Chapter 6

Tools, Converters, Add-ons

This chapter describes a number of sub-components and stand-alone tools of the E-Chalk systems. These include converters for the E-Chalk formats, E-Chalk components implementing E-Chalk API classes like the `Launchable` interface[1] and board `Chalklets`[2], as well as stand-alone tools for working on recorded lectures. Unless explicitly stated otherwise, stand-alone tools are realized as Java applications.

6.1 Export to PDF

As mentioned in Section 2.5, the E-Chalk system can automatically produce a transcript of a board recording as an Adobe PDF file. The board-to-PDF converter `echalk.tools.pdf.Board2PDF` is called as a `Launchable` from the main E-Chalk application, but it can be also used as a stand-alone application.

In a first phase the board data from a given lecture are parsed by the converter to get the board image as it looks like at the end of the lecture. Invisible actions, like scroll events and drawings removed with undo, are ignored, all other board elements are kept in memory.

For line strokes, the first phase also drops inner points which are collinear with their neighbors, as they are not needed in statically painting pages. For example, connecting the sequence of points $(0,0),(1,0),(7,0),(8,0),(8,1)$ will result in the same image as the sequence $(0,0),(8,0),(8,1)$. E-Chalk recordings store each point delivered by the mouse driver during drawing actions. Because mouse drivers often deliver points faster than users change drawing directions, omitting the redundant points results in a considerable data reduction. In practice, the stroke points are reduced by a factor of two or three.

For obvious reasons, it is not possible to directly include an interactive Applet into the PDF. Instead, they are represented as an image with an Applet symbol.[3]

In the second phase, page breaks are determined, and in the final phase the actual PDF representation is constructed and written. An early implementation

[1] `Launchable` and its sub-interface `Progressible` are described in Section 3.10.1.

[2] See Section 4.6.4 for implementation details of `Chalklet`s and associated classes.

[3] In fact, the PDF converter looks for a saved snapshot image of the Applet to include. When no such snapshot is found, it uses a standard Applet icon to mark the Applet's place. However, the implementation of the server-side board does not store such snapshots yet.

```
knipping@localhost> java echalk.tools.pdf.Board2PDF
usage>
java echalk.tools.pdf.Board2PDF [<key>=<val>...] <lecture dir>

Recognized standart keys and their values:
pdf.conf     <filename>
pdf.media    a[0-6]|letter|halfletter|legal|note|ledger|11*17
pdf.portrait true|false
pdf.bw       true|false
pdf.bg2white true|false
pdf.outlines true|false
pdf.outfile  <filename> (relative to <lecture dir>/pdf/)

Key-value pairs for debugging purposes:
pdf.grabjfifs   true|false
pdf.grabgifs    true|false
pdf.bin2ascii85 true|false
pdf.noflate     true|false
echalk.debug    true|false
```

Listing 6.1: The board-to-PDF converter program prints a summary when called without arguments.

simply split the data into pages at fixed offsets. The distribution of page breaks has depended only on the PDF output paper format, not on the content. The present algorithm uses a greedy strategy. It tries to place the next break as a horizontal cut with as much content as possible put onto the page while no visible[4] board element is being cut. If no such page break is found which uses at least the upper two thirds of the page for content, and a kind of minimal cut is searched for in the lower third of the page. This is realized by computing the amount of "ink" weight of the content by the vertical coordinate. Each visible elementary drawing (a line segment, an image, a typed text) adds its width to the y-interval of its vertical extension. A weighted partition is generated of the y-interval $[0, \infty[$, represented as a list of weighted intervals.[5] The weight of each interval is a measure of how much board content is cut by a horizontal page break which lies in the interval. The weight gives only an approximation, as overdrawings and the rounded caps and joints of line strokes are not taken into account, but in most cases this is sufficient to get pleasing page breaks.

Board content where horizontal cuts as breaks are not adequate due to written lines overlapping vertically cannot be handled well by this method, see for example Figure 6.1. A possible strategy to solve this would be to construct a graph of board elements as nodes and edges between touching elements. Pages are generated by distributing the elements on the pages so that elements of a

[4] Drawings in the board background color, like eraser actions, do not prevent the page break.

[5] While the insertion of the weighted interval may take quadratic run time in the worst case, the implementation used in the converter runs in near linear time for real world lecture data. It uses the fact that almost all board elements have only a small y-offset from the previously generated element. Using the previous insertion position as start position for searching the next position results in a drastic speedup.

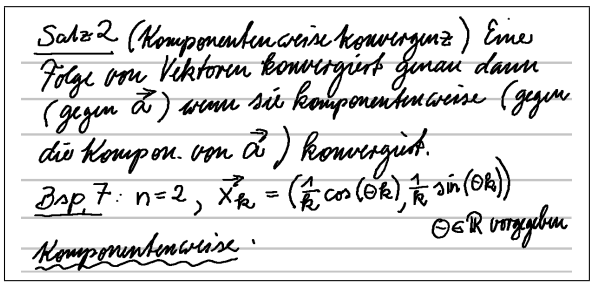

Figure 6.1: Handwritten lines that contain vertical overlay, either due to a sloped baseline or to high ascents or low descents running into neighboring lines, prevent straightforward page breaks using horizontal demarcation lines. Descender lines are plotted in light gray.

connected graph component appear on the same page whenever possible.

6.1.1 PDF Structure

A PDF is organized as a tree of objects. The main element to organize the document is the `PageTree`. A `PageTree` contains `Pages` objects, which list other `Pages` and/or `Page` objects. A `Page` has a list of objects specifying the actual content of the page. All PDF objects are numbered and used in other objects by this reference. The trailer of a PDF file contains a table that maps object references to their byte address in the file. See [Ado03] for details.

The PDF converter outputs a flat hierarchy of PDF objects that keeps the number of objects low. This makes the resulting PDF smaller as the overhead for defining objects is reduced. It also speeds up the process of loading and rendering a PDF. Only a single `Pages` list is used and each `Page` has only a single content object.[6]

The content part consists of basic PDF drawing commands, like line strokes, text drawings, included images, and changes to the current graphic state, like setting stroke width and paint color. The Portable Document Format allows to compress content data[7] with a range of compression filters, including run-length-encoding (RLE), LZW [ZL78,Wel84], and *flate* compression [ZL77,DG96, Deu96]. The PDF converter program uses flate compression[8] on the page content data, except for content objects where this does not save storage space. This is true only for very small content objects, where the compression specifi-

[6]There may be a few extra objects for images and text: images are referenced indirectly in the page contents and the image data representation may be split into several PDF objects, for example a separate object for a color table. If the PDF contains text, the font must also be included as a PDF object, but in the converter's output a single font object is shared by all pages.

[7]To be more exact, PDF *stream data* of PDF objects may be compressed.

[8]The compression factor achieved with flate is usually slightly superior to LZW and much better than RLE. Also, LZW encoding was still protected by patents when the converter application was developed. The US patent [Wel83] owned by Unisys expired on June 20, 2003, and the counterpart patents in the United Kingdom, France, Germany, and Italy expired on June 18, 2004. The Japanese counterpart patents expired on June 20, 2004, and the Canadian counterpart patent expired on July 7, 2004. See [100]. IBM also holds a US patent [MW83] with claims to LZW, that will expire on August 11, 2006.

Figure 6.2: Full GIF image (top left), truncated non-interlaced image (top right), truncated interlaced image (bottom left), and reconstruction of this interlaced image (bottom right). Original image from the *libungif* [53] package, version 4.1.0.

cation overhead of about 20 bytes is bigger than the gain from the compression. Note that compression can be more effective having all the data of a page in a single content object. This way, all of it can be compressed together instead of compressing several chunks individually.

6.1.2 Images

Accessing pixel data by the Java image rendering mechanism is used as a generic method for getting the PDF representation of an image included in a board lecture. The image file is loaded as a `java.awt.Image`, and a `java.awt.image.PixelGrabber` is applied. The converter then tries to get a more compact representation of these data. If the image contains 256 colors or less, a color-lookup table is used. The image data are compressed using flate if this reduces the amount of data. If a color table exists, it is also flate-encoded. Transparent color can also be handled in the PDF converter, as PDF allows to mark color entries as transparent.[9] Semi-transparency, a feature supported by PNG files, cannot be represented in current PDF versions. Even full transparency is not supported by all printer drivers. For example on PostScript printers, PostScript level 3 must be supported for transparency being available.

For GIF and JPEG data, the converter uses faster methods by directly parsing these formats. Still, the pixel-grabbing method serves to handle other formats and as a fall-back, for example for damaged JPEG files, see below. At the time being, the only image format used in E-Chalk recordings that is not directly parsed is PNG.[10]

JPEG Parsing

The encoded image data from JPEG images can be directly included in a PDF image object, as the JPEG decoding is supported by PDF. This is not only much faster than the pixel-grabbing method, it also returns smaller PDF files

[9] The mechanism is called *mask images by color* in PDFs.

[10] PNGs are not used in E-Chalk with the default setup, see Section 2.4. Still, the direct parsing of PNG files in the PDF converter is a desirable future extension.

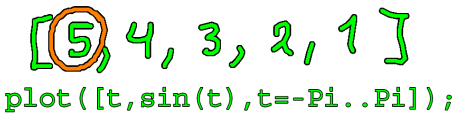

Figure 6.3: Examples of outlined strokes and text element from a PDF transcript.

because the raw pixel data of the DCT encoded JPEG images do not compress well with flate encoding.

Still parts of the JPEG data must be parsed by the converter for two reasons. First, even though the image dimension and color space (like RGB or gray scale) is given in the inline JPEG data, the information must also be stored in extra fields of the PDF image objects. Second, inserting corrupted JPEG image data, like for example from a truncated file, corrupts the whole PDF document; standard PDF viewers do not display such PDFs. For these reasons, the converter scans the JPEG data for dimensions and color space and checks the integrity of the image. If the image is damaged, the pixel-grabbing method described above is used as a fall-back mechanism, as the Java rendering mechanism can also handle truncated files, filling in the unknown portions with a background color. If the data are damaged so badly that even the pixel-grabbing method fails, the image is omitted from the PDF. In this case, it did not appear on the board anyway, because the board relies on the Java image rendering mechanism, too.

GIF Parsing

For GIF images, the complete data are parsed by the converter. Its color-lookup table is extracted, and the LZW encoded-data are decoded[11] to be flate-encoded in the PDF image object. As mentioned above, the flate encoding usually compresses slightly better than LZW. Transparent colors are handled in GIF data. If the GIF is a multiple image file, only the first one is used for the PDF, like it was on the board. If the image data is truncated, the rest of the image is reconstructed as far as possible: if the GIF image uses interlacing, the empty pixels are copied from neighboring rows, see Figure 6.2. For non-interlaced images, the data are filled with transparent color, if that is available from the color table, or with the background color defined in the GIF data. See [Com87, Com90] for details on the Graphic Interchange Format.

6.1.3 Color Conversions

The main application of the PDFs is printing. When the board's background color was not set to white, using the original background color would use a lot of printer toner. For this reason, the background color in the PDF version is set to white by default. In an older version of the PDF converter, background color and white were simply exchanged. The drawback of this method is that it does not preserve the contrast between drawings and background. For example, yellow

[11] As mentioned above, Unisys patented the LZW algorithm. However, decoding data with LZW was always considered free of charge by Unisys. Only software that encoded was subject to patent fees.

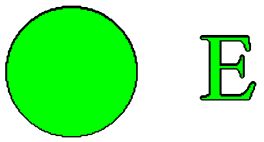

Figure 6.4: For images with transparency, the outline border for the intransparent parts is computed.

Figure 6.5: An erasure stroke crossing a regular stroke interferes with the outline.

writing has good contrast against a black background, but are hard to read against white. Also, drawings on images may be lost, for example if black board drawings on white image parts are used with a black board background. Trying to solve this by converting the background color in images is not recommendable, as this often destroys the image impression and is difficult to handle properly on colors which are similar, but not the same, like shades of white.

In the current version, a black outline for all visible board elements is added[12] when the background color is changed to white. For line strokes and texts this is realized by drawing these elements, slightly fattened and turned to black, below the colored elements. See Figure 6.3 for an example. For images, their outline has to be computed, as transparency is to be handled, see Figure 6.4 for an example.

A drawback of this method is that erasure marks crossing normal drawings are not handled well, as illustrated in Figure 6.5. Unfortunately, there is no way to handle this without the converter computing the rendered page.

Because the PDF version is often printed in black and white, the PDF converter supports creation of grayscale PDFs. Here, outlining board elements is not necessary. Instead, all visible line drawings and text elements are created in black. Images are converted to grayscale. A possible improvement would be to use grayscale colors for the drawings with gray values corresponding to the contrast of the original color to the background.

6.2 Export for Replay in Windows Media Player

To allow recorded lectures to be displayed in the Windows Media Player, a Windows command-line converter to ASF[13] was implemented in Visual C. The E-Chalk audio data is encoded as WMA (Windows Media Audio) in the ASF

[12] This is the default behavior now when the background color is changed. The old method is still available through the E-Chalk settings or from the converter's command-line parameters. Background color conversion can also be suppressed. See Listing 6.1.

[13] ASF is the acronym for *Advanced Streaming Format*, later renamed in *Advanced Systems Format* from the Windows Media framework. It is a general wrapper format for multimedia data, allowing arbitrary codecs to be used for its contents.

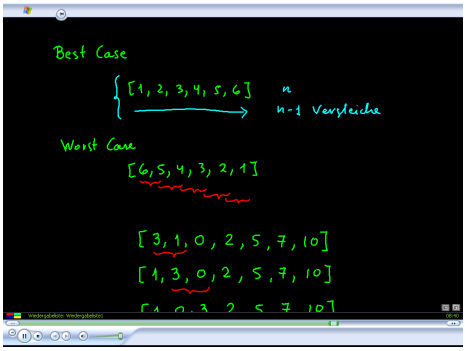

Figure 6.6: A recording converted to ASF replayed with Windows Media Player using a *DirectShow* plug-in.

file. The board data is stored as the timestamped vector data given by the original board events. This avoids huge bandwidth and blurring of sharp edges in the board image that would result from generating a standard video stream format based on DCT compression. On the other hand, this introduces the need for installing a *DirectShow* plug-in for Windows Media Player, enabling it to decode the events for video playback. See Figure 6.6 for an example of a replay.

The implementation of the decoder plug-in can handle the most frequently used board events: line drawings, pasted images, scrolls. Also, text events are partially handled: typed text is supported, but the dynamic line breaking provided by the board[14] is missing, as well as handling of cursor movements and backspace and delete. Clear-all requests, Applets, and undo/redo events are not implemented either.

While this approach requires the user to install a plug-in, it has the advantage of preserving the lossless representation of the board stream and thus enjoying the advantages of clear board drawings with low storage space requirements.

For encoding the audio stream, the standard encoders from the Windows Media API can be selected. However, decoding the E-Chalk-encoded audio and re-encode it with another codec decreases the quality of the audio stream. Also, the available codecs are often less specialized to speech data than the E-Chalk audio codec. The quality of the audio stream usually decreases significantly unless a higher bandwidth is used.

6.3 Export to QuickTime and AVI Video

A program was developed for converting the board and audio streams into a QuickTime or AVI video using the Java Media Framework for encoding the streams. The converter is realized both as a Java command-line application and as `Progressible` to be easily integrated into the E-Chalk main system and to be usable from the E-Chalk command-line console. Available codecs depend on which one are installed on the system the program is running on. The calling

[14] See Section 4.3.

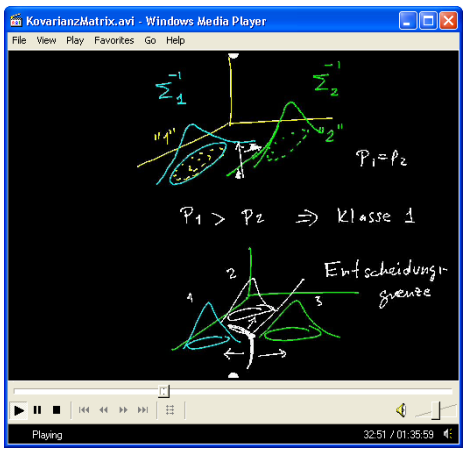

Figure 6.7: A recording converted to AVI replayed with Windows Media Player with 50 % zoom.

syntax for the converter tool is:

```
java echalk.tools.video.Echalk2Video [<options>] <dir> <file>
  <dir>    directory containing the echalk recording to encode
  <file>   the video file to write
options: [-qt|-avi] [-c=<codec>|-c#=<n>] [-fps=<n>] [-w=<n>]
         [-h=<n>] [-v] [-d]
  -qt          output quicktime, default
  -avi         output avi
  -c=<codec>   selects codec of given name
  -c#=<n>      select codec of index <n> in the alphabetical list
               of codecs available for the selected output format
  -mute        ignore audio track from the echalk recording
  -fps=<n>     frame rate for encoded video, defaults to 25,
               may be ignored by some codecs
  -w=<n>       frame width, preserves aspect ratio unless -h is
               given, may be ignored by some codecs
  -h=<n>       frame height, preserves aspect ratio unless -w is
               given, may be ignored by some codecs
  -v           verbose output of progress
  -d           debug mode, makes error output more verbose
```

To get the available codecs listed, two other `Launchable`/command-line applications are included in the `echalk.tools.video` package: `LsAviCodecs` for an alphabetical list of all AVI codes and `LsQtCodecs` for the QuickTime codecs. Most video codecs suffer from quality problems introduced by dropping higher-frequency parts of the frame images, as discussed in Section 1.1. However, note that the encoded board stream is significantly superior in visual appearance to encoded video-taped material, because the encoded signal is much cleaner.

Re-encoding the audio faces the problems already described for the conversion for playback with Windows Media Player, see Section 6.2.

Figure 6.7 shows the playback of a lecture converted to AVI. The original E-Chalk recording has a size of 39 MB for about 96 minutes. 37 MB of the data were allocated by audio data. The board used a resolution of 1016×740. The AVI video is encoded with the *ms-mpg4 v2* encoder using 24-bit color and 25 frames per second and uncompressed 16-bit mono PCM audio. The AVI has a size of 255 MB, 164 MB of that being allocated by the audio.

6.4 Creating Board Snapshots

A tool for creating static board images from board recordings was developed as a side product from the converter to QuickTime and AVI videos described in Section 6.3. Output formats supported so far are JPEG and PPM.[15] The converter is implemented both as a Java command-line application and as an E-Chalk `Launchable`. The calling syntax for the command line is as follows:

```
java echalk.tools.video.Board2Image [...] <lecture dir> <trg>
options: [-w=<n>] [-h=<n>] [-jpg|-ppm] [-p=<n>|-yoff=<n>|-yatend]
         [-t=<ts>] [-d]
-w=<n>    output width, preserves aspect ratio unless -h given
-h=<n>    output height, preserves aspect ratio unless -w given
-jpg      output as jpeg (default)
-ppm      output in portable pixmap file format (*.ppm)
-p=<n>    show scrolled to page no <n> (defaults to 0)
-yoff=<n> show for scroll offset <n>
-yatend   show for scroll offset used at last event
-t=<ts>   load only events up to timestamp <ts>, format
          [[<hh>]:<mm>:]<sec> with <hh>,<mm> integers,
          <sec> float (significant up to milliseconds)
-d        debug mode
```

6.5 Audio Format Updater

Another tool available both as a command-line application and a `Progressible` component is a converter for updating audio recordings from the WWR2 format to WWR3. It was integrated into the main E-Chalk application to make the append mode with the new audio format available on existing lectures recorded with the old format. When a user chooses to append to such a recording, the audio format is upgraded, showing a progress dialog with a cancel option during the process.

When executed as a stand-alone application, it allows to define an audio profile[16] for basic noise reduction and to choose the codec by its bandwidth:

```
usage> java wwr2wwr3.Wwr2wwr3 [<options>] <infile> <outdir>
options: [-p <profile>] [-c <codec>]
  -p <profile> audio profile for sound quality improvement
  -c <codec>   one of {128, 64, 48, 32} kbps, 128 being lossless,
               default tries to conserve bandwidth
```

[15] *Portable PixelMap* (PPM) is the color image format from the *netpbm* [67] package.
[16] See Section 5.2 for creating and using audio profiles.

When executed as a `Progressible` from the main E-Chalk system to upgrade the lecture to append, the system assumes the audio profile selected to be applicable to the old recording as well: speaker and recording equipment should be the same in both. The current profile will be used for the conversion. The upgrading process chooses the lowest bandwidth for the codec that is not lower than the original bandwidth. This should preserve the original quality without increasing the data volume too much. The same behavior is used for the command-line application when no codec option is given.

6.6 Repairing Damaged Recordings

From regular use of the E-Chalk system the demand arose for a tool to restore damaged recordings. At Technische Universität Berlin, several recordings were damaged due to a faulty memory chip in the machine running the E-Chalk server. The recorded data were corrupted by bit flips, and sometimes even the OS crashed. The recording was then shut down without properly closing the output files, so that nonsense data were appended.

A command-line tool was developed for checking the integrity of audio and board data and restoring them by replacing damaged audio packets with silence and dropping invalid board events.[17] The tool can handle both E-Chalk audio formats, WWR2 and WWR3. Checking and repairing video is not yet supported.

```
usage> echalk.tools.checker.Main [<options>] <lecture>
options: [-o<outdir>] [-na|-va] [-nb]
-o<outdir> write fixed lecture to the given directory
-na        do not check/repair audio data
-va        output verbose audio packet info
-nb        do not check/repair board data
```

Replacing the damaged audio packets with "silent" packets prevents board and audio from becoming unsynchronized, unless the audio data is so badly damaged that the number of packets cannot be identified anymore. In this case, synchronization must be adjusted manually using the Exymen editor, see Section 6.13. A possible extension would be a capability to repair board events where the redundancy of the event representation allows reconstruction.

6.7 Import of PowerPoint Presentations

With the abundance of material already produced as PowerPoint presentations, there is considerable demand to import this material into other lecturing tools. The *eClass*[18] system includes the *TransferMation* tool for automatically converting PowerPoint presentations to images and importing them into *eClass*

[17] Note that damaged board events in unrepaired lectures do not prevent the recording from being used for replay or append mode, as the board event interpreter just skips unrecognized events with logging a warning to its error stream, unless the option to abort on invalid events is explicitly set. The same holds true for the PDF converter and for the events interpreter used by Exymen.

[18] For a description of *eClass*, see Section 1.8.5.

sessions. Written as a wizard interface in Visual Basic, it runs only on Windows98 [Bro01].

Another example is the `ppt2aof` [1] add-on for PowerPoint, which enables PowerPoint to export the presentation to the format used by the *AOF*[19] tool `AOFwb`, a `.wb` index file and GIF images of the individual slides.

For E-Chalk, a program originally developed as a student's project will convert Microsoft PowerPoint presentations into E-Chalk board macros. The `ppt2echalk` converter is a Windows command-line executable written in Delphi, relying on the MS PowerPoint automation API to convert the presentation's slides into image files. This means the converter program needs PowerPoint to be installed on the system and running the converter starts PowerPoint, though the application is partially hidden since as it runs in iconified mode.

The resulting macro adds the slides one after another to the board, from top to bottom. Depending on the converter's command-line options, the macro contains scroll events to change from one slide to the next:

```
usage> ppt2echalk [<ts>|(<ts>)] <lecture dir> <out dir>
```

When called without any options, all the slides are added to the board immediately and the macro ends, allowing to scroll the slides at the user's own pace.

When a number `<ts>` is given, it denotes the time in seconds that the macro pauses between slide images before the next slide is added and a scroll event is issued to move to the new slide image, simulating a slide show with automated transitions. When the number is given in brackets, it will only be used on those slides for which the presentation does not define its own transition time.

The board width the macro will produce is determined by the width of the output slide images. The converter provides no option to choose this size, as the PowerPoint automation API does not (currently) allow to change the image dimensions. To provide control over the macro's dimension, the slide images would need to be scaled by the converter program, a feature that has not been implemented so far.

6.8 Keywords from Handwriting Recognition

To automatically generate keywords for indexing purposes, both speech recognition on the recorded audio data [HKW02, Hür03] and handwriting recognition on the board data may be used. For the given purpose, even a relatively low recognition rate is sufficient.

The performance of the handwriting recognition that comes with the Tablet PC Edition of Microsoft Windows XP has recently been tested with recordings from the *Classroom Presenter* described in Section 1.7.2. The recognition engine was tested on slide annotations from five lecture recordings, with segmentation done by manual preprocessing. Recognition was reported as "good" (68 %), even though the lectures are considered to be a difficult recognition task by the authors [AHP+04a].

An application has been developed for E-Chalk to extract text from board writing using a handwriting recognition [The04a]. Like in the *Classroom Presenter* project, it uses the handwriting recognition that is part of the Microsoft

[19] For a description of *AOF* and its tools, see Section 1.8.6.

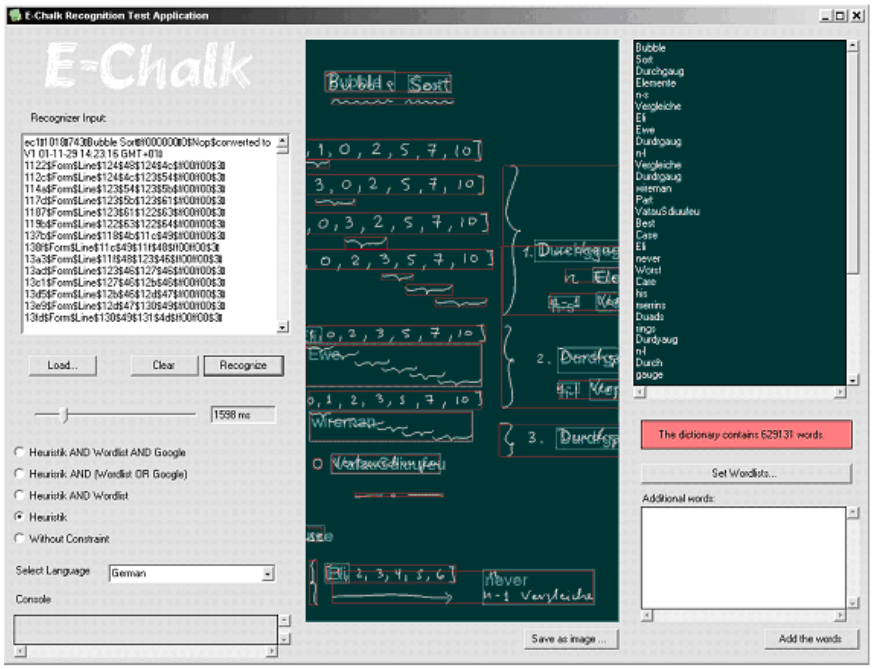

Figure 6.8: The indexing tool as a GUI-based application with control output for the developer. Figure from [The04a].

Windows XP Tablet PC Edition and Microsoft Windows CE. The analyzer works as a post-processor for recordings, implemented both as Java library and as a stand-alone application with a graphic front-end. The application is shown in Figure 6.8.

The recognition process performed rather poorly in distinguishing between drawings and text. This part of the segmentation process is handled by the Microsoft Tablet PC SDKs `Divider`, see [63]. This caused drawings to be interpreted as text, cluttering the good recognition results with random ones. Several approaches for a post-processing have been tried in the project to remove the bad recognitions from the output. Simple heuristics on word structure and dictionary lookups were applied. Unfortunately, dictionary approaches often remove the most significant of the correctly recognized keywords since more unusual words are more important for indexing purposes. See [The04b] for details.

Any further development of this tool should improve the `Divider`'s strategy. A likely reason for the poor performance of the `Divider` provided by the SDK is that the development kit currently does not allow to tag the strokes generated from the board with timestamps.[20]

[20]Strokes fetched from a connection with a mouse driver are generated with time information, but there is no method to add them to strokes generated synthetically. The SDK allows to associate extra information with its `Stroke` objects by lookup keys. It it possible that a key already exists for timestamps, but none is given by the current SDK documentation.

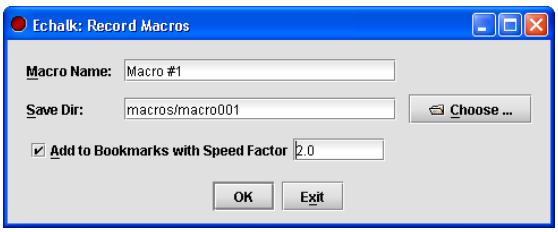

Figure 6.9: The setup dialog of the macro recorder.

6.9 Macro Recorder

The format for a board macro is the same as for a lecture containing a board stream. Only Applet and clear-all events are disallowed in board macros. Macros are basically a short lecture recording with the board being the sole recorded stream.

The macro recorder application allows to create a sequence of macros using the current board settings from the main application. To enforce compatible assumptions between the macro recording and the board session, a macro is only available at the board if they use the same background color and the width is not bigger than the current board width. Note that macros recorded with the macro creator are available on recordings with the current board settings.

When the macro recorder is started, it opens the dialog shown in Figure 6.9. It allows the user to define the storage path and macro name as well as optionally appending the macro to the macro bookmark list. The entries for macro name and path are preset to generic names, enabling the user to start recording immediately. If a previous recording made is encountered, the name is generated from the previous recording's name plus a trailing number, or by increasing an already existing trailing number. Path name generation is done the same way, only adding capabilities to handle name clashes with existing files. Path information is saved from the last macro recording and used to determine the preset to be displayed at the next macro recorder start.

When the user hits *OK*, the application starts recording after checking for any write conflicts, like possible overwrites to be confirmed by the user or any storage permission problems. The code that prepares a directory for recording and starting the recording components is part of the main application described in Chapter 3. With the main E-Chalk application being a `Launchable`, the macro creator just uses it as a subcomponent to be started. Normally, this would show the main setup dialog. To prevent that from appearing, certain configuration properties are redefined. Others properties are used to suppress dialogs like the *Append/Overwrite/Cancel* dialog[21], as the macro recorder already handles such conflicts itself. The redefinition of the properties is realized by calling the `Launchable`'s `init` method with arguments. These arguments are handled exactly like the command-line parameters for E-Chalk being executed as an application, see Section 3.1. Parameters in the form <*key*>=<*value*> are used to change the configuration properties. Parameters `-X`<*key*>=<*value*> are used to modify the recording settings, to assign title, and target directory, and to

[21] See part on Recording in Section 2.4.

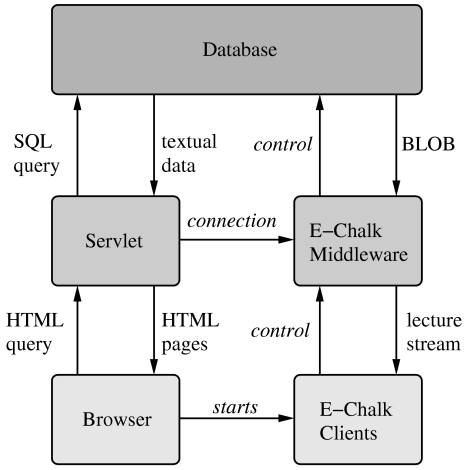

Figure 6.10: Communication between browser and database.

set the recorded streams to board only, as well as to disable macro-incompatible board actions (clear-all requests and Applet usage). To prevent these settings from being stored in the standard `echalk.conf` file and thus being effective at the next E-Chalk start, the macro recorder creates a temporary copy of the file and assigns the copy to the `Launchable` as a settings file through setting the `echalk.conf` property by an `init` parameter.

When the user finishes the recording, the macro creator dialog opens again, with new default values set for title and target directory, allowing to record another macro until the user chooses to close.

6.10 Automated DB or LMS Storage

Already in the early stages of the system's development, experimental support for creating repositories of E-Chalk recordings was implemented. A tool for automated storage of lectures into an Oracle database was developed in addition to a middleware to provide access to the replay feature [FKR02]. The insertion tool was developed as a command-line application, but it was also integrated for automatic call in some early E-Chalk versions. The user interface was realized as a Web front-end for a search engine, allowing to search for lectures by different criteria, such as lecturer, topic, dates, keywords, etc. The front-end supported direct replay of the search results in the browser, as illustrated in Figure 6.10. The system also included management of the clients' Jar files, delivering the latest compatible client version instead of the client code used at the time of the recording.

Since Oracle database supports random access to stored BLOBs, it is possible to use the fast-forward/rewind audio methods described in Section 7.3.1. While the audio format used at that time was WWR2, lacking a jump table for random access, insertion in the database created a similar data structure to enable random access by time offsets using the middleware. User administration for access restriction to the recordings are also featured. Administration of the

Figure 6.11: Board output generated by handwriting synthesis for the input string `"handwriting synthesis"`.

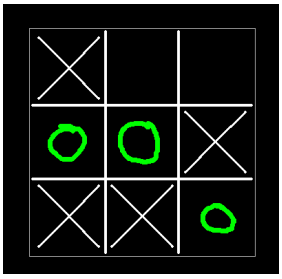

Figure 6.12: Tic Tac Toe test chalklet. The computer places crosses, the human player circles.

database content and the users is done via another Web interface.

This is not the only repository support tested with E-Chalk. In the context of a lecture given in the winter term of 2003/04 where E-Chalk and a Learning Management System (LMS) were employed[22], a GUI application was created for storing into the LMS, namely BlackBoard [4] Version 6. However, the implementation of the tool remained at a very experimental stage.

6.11 Handwriting Synthesis

Another application that started as a student project converts ASCII text to handwritten text on the E-Chalk board. The output features script-style connections between different characters of a word, and strokes are timed. Figure 6.11 shows an example of the output. The handwriting synthesis application was written as a command-line tool that generates an E-Chalk lecture usable as a board macro. Another application provides a graphical interface to define character shapes and the types of connection with neighboring characters.

The application can be thought of as an inverse handwriting recognition software, generating human-like output from machine input. A later version might be used in future E-Chalk implementations to replace all remaining ASCII output on the board, like the text output of computing algebra servers or from CGI queries. In addition, a converter for text to board strokes should be added to the E-Chalk API for the benefit of chalklet developers.

[22]The lecture is the course on Neural Networks evaluated in the study described in Section 8.4.2.

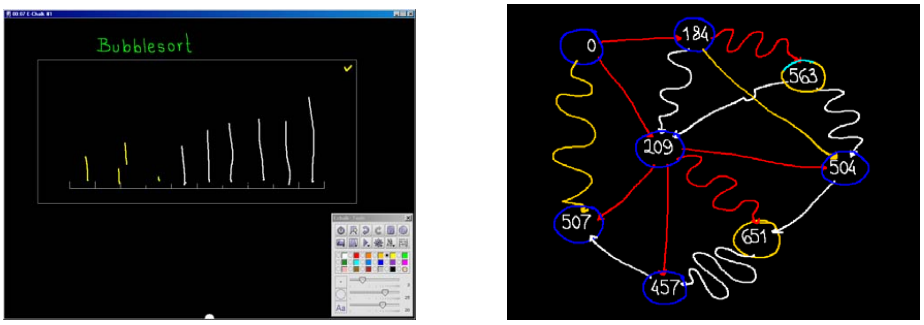

Figure 6.13: Chalklet for animated algorithms in mid-operation. Left: Sorting handdrawn lines according to their height with Bubble Sort. Right: Finding the shortest path in a weighted graph with Dijkstra's algorithm. Figures from [Esp04].

6.12 Example Chalklets

6.12.1 EchoChalklet and TicTacToe

Two chalklets that were not developed for use in actual lecturing are nonetheless included to serve as examples for developers using the E-Chalk API to create their own chalklets. Their source code is linked in the documentation of E-Chalk's Java API classes. The `EchoChalklet` just reflects any `Stroke` it receives at a center line in its input area, with a certain delay to the input. The delay and whether the reflection is to be done horizontally or vertically are parameters for the chalklet's bookmark setup.

The `TicTacToe` chalklet allows to play a game of Tic Tac Toe against a computer opponent, see Figure 6.12. The computer does not play an optimal strategy, it only pre-computes the results for one move of the opponent, giving human players a chance to win. The chalklet does not take any parameters. Which player gets the first move is determined at random. As long as the match has not reached a decision, undos are handled by the chalklet. Marks to positions already set or marks that can be attributed to more than one position are considered invalid and are deleted with background-colored repaints by the chalklet.

6.12.2 Animated Algorithms

A number of animated sorting and graph algorithms have been realized as chalklets [Esp04], see for example Figure 6.13. They aim at visualizing algorithms in computer-science teaching. These chalklets take user strokes as handdrawn input to the algorithm, e. g. a collection of strokes to sort according to their height, or handdrawn graphs, using the lengths of the connecting edges as weights.

[Esp04] also presents an engine called *Flashdance* that produces algorithm animation in Flash from pseudocode input and provides a framework for generating E-Chalk macros via *Flashdance*. The algorithm uses handdrawn strokes from an E-Chalk recording as input. The event file with the strokes and the definition of the algorithm are fed into an animation generator, producing a

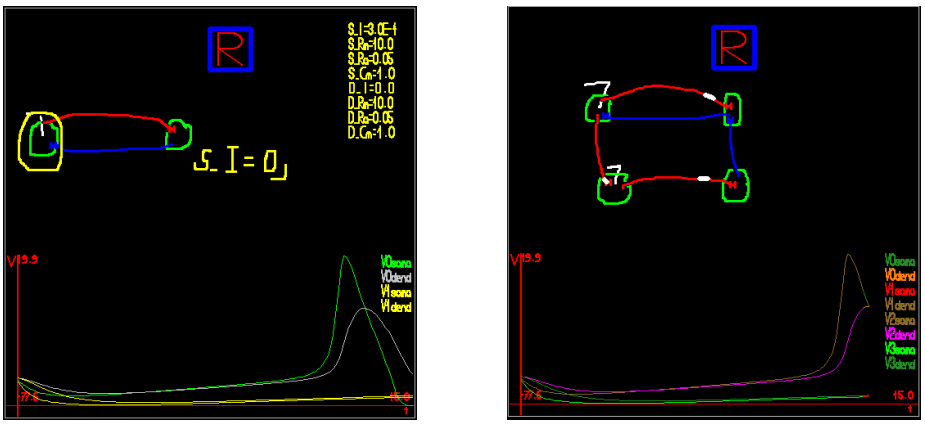

Figure 6.14: Two snapshots from the Neural Networks simulating chalklet. Images courtesy of Olga Krupina.

Flashdance script that is then converted to an E-Chalk macro. This allows to generate macros showing animated algorithms. Unlike the chalklets described above, these animation macros are not interactive on the board, and their operations have to be generated in advance to the teaching. However, in [Wat04] an approach is researched to extend the *Flashdance* engine to produce chalklets.

6.12.3 Simulation of Neural Networks

A solution for simulating biological and pulse-coded Neural Networks is presented in [Kru05]. The author has developed a board chalklet that allows to define networks by drawings and pen-drawing actions for loading parameters of neurons. The chalklet can graphically simulate these networks. The numerical simulation underlying the chalklet visualization is computed by a server that the chalklet connects to using Java RMI calls. See Figure 6.14 for example snapshots.

6.12.4 Simulation of Logic Circuits

A chalklet has been written for teaching computer-architecture principles, which is capable of simulating hand-drawn logic circuits [Liw04]. To classify the user input, the chalklet uses a multilayer neural network trained with backpropagation. It recognizes clock elements, And, Or, Not, Multiplexer and Demultiplexer gates, and connections between the control elements. Circuits can also be saved to the local disk for reuse as "modules" in other logic chalklet instances. Elements that are touched by the eraser are completely erased by the chalklet, repainting the element using the background color. The recognized circuit is simulated by using the Hades framework [Hen98] [32].

Handwritten zeros and ones set the inputs of the circuits for simulation. Alternatively, the user can choose an animated simulation for all possible inputs, optionally combined with a state-timing diagram, as shown in Figure 6.15.

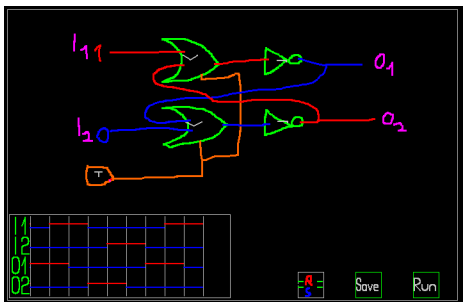

Figure 6.15: The logic circuits chalklet simulating a clocked RS Flip-Flop. High inputs are shown with red lines, low inputs with blue lines. A state-timing diagram for the circuit is shown at the bottom left. Figure from [Liw04].

The chalklet usability has been tested in a video-taped laboratory test.[23] Eight users with different background knowledge about logic circuits and the E-Chalk system got a 15 minute introduction into logic circuits and using the chalklet. Then they received a number of exercises, requiring them to use the chalklet. Afterward, they were interviewed for their experiences and opinions. Some minor interface flaws in the chalklet were identified in the test, notably of features not being being recognized as connected by wires (due to the recognizer's distance tolerance being to small) and in the workflow for storing circuits for reuse. The feedback resulted in certain adjustments being made to the chalklet. See [Liw04] for details.

6.12.5 Python Interpreter

Python [vRD03] [78] is a programming language that was conceived as a teaching language. The technical overhead from language syntax is very small compared to most other languages. Python programs look very similar to algorithms in pseudo-code notation. The idea of building a Python-interpreter chalklet came in mind when Computing Science lecturers were looking for a tool to interactively teach programming and algorithms with E-Chalk.

Hence, a Python interpreter chalklet was developed [Ste04]. Its supports interaction types of a Python programming environment interface to be used during a lecture. It accepts handwritten Python commands that are then recognized by the chalklet. Recognized lines can be combined and reordered into a Python script by drag-like pen drawings. Recognized writing can also be inserted at arbitrary program positions, again by a drag-like gesture, and recognized letters can be deleted by a strike through. As a fallback input possibility, the chalklet provides a keyboard-like input. It shows all letters and takes any letters stroked out as input. The scripts can be run using the Jython [48] interpreter, a Python implementation in Java. A running script can be stopped by the lecturer, allowing to exit from programs with long runtimes or with infinite loops. As scripts and program output can be too long to fit in their chalklet areas, horizontal and vertical scrolling by horizontal and vertical pen strokes is supported.

[23] For a short introduction to usability laboratory tests, see [Shn98].

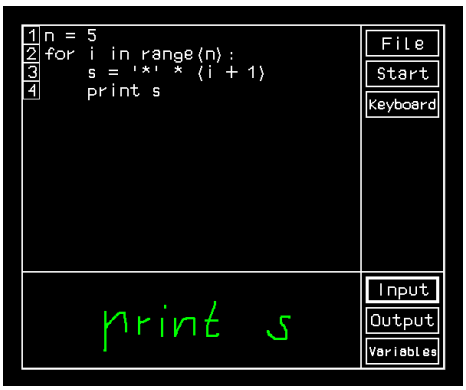

Figure 6.16: Python interpreter chalklet. At the top, the program "window", at the bottom an area used for both handwriting input by the user and program output for running Python scripts. Image courtesy of Henrik Steffien.

The handwriting mechanism uses the Microsoft Tablet PC recognizer.[24] To avoid having to run E-Chalk on Windows XP Tablet PC edition, a Java interface to the recognizer engine has been developed which allows to be queried by RMI calls. With this approach, one can use the Python chalklet on any platform with using the handwriting recognizer by remote calls, as long as the server program can be connected via a network connection. When E-Chalk actually runs on a Tablet PC Windows, the server can be started locally and queries can be made without a network connection.

6.13 Post-production with Exymen

Although E-Chalk has been conceived as a tool for capturing live and spontaneous lectures, instructors would like, of course, to edit the result of a lecture in order to correct errors or just to eliminate superfluous parts. By editing, recordings can be shortened, parts of lectures can be reused and recombined, and lectures can be dubbed. For this, an editor capable of handling the three multimedia streams used by E-Chalk is needed: audio and video streams as well as the event-based board format must be supported.

Instead of writing a specialized editor for E-Chalk, a generic editor for streamed data formats called Exymen [Fri02a, Fri02b] [25] has been developed. It provides a single GUI for all kinds of multimedia editing and a framework to represent multimedia data. A number of formats are handled by plug-ins filling the abstract data containers of the Exymen API and providing editing operations on them. To make for convenient extension of the editor, a powerful plug-in management system has been included.[25] Figure 6.17 shows a

[24] The Java connection to the Tablet SDK is based on the work described in Section 6.8.

[25] Speaking of the E-Chalk client, a major point in usability was to avoid having to install plug-ins or any other special viewer software, see Section 1.2.

On the other hand, employing a plug-in architecture is not a problem for the server side, where the Exymen editor is used. The server software needs to be installed anyway, and all necessary Exymen plug-ins are installed automatically by the E-Chalk server installer. The

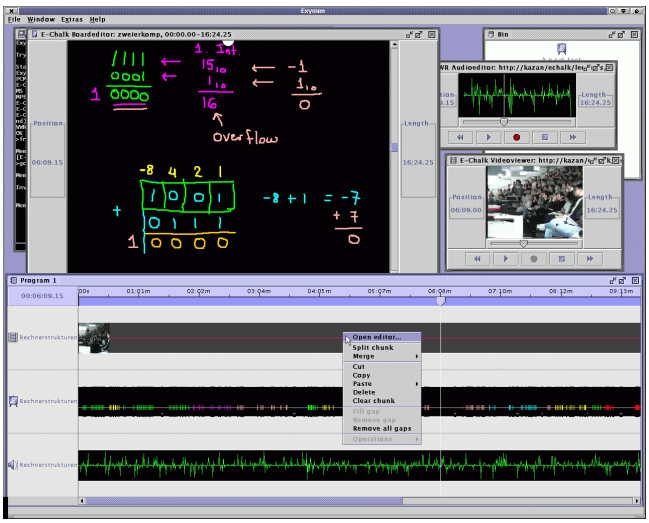

Figure 6.17: Exymen being used to edit three E-Chalk streams: video, audio, and board. The streams are shown at the bottom along a timeline. At the upper left, the state of the board for the timeline's cursor position is visible. The frames at the top right and middle right show the video and the audio signal curve for the same time. Figure from [Fri02b].

screenshot of the editor.

6.13.1 Plug-in Management

Programs using plug-ins often irritate their users by consuming a lot of time for loading the plug-ins on each startup, even if the plug-in is not needed for the current session. Also, many applications require a restart of the application or, even worse, of the operating system when new plug-ins have been installed. Exymen avoids these problems by using a solution based on an open-source implementation of the OSGi standard, named Oscar [70]. While the OSGi standard was originally conceived for component management in the field of Ubiquitous Computing, it is also suited for desktop applications. Having been developed for small, mobile devices, it was designed to be small, compact, and efficient. It allows the editor to install and update plug-ins via the Internet or remove installed ones, all while the application runs and without requiring a restart.

Plug-ins can also manage themselves for handling version updates or installing other plug-ins. For example if an Exymen format handler encounters an unknown data format, it may search for a plug-in to handle it, install the plug-in and let it loose on the data. Plug-ins may also extend other plug-ins, providing a kind of modular construction system. The Oscar/OSGi system takes care of all resulting dependencies. On shutdown, all plug-in states are stored to

plug-in mechanism is effectively hidden from the standard E-Chalk user. The same holds true for the SOPA plug-in architecture underlying the audio and video server components. In both cases, the plug-in architecture is only relevant for developers extending E-Chalk.

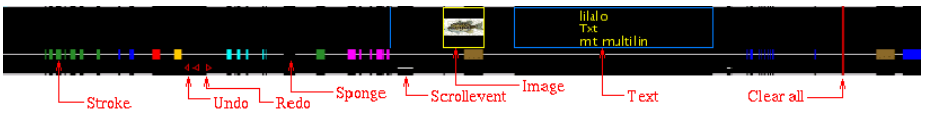

Figure 6.18: Board chunk visualization and explanatory legend. Figure from [Fri02a].

provide for fast system startup without checking the dependencies again.

Download time for installations and updates is low since plug-ins tend to be small. Even including an HTML-based online help, none of the plug-ins described in Section 6.13.3 is bigger than 200 kB.

The approach of Exymen is inspired Emacs philosophy of free extensibility of the program, formulated by Richard Stallman [Sta81]. It motivated the choice of the acronym for "*EXtend Your Media Editor Now!*" as the editor's name.

6.13.2 Data Structures

Media data in Exymen are organized in a hierarchical way, similar to Apple's QuickTime [App01b] format, which also serves as a wrapper for different media types.[26] The top-level data structure is the `Program`, which would correspond to a film reel of the analogous video cutting domain. A `Program` contains a number of `MediaTracks`. A `MediaTrack` is composed of a sequence of `Chunks` (corresponding to a tape snippet in video cutting), ordered along the timeline without overlaps within the track. An Exymen plug-in fills the chunks with its data. It may use an arbitrary representation and has to implement a small number of basic operations on the data, e. g. splitting, merging, and cloning. With these operations, video-cutting operations along the time scale in Exymen become possible. A plug-in also has to provide a graphical visualization along the time scale. See Figure 6.18 for an example.

Optionally, a plug-in may provide media chunk scaling, in both time and space, and define arbitrary effects on its chunks, called filters, for example to adjust the gain of an audio chunk. Plug-ins can provide a `MediaEditor` to record new data. A `MediaEditor` also acts as a replay tool. Optionally, it may allow editing actions in the space coordinates, see for example Figure 6.20.

6.13.3 Editable Formats

Exymen is distributed [26] including a number of standard plug-ins for the following formats:

- E-Chalk board format, described in Section 4.10,
- E-Chalk audio format, both the old WWR2 and the new WWR3 audio, described in Section 5.1,
- E-Chalk video format, described in Section 5.3,
- E-Chalk slide-show format, described in Section 7.5,

[26] The ISO based their file format specification for MPEG-4 on QuickTime, too [PE02].

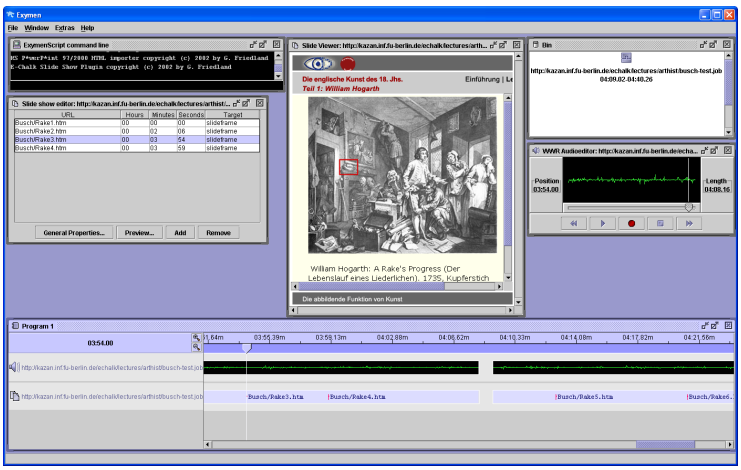

Figure 6.19: Editing an E-Chalk lecture combing a slide show and audio.

- E-Chalk lecture recordings, handled by a meta-plug-in using all four plug-ins listed above,
- JMF-supported audio formats [44], including MP3, QuickTime audio, and PCM audio data (.wav, aiff, .snd, .au, etc.),
- C64 SID audio [Com82, Yan83]. This was integrated mainly for educational purposes on historical computer architecture. The fact that the C64 SID format is a fully computationally complete program format made the editing support of this format quite challenging [JFK03, FJK04a].

The audio plug-ins allow to convert a stream of any of its supported formats to any other of the audio formats listed above. Another Exymen plug-in available with the standard distribution allows to convert Microsoft PowerPoint-Presentations exported as HTML pages to the E-Chalk slide-show format and to be edited with the above mentioned plug-in for the slide-show format.

An example of editing an E-Chalk lecture of a slide show including an audio stream is shown in Figure 6.19.

E-Chalk Board Plug-in

While editing the multimedia data for audio, video, and slide shows is pretty straightforward, editing the board content poses a few challenges. The board stream represents the development of the board content as a sequence of timed events, but events are not independent from previous events. For example, a redo event is only allowed after an undo event and a line drawing can only happen on the visible portion of the board, meaning it is dependent on the offset generated by the antecedent scroll event.

These structural and spatial dependencies are classified in [Fri02a]. It describes how the plug-in manages to constantly maintain the dependencies within a `Chunk` for all editing operations. The main idea behind preserving the non-spatial dependencies is to organize the board chunks' content data into atomic

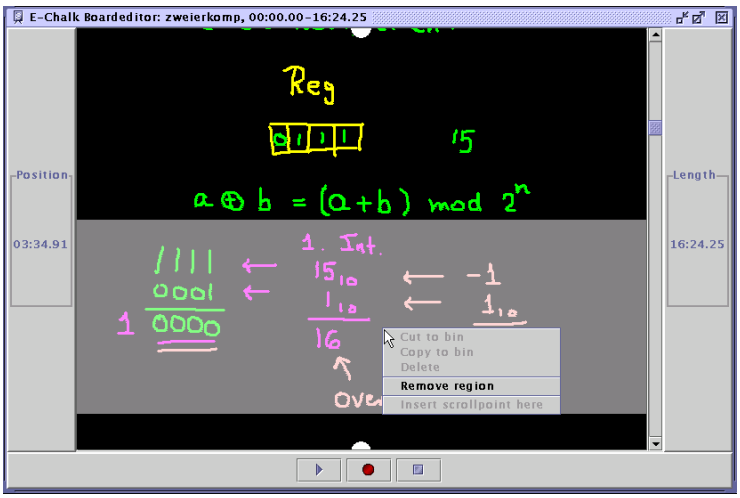

Figure 6.20: Removing a vertical board interval with the board `MediaEditor`. Figure from [Fri02a].

sequences of dependent events.[27]

The `MediaEditor` for the board stream not only allows recording new chunks and replay of the stream. It also lets the user insert scroll events and apply other spatial editing operations. Users may select rectangular board sections for cut, copy, and paste operations, as well as for moving the selected elements by dragging. They may also remove vertical intervals from the board, see for example Figure 6.20.

[27] A coarse separation into atoms would seriously constrain the editing facilities on board data. Fortunately, the atoms are almost always single strokes. Only interactions with Applets and undo/redo operations can create longer atoms.