the browser shows the help section relevant to the subcomponent it was started from, providing basic context sensitivity for the help system. For example, when it is started from the audio profile wizard[24], it opens the help page at the section describing the current wizard page, accessing the different help sections by the use of HTML anchors. See Figure 3.2 for an example.

The built-in browser implements the `Launchable` interface and can therefore also be started from the E-Chalk command-line console, see Section 3.10.1.

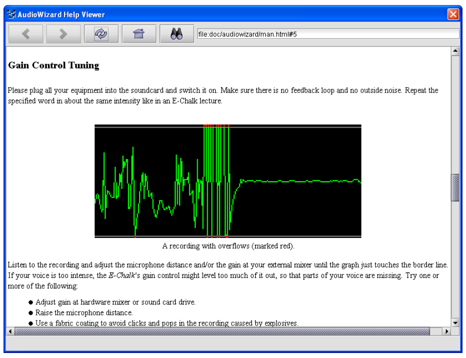
3.9 Localization

The user interface of the E-Chalk server application (and its installer) is internationalized, including subcomponents like the E-Chalk board. Only those software tools and documentations provided exclusively for developer use have an English-only interface. These include the command-line console described in Section 3.10.1, the internal message log described in 3.10.2, the included API documentation, and the comments in the configuration files.

Localizations realized are English (US variant) and German (DE variant). An old version of the server is also available in Spanish, and parts of the interface

[23]This feature was actually requested by the department of Economics at Universität Regensburg. Multimedia-supported lectures were produced at the department in a project of the *Virtuelle Hochschule Bayern* framework. To this end, RealAudio and RealVideo streams were combined with E-Chalk board recordings.

[24]See Section 5.2.3.



Gain Control Tuning

Please plug all your equipment into the soundcard and switch it on. Make sure there is no feedback loop and no outside noise. Repeat the specified word in about the same intensity like in an E-Chalk lecture.

A recording with overflows (marked red).

Listen to the recording and adjust the microphone distance and/or the gain at your external mixer until the graph just touches the border line. If your voice is too intense, the *E-Chalk*'s gain control might level too much of it out, so that parts of your voice are missing. Try one or more of the following:

- Adjust gain at hardware mixer or sound card drive.
- Raise the microphone distance.
- Use a fabric coating to avoid clicks and pops in the recording caused by explosives.

Figure 3.2: The help browser showing the help page for the audio wizard's step on "*gain control tuning*".

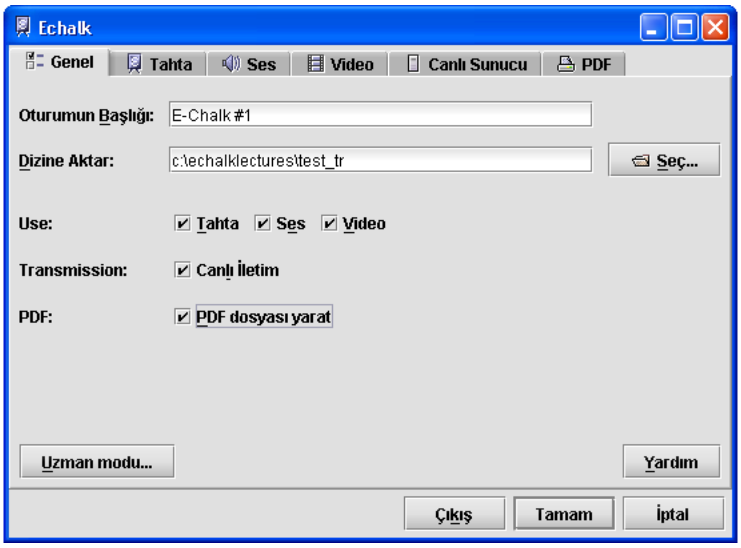
have also been translated to Turkish, see Figure 3.3.

To provide the mnemonics for GUI elements and text in the current locale, calls to static methods of the class `de.echalk.util.Txt` are used. The class `Txt` loads a `java.util.ResourceBundle` for the locale set.[25] E-Chalk uses Java's property resource bundles, meaning a property file is loaded to map lookup keys to localized entries. In addition to using this mapping, the loading mechanism of `Txt` has a special handling for the `echalk.l10n.load` key. It allows to specify a comma-separated list of additional property resource bundles to be loaded. These can in turn have such an entry of additional bundles to be loaded and so on, allowing to organize the localization data in a tree hierarchy. The root property bundle for E-Chalk's localization is `dat.echalk` (meaning the property files are named `dat/echalk_de.properties` for the German locale, `dat/echalk_es.properties` for the Spanish locale, etc.). This resource bundle only contains a list of bundles to be loaded, with different bundles for different contexts. For example, one resource bundle contains all error messages and another bundle contains all labels occurring in the E-Chalk GUI. When the same key appears in more than one resource bundle already loaded, a warning is printed to the error stream, see Section 3.10.2.

One special resource bundle is initially empty in the standard E-Chalk installation: the bundle `etc.usertexts` is installed as files in the directory `etc`.[26] While the other resource bundles are packed into Jar archives, the property files for `etc.usertexts` can be directly modified with a text editor, allowing users who want to plug in their own components to the E-Chalk system to easily add their own localizations to the E-Chalk localization resources and use the `Txt` class for their components.

[25] Java's loading strategy for loading the resource bundles uses the most specific available match for the defined locale in the applications classpath. For example, for a locale named `de_DE_EURO`, the language German for the country Germany in the variant with Euro currency symbol, the bundles for `de_DE_EURO` are searched for first. If they cannot be found, bundles for `de_DE` will be loaded. Failing this, `de` bundles are loaded, and as a final fallback the general resources with unspecified language are used. For details, see [41] or [DC01].

[26] The directories `etc` and `addons` are provided to store user-defined additions to the E-Chalk system. The `addons` directory is used for all resources to be available via the standard E-Chalk class-loading mechanism described in Section 3.1.1. All other additions, for example local files generated by chalklets, should reside in `etc`.


Echalk

Genel | Tahta | Ses | Video | Canlı Sunucu | PDF

Oturumun Başlığı: E-Chalk #1

Dizine Aktar: c:\echalklectures\test_tr [Seç...]

Use: ☑ Tahta ☑ Ses ☑ Video

Transmission: ☑ Canlı İletim

PDF: ☑ PDF dosyası yarat

[Uzman modu...] [Yardım]

[Çıkış] [Tamam] [İptal]

Figure 3.3: The setup dialog localized to Turkish.

To map a key string to its localization, the method `String Txt.get(String key)` is used. The method `String Txt.get(String key, Object[] a)` is provided for formatted messages. In the localization entry, references to the array elements can be used to be replaced by the elements transformed to strings, for example in the entry `"Reading␣file␣{0}␣failed."`, the substring `"{0}"` is replaced by the array element at index zero. The referencing syntax also allows further formatting specifications, like number and date format specifications. It is the same as for formatted message output with `java.text.MessageFormat`, see [39] for details.

A problem introduced by dynamic localization is that lookup of keys cannot be statically checked by the Java compiler. Localization fails at runtime if the key is missing in the resource bundles, for example due to a typing error. Flaws in localization have to be identified by test runs, but reaching a complete code coverage is difficult, especially for localized error messages, which are accessed only in exceptional cases.

To address this problem, some tools have been developed for static checks, described below. In addition, a fallback mechanism is used: The localization used by E-Chalk are the English strings to be displayed. When a lookup for a key fails, the failure is written to the error log[27] and the key itself is used. As a result, the user gets the English version displayed instead of some cryptic internal messages or blank field. The drawback of using the English versions as keys is that there cannot be different localized texts where the English locale uses identical texts. This may be undesirable for translations which differ according to context.

The `GrepTxtCall` command-line application from the `echalk.tools.l10n` package[28] scans for all keys used in calls to the `Txt.get` methods and lists

[27]See Section 3.10.2 for the error-message logging mechanism.

[28]Localization is often abbreviated with L10N and internationalization with I18N, hence

```
> java echalk.tools.l10n.GrepTxtCall
usage> java echalk.tools.l10n.GrepTxtCall [-h|-k|-l|-s|-p|-0]
 [-nw] [-v] [-r] <file> ...
  -h  Print this message and exit.
  -k  Plain listing of all L10N keys in order of appearance
      (default).
  -l  List with filename and line number for each key.
  -s  List sorted keys with duplicated removed.
  -p  Sorted property output for default (identity) mapping.
  -0  No output of keys. Useful for getting just the warnings.
  -m  Search for mnemonics keys instead of text keys.
  -nw Nowarn - do not report non-literal keys.
  -v  Verbose output. Only used for -s and -p modes.
  -r  Recursively search all dir files for *.java files.
> java echalk.tools.l10n.KeyCmp
usage> java echalk.tools.l10n.KeyCmp <propfile1> <propfile2a>
 [<propfile2b> ...]>
```

Listing 3.7: Usage of command-line utilities for the internationalization of the system.

them. Because the `GrepTxtCall` performs only a static analysis of the Java source code, it cannot evaluate variables or method calls as keys and prints warnings if it encounters such keys. To be able to use this tool, the source code of E-Chalk uses only literal strings and strings composed of concatenated literal strings.[29] The command-line options of `GrepTxtCall` allow to directly output a property file usable as a resource bundle for the English localization, using the identity mapping between keys and entries. It handles any special encoding needed for the property file, for example spaces in the key must be escaped, as unescaped spaces separate property keys from entries. See Listing 3.7 for a complete listing of the program's command-line options.

The command-line application `echalk.tools.l10n.KeyCmp` prints a Unix `diff` style difference between the keys in a single property file and the set of keys from a list of property files, see Listing 3.7. This allows to conveniently check the keys from all the property files for a single locale against a list of all keys from the source code, generated by the `GrepTxtCall` tool. The combination thus provides a static check of the resource bundle entries against the localization calls in the program.

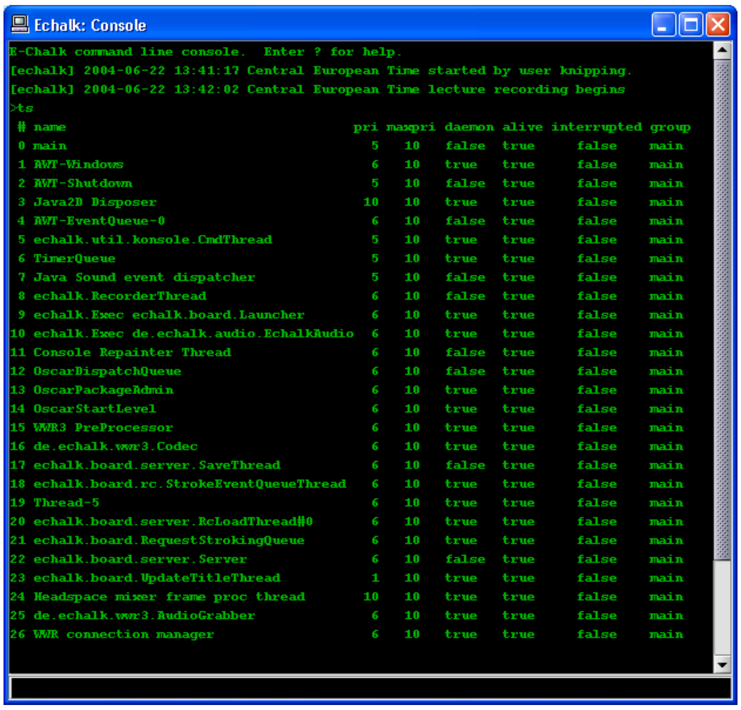
3.10 Debugging Tools

3.10.1 E-Chalk Command-Line Console

For debugging purposes, a command-line interpreter was added to E-Chalk, which can be run at the same time as the main system. It shows all outputs

the package name.

[29]`GrepTxtCall` *is* able to parse keys like `"Reading␣file␣{0}␣"+"failed."` and assemble them to a single string. It can also handle bracketed expressions.



```
E-Chalk command line console.  Enter ? for help.
[echalk] 2004-06-22 13:41:17 Central European Time started by user knipping.
[echalk] 2004-06-22 13:42:02 Central European Time lecture recording begins
>ts
 # name                                 pri maxpri daemon alive interrupted group
 0 main                                   5   10   false  true      false    main
 1 AWT-Windows                            6   10   true   true      false    main
 2 AWT-Shutdown                           5   10   false  true      false    main
 3 Java2D Disposer                       10   10   true   true      false    main
 4 AWT-EventQueue-0                       6   10   false  true      false    main
 5 echalk.util.konsole.CmdThread          5   10   true   true      false    main
 6 TimerQueue                             5   10   true   true      false    main
 7 Java Sound event dispatcher            5   10   false  true      false    main
 8 echalk.RecorderThread                  6   10   false  true      false    main
 9 echalk.Exec echalk.board.Launcher      6   10   true   true      false    main
10 echalk.Exec de.echalk.audio.EchalkAudio 6  10   true   true      false    main
11 Console Repainter Thread               6   10   false  true      false    main
12 OscarDispatchQueue                     6   10   false  true      false    main
13 OscarPackageAdmin                      6   10   true   true      false    main
14 OscarStartLevel                        6   10   true   true      false    main
15 WWR3 PreProcessor                      6   10   true   true      false    main
16 de.echalk.wwr3.Codec                   6   10   true   true      false    main
17 echalk.board.server.SaveThread         6   10   false  true      false    main
18 echalk.board.rc.StrokeEventQueueThread 6   10   true   true      false    main
19 Thread-5                               6   10   true   true      false    main
20 echalk.board.server.RcLoadThread#0     6   10   true   true      false    main
21 echalk.board.RequestStrokingQueue      6   10   true   true      false    main
22 echalk.board.server.Server             6   10   false  true      false    main
23 echalk.board.UpdateTitleThread         1   10   true   true      false    main
24 Headspace mixer frame proc thread     10   10   true   true      false    main
25 de.echalk.wwr3.AudioGrabber            6   10   true   true      false    main
26 WWR connection manager                 6   10   true   true      false    main
```

Figure 3.4: The command-line console listing the threads during recording.

from the standard out and error streams and provides a number of commands to control the application.

Commands provided include instructions to list and modify the running Java threads, see for example Figure 3.4, as well as instructions to access the property settings of the E-Chalk application. For example, the `sset` command can be used to set a property represented by a GUI input element in the setup dialog, changing the input component's value. This allows to control all setup dialog's settings from the command line. For convenient usage, the command line also features a persistent command-line history, controllable with the up and down arrow keys. Calls which are a duplicate of the previous call are not added to the history again.[30]

The commands are implemented by inheriting from an abstract `Command` class, which declares an abstract method reporting the command name, argument syntax, a short description, and a method for actually executing the command with the included arguments. The description and the argument syntax are used by the help command of the console. See Listing 3.8 for currently implemented commands.

On startup of the command-line console executes a startup file, named `conf/console.rc` in the default configuration. This is useful to define command aliases, to output information messages to the user, and to immediately start other debugging tools using the `Launchable` interface described below.

[30]This is equivalent to duplicate settings being ignored in Unix shells, e. g. `histdup=prev` in the `tcsh` or `HISTCONTROL=ignoredups` in the `bash`.

```
[echalk] 2004-06-21 15:41:44 Middle Europe Time started by user
knipping.
>help
# [<string> ...]                comment, no effect
alias [<alias> [<cmd>]]         list/set command aliases
cat <file> [...]                print out files
clear                           clear console
close                           disposes console
cset [<key> [<value>]]          list/set config properties
cunset <key>                    remove config property
deorbit <no>                    request termination of job <no>
destroy <no>                    destroy thread <no>
echo [<string> ...]             output strings
exec <command> [<arg> ...]      execute native system command
finalize                        request of pending finalizations
free                            show memory statistics
gc                              request garbage collection
help                            show this help
history [<n>]                   print console history entries
hsave <file>                    dumps history to file
hsetsize <n>                    sets command history length
hsource <file> [...]            load commands into history
iconify                         iconify console
info                            show echalk copyright info
interrupt <no>                  interrupt thread <no>
jobs                            list jobs (launchables started in
                                bg with '&')
launch <class> [<arg> ...]      starts a launchable
ls [-alFQRkhfStXr] [<file> ...] list directory contents
mkdir [-v] <dir> [...]          make directories
ps                              list processes (launchables
                                started)
quit                            request programm shutdown
resume <no>                     resume thread <no>
rmdir [-vp] <dir> [...]         remove directories
save <file>                     dump console content to <file>
seppuko                         exit w/o saving (dangerous)
set [<key> [<value>]]           list/set system properties
setpriority <no> <pri>          set priority <pri> to thread <no>
source <file> [...]             execute commands from file(s)
sset [<key> [<value>]]          list/set echalk settings
suspend <no>                    suspend thread <no>
ts                              list threads
tstop <no>                      stop thread <no>
unalias <alias>                 remove an alias
unset <key>                     remove system property
version                         show echalk version info
```

Listing 3.8: Log from the command-line console: List of available commands.

```
# uncomment to start memory monitor tool on console startup:
#launch echalk.tools.debug.MemoryMonitor &

# alias definitions:
alias ?       help
alias !       exec
alias print   echo
alias alloc   free
alias kill    deorbit
alias whoami  set user.name
alias uname   set os.name
alias arch    set os.architecture
alias pwd     set user.dir
alias la      ls -aF
alias ll      ls -alhF
alias edit    launch echalk.util.TextEdit
alias topdf   launch echalk.tools.pdf.Board2PDF
alias browse  launch echalk.util.HtmlViewer
alias showmem launch echalk.tools.debug.MemoryMonitor

# print welcome message:
echo "E-Chalk command line console.  Enter ? for help."
```

Listing 3.9: Example `console.rc` file.

See Listing 3.9 for an example startup file.

The Interfaces Launchable and Progressible

Several parts of the E-Chalk system are loaded and started dynamically, see Section 3.1.1. Their names and the parameters are not hard-coded, but defined in the startup configuration files as properties. The loading process relies on the components implementing the `de.echalk.util.Launchable` interface and such `Launchable`s can also be loaded and started from the command-line console using the `launch` command.

Examples for `Launchable`s in E-Chalk are the components for the streams to record, board, audio, and/or video. Further examples are the audio profile wizard, the board-to-PDF converter, the html browser used for the help system, and the text editor used to edit the configuration files in the expert-mode setup.

The `Launchable` interface specifies three methods, a `void init(String[])` method, a `void launch()` method, and a `void deorbit()` method. The `init` method initializes the components with component-specific arguments. The `launch` method actually starts the recording process without further delay. The two phases are separated to establish a synchronization point when used to start E-Chalk's different stream-recording components, see Section 3.6. The `launch` method is expected to block the thread it was called from until the `Launchable`'s process is terminated[31] or until it is stopped externally by calling the `deorbit`

[31]For example, the PDF converter does not respond until the conversion process from board data to a PDF file is finished.

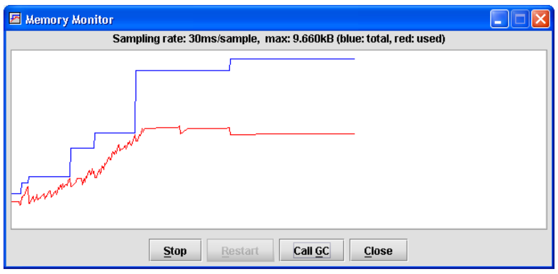

Memory Monitor
Sampling rate: 30ms/sample, max: 9.660kB (blue: total, red: used)
Stop
Restart
Call GC
Close

Figure 3.5: A memory-usage monitor for the JVM realized as `Launchable`.

method.[32]

A `Launchable` must provide a default constructor. Note that there is no language mechanism in Java to enforce the presence of the default constructor[33] or any other constructor signature in classes implementing an interface. This can only be specified informally in the documentation. It would have been possible to merge the init call with the constructor, but demanding a default constructor for components by documentation is common practice (like for example for Java Beans or for OSGI bundles) in contrast to demanding other constructor signatures.

The `launch` command of the command-line console dynamically loads the `Launchable` class given as the first argument, instantiates it, calls `init` on the instance with any further command-line parameters as arguments, and calls the `launch` method. The command interpreter's thread is blocked until the `launch` method returns. To avoid `Launchable`s blocking the interpreter, they have to be started in the background. For this purpose, any command can be started in a separate thread by using a trailing `&` sign, similar to Unix shell-style commands started in the background. The `Launchable` can then be stopped by using the `deorbit` command on it, executing the method of the same name on the `Launchable` object.

An example of a `Launchable` supplied purely for development purposes is the memory-usage monitor shown in Figure 3.5. Another `Launchable` used only for debugging from the command-line console is `OscarLaunch`. It allows to access the Oscar OSGi component management system.[34] Launching it adds all commands of the Oscar OSGi shell to the command interpreter[35], allowing to load, update and start Oscar bundles from the command line.

A sub-interface of `Launchable`, `de.echalk.util.Progressible`, is provided for implementing processes which can report their progress when launched. For example, the module converting E-Chalk audio data from the old WWR2 format to WWR3 is a `Progressible`. The sub-interface extents the `Launchable` by adding a `getProgress()` method which returns the relative progress as a double ranging from zero to one. When the setup dialog converts the audio

[32]For example, recording by the audio component needs to be stopped by `deorbit`.

[33]The default constructor is a constructor with an empty parameter list.

[34]The Oscar OSGi system is used by the audio and video server implementations to manage encoding and streaming components, see Sections 5.1.1 and 5.3.

[35]The Oscar shell command are assigned a common prefix in the E-Chalk command interpreter. This prefix, given as an argument to the `Launchable`, serves to avoid name collisions with the interpreter's built-in commands.

format to WWR3, it can show a progress bar for the conversion to give the user feedback.

3.10.2 Message Logging

The E-Chalk system defines its own `java.io.PrintStream` class for message streams. Its constructor allows to specify a label to prepend to each output line. The system uses this to mark output by the component it came from: the board component's output starts with `[board]`, the PDF converters output with `[pdf]`, output from the main control with `[echalk]`, etc. With the stream inheriting from `PrintStream` and handling the tagging of a line start internally, it can also be used for printing tagged exception stack traces by calling the `printStackTrace(PrintStream ps)` method of a `java.lang.Throwable`.

With the default settings for the E-Chalk configuration the stream output is sent to the system's standard output stream, the command-line console (if it was activated), and to a log file, by default named `conf/echalk.log`.

The E-Chalk components use two output streams, one being the tagged standard output stream and another for debugging purposes. When the system is started with the `echalk.debug` property set to true, the debug stream is set to the standard output stream. Otherwise it is set to a `/dev/null`-like stream, ignoring all output calls.