Chapter 5

Client Architecture

A key concept of E-Chalk is remote teaching over the Internet. This chapter is devoted to technical considerations on E-Chalk's distance teaching facilities and their remote replay. Currently, there are basically three ways to replay E-Chalk lectures: using Java-based client software, using traditional video players, and using MPEG-4.

5.1 Preliminary Considerations

Successful distance teaching requires the awareness of the technical abilities and pre-requisites of the targeted students. For example, when a participant has to download and install client software, the "psychological barrier" for following a remote lecture for the first time can be very high. Furthermore, it might not be a good idea to assume that all students have an Internet connection, let alone one with a high bandwidth. A survey among engineering students in Berlin revealed that while 93 % had Internet access at home, more than half of them had to dial-in with a modem [Friedland et al., 2004c]. Subsequently, broadcasting at different levels of quality and/or splitting up the content into different streams providing the remote viewer with the choice to turn off individual streams is advisable.

5.2 The Java Client

In E-Chalk, remote listeners receive the board content and listen to the lecturer. In some of the lectures a small video of the teacher has been transmitted, too. The small video screen does not deliver very much content-related information, but provides an impression of the classroom in order to achieve a certain "psychological closeness" to the classroom (see Chapter 8). Later, the small-video approach was abandoned in favor of a semi-transparent transmission of the segmented lecturer in front of the board. The reasons for this and the details of this approach form a huge part of this dissertation starting at Chapter 9.

Upon starting the E-Chalk project in 2000, there were three main reasons for the decision to create a purely Java-based Applet client. First, the rendering engine of the server could be reused in the client, thus making it easier to guarantee that the replay looked exactly like the server presentation [Raffel,

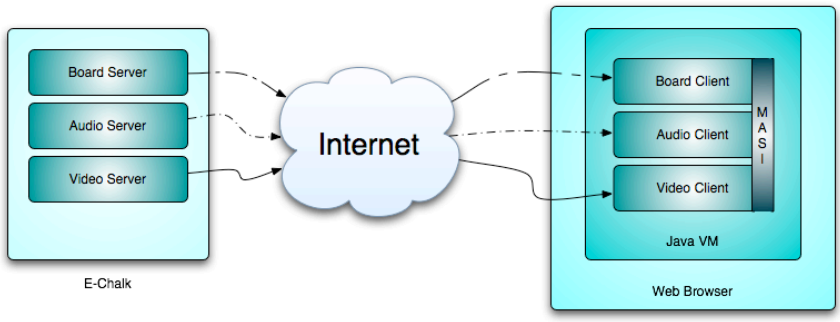

Figure 5.1: Conceptual diagram of the Java-based client replay architecture for live streaming. The content is transmitted directly over different TCP sockets.

2000]. Second, the World Wide Radio 2 Java-based audio client could be used as an audio replay facility (see Chapter 6). Last but not least, as discussed in Chapter 2, most lecture-recording tools require the remote learner to install a special receiving software, usually designed as a browser plug-in. This introduces a psychological barrier for first-time users, compare for example [Nielsen, 1999]. Moreover, remote learners often do not have the skills or even the permissions (for example on campus computers) to install such a client software.

As explained in [Knipping, 2005] Section 2.4.1, E-Chalk generates a directory tree containing the Java client and other resources for replay, including the web page that is to be displayed by the web browser. The E-Chalk client system internally has three modes of operation. In live mode, each client connects to its corresponding server through a socket connection. In on-demand mode, clients use an HTTP connection to receive the files and no E-Chalk server is needed. In local mode, the client just reads files from the local hard disk as they were stored in the directory tree. This last mode is often used by students to save connection costs when they watch a lecture at home. For easier download, the generated files are collapsed into one archive file.

The Java-based replay system consists of a set of independent receiver Applets that synchronize by communicating through the *Media Applet Synchronization Interface (MASI)*. MASI can be used to synchronize several media Applets located in one web page to make them cooperate and deliver the same multimedia content. The E-Chalk system uses this to synchronize the audio, video, and board client. The underlying concept of this Interface is the notion of a frame. All methods that address offsets use frames as their basic unit. A frame is a certain amount of time. Frames are atomic in the sense that there are no fractions of frames. Every Applet has to provide methods to convert the abstract unit frame into concrete amounts of time. The amounts of time should be chosen to be as small as possible to allow optimal control of the streams. MASI provides around 26 methods for synchronization and concurrent stream control, these include methods for pausing, fast-forwarding, and rewinding as well as communication methods and error handling. For a detailed description of MASI see [16]. Although the text refers to "the client", the replay system consists of a board client, an audio client, a video client (for both small window replay and overlaid instructor replay as explained in Chapter 9), a slide-show client, and a console client to provide VCR-like GUI control. Figures 5.1 and

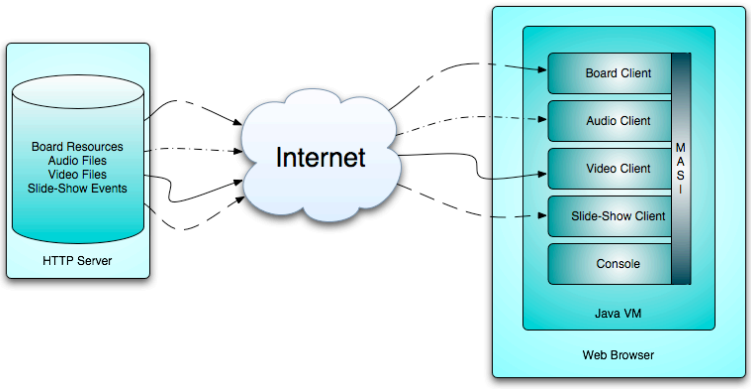

Figure 5.2: Conceptual diagram of the Java-based client replay architecture for remote on-demand replay. The content is streamed from files over HTTP.

5.2 illustrate the basic architectures for live and on-demand operations. Local mode works similar to on-demand mode, with the exception that files are directly read from a local storage device. Figure 5.3 shows the Java-based replay client running in a web browser. All clients are backwards compatible to Java 1.1 to achieve maximum compatibility. For example, on PDAs that use the Pocket PC operating system, Insignia's Jeode Java Virtual Machine can be installed which implements most features of Java 1.1 Standard Edition. This already allows for a Java-based play back of E-Chalk lectures. However, a lot of scrolling is required, since scaling has not yet been implemented in the board client. Figure 5.4 shows an example.

A detailed description of most of the features of the client Applets can be found in [Knipping, 2005], Chapter 7. The description here only summarizes the key aspects.

5.2.1 Board Client

The code for the *board client* is an intersection with the board server code. The board client opens a window of the same size as the board server in the classroom. The strokes are rendered by the same code as in the server. If the resolution of the remote screen is smaller than the server screen, the user is still able to see the entire content by scrolling the board. The board can be scrolled manually in all four directions. The server also sends scroll events that may interfere with a client's scroll action. Therefore the user can choose between server-only scrolling, client-only scrolling, and combined mode. The board server and client use a textual format to encode events, the bandwidth consumed depends on the sampling rate of the drawing device. In practice it varies between 2.5 kbit/s and 6 kbit/s, not counting any image or Applet data. Random seek to a specific time position is implemented using a fast redraw of the events from time position 0 to the desired position. An overview of the format can be found in Appendix C, a detailed discussion can be found in [Knipping, 2005], Section 4.11.

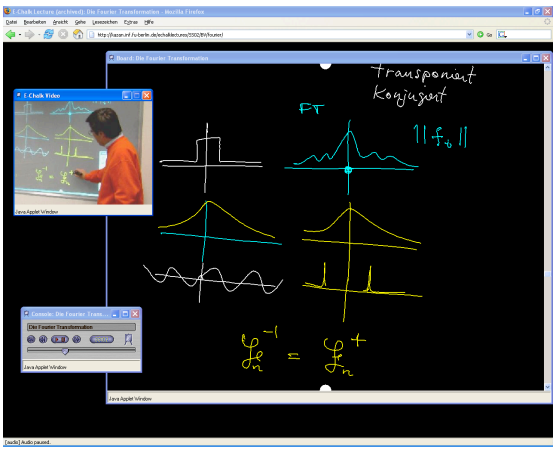

Figure 5.3: Java-based replay of an E-Chalk lecture in a regular browser. The additional video window is used to make the lecture appear more personal. Presenting the lecturer in a second window has several disadvantages that are discussed in Chapter 9.

5.2.2 Audio Client

The current *audio client* is able to decode World Wide Radio 2 (WWR2) and World Wide Radio 3 (WWR3) formats. The format is a packetized ADPCM variant. The bandwidth needed depends on the selected quality and varies between 15 kbit/s and 256 kbit/s. The details on the audio format are presented in Appendix D. Depending on the installed Java version, the client chooses to playback the audio at a sampling rate of either 8 kHz (Java 1.1) or 16 kHz (Java 1.3 and higher) with 16 bits per sample.[1] Random seek depends on mode of operation. In live mode, random seek is impossible. The client just plays the stream back as it comes in. In local mode, random seek is provided using operating system file I/O operations. Because the sizes of the compressed audio packets are not constant, WWR3 uses an index file that contains a list of offsets, each pointing to the beginning of a packet. If this index file is missing (or an old WWR2 file is played back) the client has to start from the beginning of the compressed audio file and iterate over each packet header reading the packet sizes and then skip over the actual packets. When an audio file is to be played back remotely over HTTP, sometimes even skipping is not possible. The only option is to transmit the entire compressed file from position 0 to the desired position. Even though browser caching might help, this random seek method is very inefficient. The reason for this is that even though it had already been specified for HTTP 1.1 [Fielding et al., 1999], it took until about 2003 until HTTP server daemons implemented the `GET` command with the ability to specify a start offset (so-called *partial GET*). In other words, only entire files could be retrieved. Today, all the frequently used HTTP servers allow specifying byte-range offsets for file retrieval, thus enabling an efficient random seek. The audio client then uses the index file to directly retrieve the desired package. The audio client serves as a synchronization master, which means that audio replay is never suspended unless bandwidth restrictions or net congestion interfere with audio

[1] Java 1.2, for a short time, did not support any audio playback in Applets.

Figure 5.4: Java-based replay of an E-Chalk lecture on a mobile device running the PocketPC operating system (former Windows CE).

replay. Board, video, and slide-show client pause when they are too fast or skip (if possible) when they are too slow. An interruption of the audio replay is perceived to be more distracting than most visual distortions [Tellinghuisen and Nowak, 2003].

5.2.3 Video Client

The *video client* is very similar to the audio client. The compression works using JPEG compression in combination with a simple lossy motion compensation. The implementation of the client is straightforward because Java Applets rely on browser functionality and one of the main capabilities of web browsers is to decompress JPEG files. As the Java Applet SDK directly uses the browser's rendering engine to decode JPEGs, they can be decompressed very efficiently.

A small technical intricacy concerns memory usage. Like the other clients, the video client needs a buffering strategy to guarantee continuous operation. However, many browsers restrict memory usage for Applets such that the available memory would not last for caching several seconds of video data. A dynamic cache was implemented that increases and decreases with the amount of memory available.

The client works in two modes. Either it opens a window to play back the video, or it gets a `DrawPanel` from the board client. The later mode is to enable replay of a semi-transparent instructor video in front of the board (compare Chapter 9). Figure 5.5 shows a screenshot. The transparency of the instructor can be adjusted using an HTML parameter. When the instructor is overlaid on the board, the color black is interpreted as transparent and the video is scaled up to fit the resolution of the board. Semi-transparent drawing, however, seems to be a Java bottleneck at the time of writing this dissertation. For example, using a board resolution of 1024×768, the client does not achieve frame rates of more than five frames per second on a 3-GHz Intel Pentium 4.

Since video streams consume CPU and connection resources, the client can close the video stream at any time in order to save bandwidth and processor

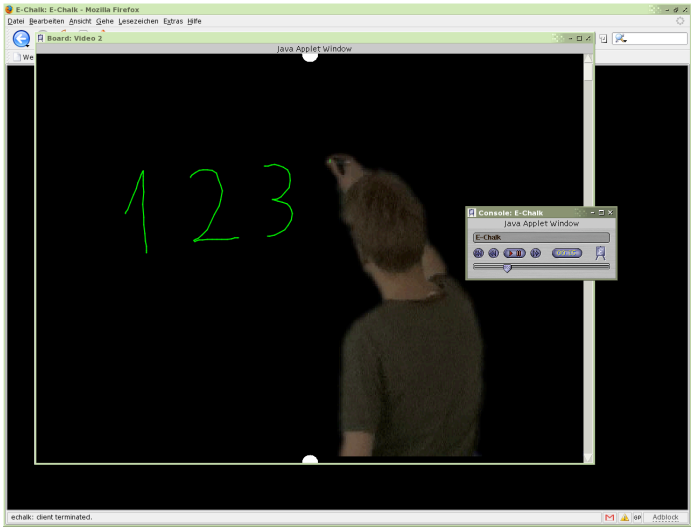

Figure 5.5: The instructor video played in front of the board using the Java client. The approach is discussed in Chapter 9.

time. The required bandwidth for video transmission ultimately depends on the contents of the video stream. A 192×144 video stream at four frames per second needs roughly 64 kbit/s. This means approximately 16 kbit per picture. Random seek is implemented by directly skipping to a specified position (if possible, see Section 5.2.2) and begining to play back from the next frame found. The codec has been built such that motion compensation is rather self-repairing (see Chapter 8 for details).

5.2.4 Slide-show Client

The E-Chalk *slide-show client* is a part of the E-Chalk system that has no corresponding server component. The component had originally been integrated for historical reasons as it was part of the World Wide Radio 2 system. Due to user demands it has remained part of the E-Chalk system. E-Chalk slide-shows can be generated by using Exymen to convert an HTML-exported PowerPoint slide-show into an E-Chalk slide-show. Using Exymen, slide-shows can also be generated from scratch and then edited [Friedland, 2002a]. The slide-show client gets a list of events that contain a timestamp, a URL, and a target frame. A slide-show event at some point during a lecture triggers the browser to open a URL. The URL can point to any content, such as HTML pages, images, animations, or video files. The target frame is used to specify the frame in the browser where the webpage should pop up. The slide-show client, like all others, synchronizes itself with the other clients using the MASI interface. An example of an E-Chalk slide-show lecture generated with Exymen can be seen in Figure 5.6.

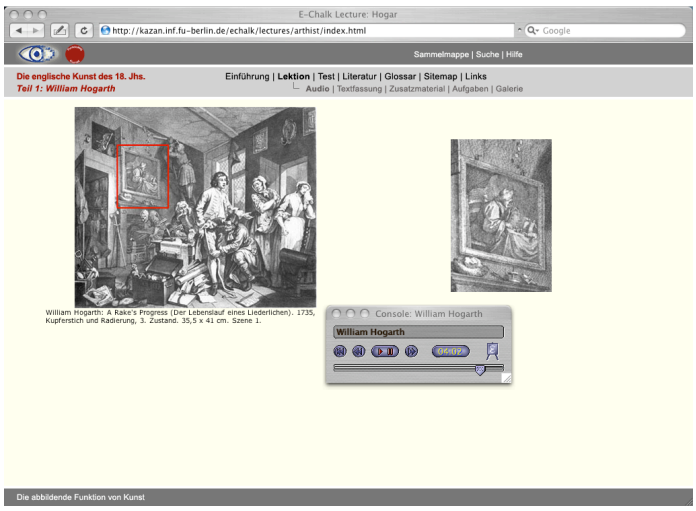

Figure 5.6: An example of a slide-show lecture played back with the E-Chalk client.

5.2.5 Console

The console is only available for on-demand or local replay. The Applet is a GUI interface that uses MASI to provide VCR-like operations to the user. The console allows direct seek to a certain time position as well as a relative seek with operations similar to fast-forward and fast-rewind. The console also allows to pause and continue a lecture. Although a lecture replay can easily be terminated by closing the browser or going to a different URL, closing the console also terminates all clients. The console also provides feedback to the user: It shows the current time position, the length of the lecture, and the time until the ending of the lecture. In order to allow universities to customize the display panel to their corporate identity, the console supports different GUI themes [Knipping, 2005].

5.3 Playback as Video

In addition to Java replay, E-Chalk lectures can also be stored as regular video files by instructing the SOPA framework to use the appropriate nodes during lecture recording. Lectures can also be converted offline using the *E-Chalk2Video* converter. Both methods use the Java Media Framework to encode the content using codecs that are either provided by the framework or by the operating system. Depending on the chosen format, lectures can be played back using any standard multimedia player such as the QuickTime Player, RealPlayer, or Windows Media Player. The content is encoded by internally rendering the board content (plus the optional video of the instructor) using a frame-buffer client. Each internally rendered frame is then passed to a video codec.

When only board data (plus audio) is to be encoded, most video formats provide only very bandwidth-inefficient storage. Video codecs use a frame-by-frame encoding. This results in the stroke data being converted from vector format to pixel format. Even though motion compensation accounts for redundancies,

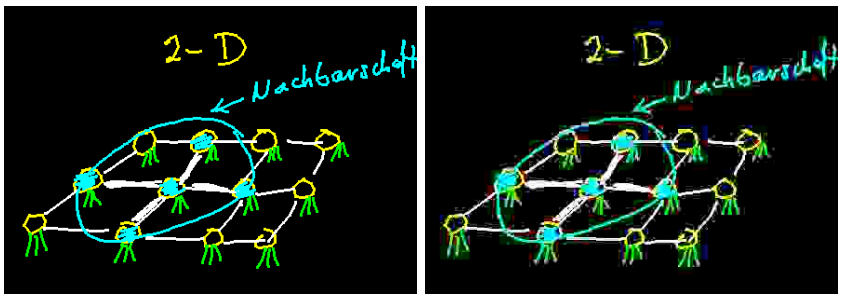

Figure 5.7: Original rendered chalkboard picture (left) and image showing the typical artifacts resulting from quantizing the higher frequency coefficients of a DCT-transformed image for low bandwidth transmission (right).

the storage space is still several orders of magnitude higher (see Section 5.5). Vector format storage is not only smaller, it is also favorable because the stroke semantics is preserved. After a lecture has been converted to video, it is for example not possible to delete individual strokes or to insert a scroll event, without recalculating and rendering huge parts of the video again. Another disadvantage concerns the way most traditional video codecs work. Most often, lossy image-compression techniques are used that are based on a DCT or Wavelet transformation. The output coefficients representing higher-frequency regions are mostly quantized because higher-frequency parts of images are assumed to be perceptually less relevant than lower-frequency parts (see for example [ISO/IEC JTC1, 1993, ISO/IEC JTC1 and ITU-T, 1996, ISO/IEC JTC1 and ITU-T, 1999]). These and similar techniques (for example vector quantization as in [19]) work for most images and videos showing natural scenes where a slight blurring is perceptually negligible. For vector drawings, such as electronic chalkboard strokes, however, blurred edges are clearly disturbing. Figure 5.7 shows the typical artifacts resulting from frequency quantization applied on an electronic chalkboard drawing. Specialized screen capture codecs (such as used in Section 5.5) are able to compress board strokes well with few artifacts. However, images or colorful Applets inserted into the board are reproduced in an unacceptably low quality or result in an unacceptably low compression rate.

In order to work around the disadvantages of inefficient compression, disturbing artifacts, and the loss of semantics, Stephan Lehmann has developed a converter and a plug-in for the Windows Media Player that allows the replay of board data simultaneously with audio. The converter is an experimental command line tool that encapsulates the board events into *ASF files*. ASF stands for *Advanced Systems Format* and is a proprietary container format which is part of the Windows Media Platform (see Section 2.3). The audio data is encoded using Windows Media Audio. The generated file can be played back with Windows Media Player using the plug-in developed by Stephan Lehmann. Random seek is implemented by the player and works just as in the board client. The implementation (Figure 5.8) only served as a proof of concept and replay is limited to strokes and images. Live streaming is not possible.

When the board data is to be transmitted in combination with an overlaid instructor, however, traditional video codecs provide an alternative. Although the instructor image usually takes only about one third of the board surface, it

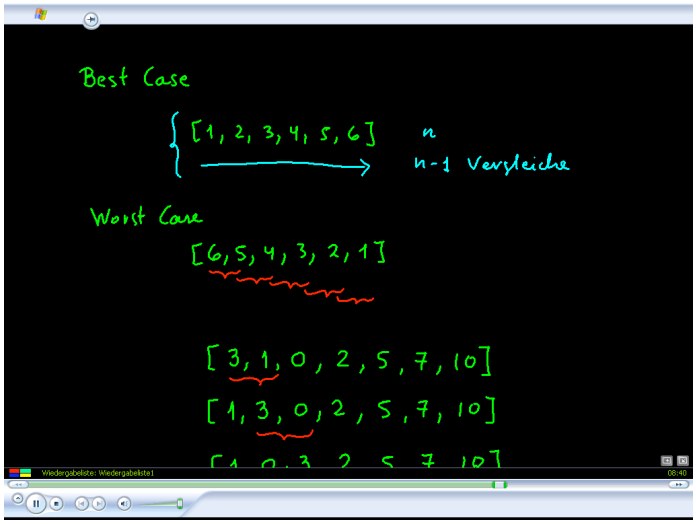

Figure 5.8: A plug-in developed for Windows Media Player makes it possible to view E-Chalk lectures without losing the vector-based storage format.

consumes most of the bandwidth. Since the instructor image has the properties of a natural scene, applying the strategies of traditional video encoders results in an adequate bandwidth reduction (see Section 5.5). Still, board semantics is lost. However, editing board contents in such a lecture is not straightforward. Deleting or inserting board events, such as strokes or scroll events in an E-Chalk lecture with overlaid instructor, may result in a confusing replay because the lecturer's arm movements do not match the created strokes. Synchronized timeline-based editing can still be done, but this functionality is also provided by standard video editing tools. However, encoding E-Chalk lectures using an MPEG-4 video codec is not yet possible in real time and in my experiments many players had problems to show a 1024×768 video with overlaid instructor at the 25 frames per second (again an Intel Pentium 4 with 3 GHz was used).

A general advantage of video-based replay versus Java-based client replay is that many tools are available for conversion and processing. This is especially useful for the replay of E-Chalk lectures on handheld devices. PDAs, mobile phones, and iPods are able to play back different types of video formats. Manufacturers often ship the appropriate conversion and processing tools for their device along with other accessories. The tools encode and scale any operating-system-supported video format down for playback on the small device. The quality of the final replay depends on the quality of the video scaling and on the properties of the device's display. The same is true of the ability to randomly seek into the lecture. Live replay seems to be impossible yet and the conversion speed is far from reaching real time. My experiments have shown that the tools provided around the ITU-T video standard H.263-2000 [ITU, 2000] and the RealPlayer on Symbian OS work very well for E-Chalk. The bandwidth consumed for a lecture with audio and board (no instructor video) is about 16 kbit/s. A video containing audio, board, and overlaid instructor needs 64 kbit/s. Which means that a 90-minute lecture takes about 10 MB or 40 MB, respectively, and can easily be stored on a 64 MB or 128 MB SD-memory card. The display resolu-

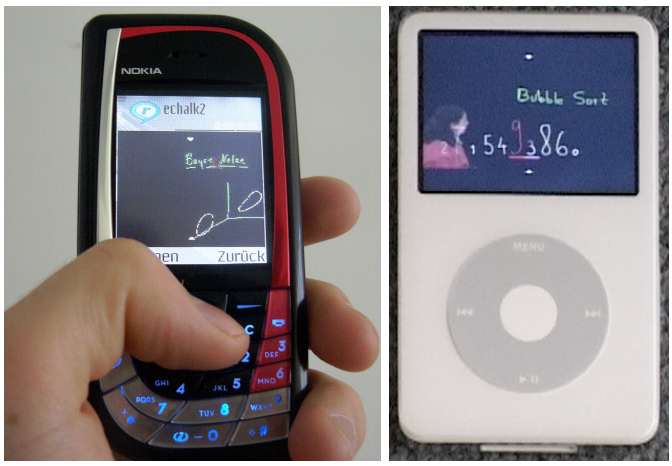

Figure 5.9: E-Chalk replay using the video capabilities of handheld devices. Left: A Symbian OS-based mobile phone. The resolution is 176 × 144 pixels. Right: A video iPod.

tions of mobile phones are usually very low. As a consequence, complex images or small strokes sometimes disappear. Random seek is mostly not supported. Apple's iPod supports playback of several movie profiles of *MPEG-4* [ISO/IEC JTC1 and ITU-T, 1999] using a screen resolution of 320 × 240. The converted E-Chalk lectures have a fairly good quality, and a 90-minute lecture (including audio and overlaid instructor) uses about 90 MB of storage. This makes them easily portable on this device (current video iPods come with 60 GB of storage space). Random seek is supported, too. Figure 5.9 shows E-Chalk lectures played back on a mobile phone and on a video iPod.

5.4 MPEG-4 Replay

When new techniques for video storage and compression are discussed, the video standard that is most often mentioned is MPEG-4. MPEG-4 [ISO/IEC JTC1 and ITU-T, 1999] is the successor of the MPEG-1 [ISO/IEC JTC1, 1993] and MPEG-2 [ISO/IEC JTC1 and ITU-T, 1996] standards. It extends them in many ways. For the purpose of encoding chalkboard events, it contains an interesting part called *Binary Format for Scenes (BIFS)* [ISO/IEC JTC1 and ITU-T, 2005]. BIFS includes support for the vector-based storage of 2D and 3D scenes, as well as some interactivity. The following paragraphs will provide a few technical details on the storage format before discussing the advantages and disadvantages of using MPEG-4 for replay of E-Chalk lectures. Figure 5.10 shows E-Chalk lectures played back using MPEG-4 players. For a short description of the board format and some sample mappings to MPEG-4, please refer to Appendix C. Further work on the conversion of E-Chalk lectures to MPEG-4 is presented in [Jankovic et al., 2006].

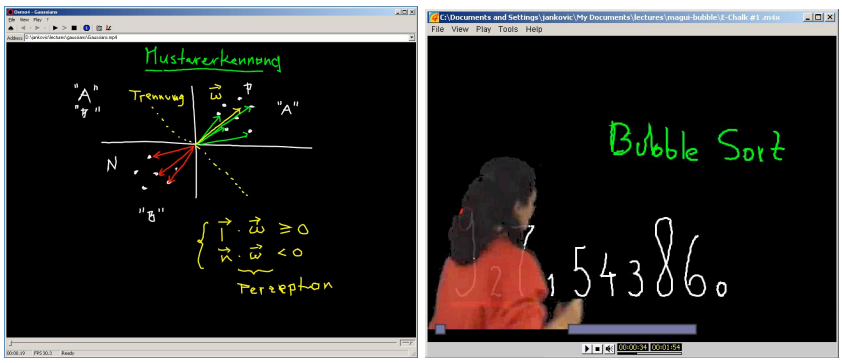

Figure 5.10: Two lectures played back using MPEG-4. Using Osmo-Player (left) and using the IBM Java-based player (right).

5.4.1 Encoding E-Chalk Lectures in MPEG-4

As its name implies, BIFS is a binary representation that has to be compiled from a user-editable source format, called *Extensible MPEG-4 Textual (XMT)*. XMT has two levels: *XMT-A* format and *XMT-Ω* format. XMT-A is a low-level representation that can be easily mapped to BIFS, while XMT-Ω is a high-level format that uses a subset of the tags defined by SMIL [67]. XMT-Ω compiles to XMT-A and then to BIFS. Although XMT is specified as BIFS source format by the ISO standard, its biggest downside is that it is an XML-based format. XML files can only be parsed entirely, since the document opening tag has to be closed by the document ending tag at the end of the file. This makes it impossible to compile XMT files incrementally for live streaming, although BIFS is by itself a streamable format. A solution has been provided by the authors of the GPAC framework [78], developed at the Ecole Nationale Supérieure des Télécommunications (ENST) in Paris. The format is called *BIFS Text (bt)* and is a non-XML-based exact transcription of the BIFS stream. Some users also prefer the format for better readability as the bt document architecture is very similar to XMT-A and the syntax is close to *VRML* [ISO/IEC JTC1, 1997].

BIFS uses trees as basic structures. As a consequence, the basic structures are nodes and there are two types of them: *group nodes* and *leaf nodes*. Group nodes link to a subtree, and leaf nodes are final. Once a node has been defined, its properties can be changed at any time or the entire node can be removed. Mapping the E-Chalk board events to BIFS is straightforward in most cases. Timestamps are directly supported by BIFS: Every node definition or command can be preceded by an `At <timestamp>` command which has the same semantics as E-Chalk's event timestamps. After an obligatory `InitialObjectDescriptor`, which defines some basic parameters such as the video size (in this case the board size), the main scene is described by a group node of type `Transform2D` that forms the root. All other nodes, including a node that defines the background color, are added to this group node. The node has a property called `translation`. Changing the value of this property results in a change of the top left position of the group node. Since all other nodes are children of this node, their position is also changed. Because the viewpoint of the player stays constant, this is a very easy way to implement E-Chalk's `Scrollbar` event. `RemoveAll` events can be implemented by deleting

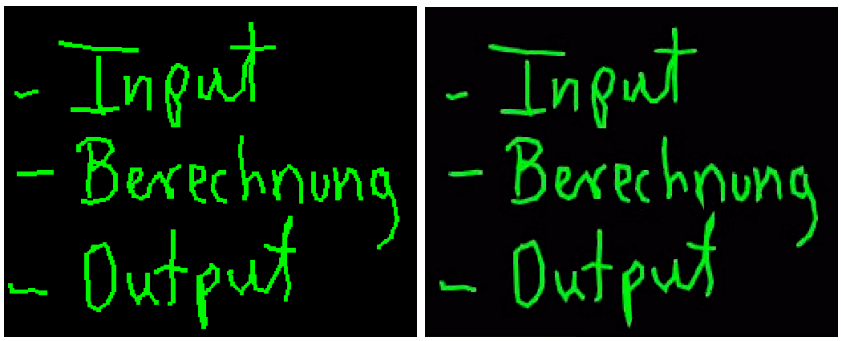

Figure 5.11: E-Chalk client-based replay (left) and the MPEG-4 replay in OsmoPlayer (right). The output differs because MPEG-4 players use stronger antialiasing and draw angles of connected line segments differently.

the group node defining the scene, which also triggers the deletion of all its child nodes. After this, a new group node is defined. `Undo` events can be implemented by deleting the last inserted node, `redo` by again defining and adding the last deleted node. Images can be directly placed into the board by adding a rectangle object at the appropriate position and then placing the image on top of it as a texture. Later image updates are directly supported by BIFS as any node can be updated, so Applet replay can be easily implemented. Text events can be inserted using a `TextNode`. However, it is impossible to guarantee that the appearance is identical to the board server appearance because Java fonts may have a different appearance compared to the fonts the MPEG-4 player is using. Typing text is mapped by consecutively changing the string presented by the text node. In order to draw strokes, connected line segments defined by the `Form$Line` events are consolidated into a polyline and then drawn as an `IndexedLineSet2D`. However, the output in the player is not pixelwise identical to the rendering output of the Java-based E-Chalk client. The main reason for that is that MPEG-4 players use stronger antialiasing and draw angles of connected line segments differently. Figure 5.11 shows a comparison. Finally, the output differs from player to player because every player uses slightly different rendering methods. Video and audio tracks are added by adding appropriate nodes to the root of the tree (Appendix 5.4 shows an example). Overlaying the instructor video is directly supported since layers of different content are a key concept in MPEG-4. The video can be scaled or put at different positions on the board. The transparency of the video can be either hard coded or, using a `TouchSensor`, interactively controlled by the user during playback.

5.4.2 Practical Experiences

In theory, the MPEG-4 format seems to provide a very good representation for the storage and transmission of E-Chalk lectures. In our experiments, however, we experienced several disadvantages. Although many programs are available that are described as capable of playing back MPEG-4 content, most of them, for example the QuickTime Player, RealPlayer, Windows Media Player, or Apple's Video iPod, only support movie profiles and are not able to play back BIFS content. We identified three players that are capable of playing back BIFS

content in combination with audio and video, namely the *Osmo-Player* that is part of the GPAC Framework, a Java-based player that is part of the *IBM MPEG-4 Toolkit* [81] called *M4Play*, and a plug-in for Windows by Envivio [75] that adds this functionality to the Windows Media Player and the QuickTime Player.

Both Osmo-Player and IBM's Java-based player do not support random seek for BIFS content. Fast-forwarding and rewinding can only be implemented manually using a `TouchSensor` and XMT-Ω's `accelerate`, `decelerate` or `reverse` commands. Video codecs generally do not support so-called α-transparency, which means that they work with a three-byte RGB representation for each pixel and not with a four-byte α-RGB encoding, like the graphics hardware. For this reason, encoding transparency in the instructor video itself is currently impossible. However, MPEG-4 supports tagging colors as transparent. Shades of transparency needed for sub-pixel-accurate segmentation (see Chapter 10) cannot be used. For encoding E-Chalk lectures, black is tagged as transparent. Osmo-Player, however, does not yet support the transparency tag. The player uses a different strategy: The video is overlaid onto the board by pixelwise mixing the colors of the two layers. This results in a darkening of the board strokes (mixture with black) in the areas not occluded by the instructor and other undesired effects. M4Play supports transparency; however, as it is Java-based, the IBM player drops many frames when playing back such a video. It has the same problems as the Java-based E-Chalk client (see Section 5.2.3).

Streaming of MPEG-4 content over HTTP requires the conversion from plain MPEG-4 to a format called `m4x`. These files contain so-called "hints" that enable partial playback of MPEG-4 files. Of course, the conversion itself does not work incrementally. In other words, streaming of MPEG-4 files is only possible after lecture recording has been completed. Although the GPAC Framework allows incremental compilation of BIFS content using BIFS text, it does not yet allow for incremental creation of the audio and video track. Live streaming is usually done using the *Realtime Transport Protocol (RTP)* [Schulzrinne et al., 2003], and MPEG-4 supports the streaming of BIFS content using so-called *BIFS commands*. Although the computational needs for the generation and conversion of MPEG-4 BIFS would easily allow it, there is no program or framework available yet that supports live encoding and streaming of content that consists of BIFS, audio, and video. One reason for this might be the license policy [88] connected to the MPEG-4 standard which requires every application generating MPEG-4 content to pay a royalty fee. The policy has often been critizised as being the primary reason for the slow adaptation of the standard, see for example [87].

5.5 A Note on Bandwidth Requirements

The following short experiment is to support the discussion of the possible approaches for E-Chalk lecture replay. A sample lecture [45] was converted into different formats. The lecture contains only chalkboard strokes, no images, no texts, and no Applets. The lecture was held using a digitizer tablet with a high resolution and relatively high sampling rate – thus generating many events (see [Knipping, 2005] Section 4.11 for a discussion). The experiment was conducted in two runs, the first run containing no video and the second run

Format	Board only	Video overlay
E-Chalk format	1,861 kB	277,102 kB
MPEG-4 BIFS	1,662 kB	57,839 kB
Windows Media ScreenCapture v9	2,873 kB	inapplicable
Microsoft MPEG-4 v2	44,251 kB	147,367 kB

Table 5.1: Comparison of file sizes resulting when representing electronic chalkboard content alone and with overlaid instructor in different formats. Please refer to the text for a description of the experiment.

containing an overlaid instructor video. The resolution of the board is 1024×768, and the total length of the lecture is 1 hour, 37 minutes. The lecture was converted from E-Chalk's regular event format (see Appendix C) to MPEG-4 vector format (BIFS), to Windows Media Video using a frame-by-frame screen-capture codec, and to pure MPEG-4 video format. All videos were converted using 10 frames per second. In our experience, this frame rate mostly provides an appropriate tradeoff between bandwidth requirements and video quality when playing back electronic chalkboard lectures using video, because board drawings usually do not appear at a very high speed. Moreover, recent studies have confirmed, that animations appear smooth even with lower frame rates when they are supported by an audio track (see for example [Mastoropoulou et al., 2005]). The video files were converted with the lowest possible bandwidth that did not result in any visible artifacts during replay. Of course, conclusions drawn from the presented figures take into account that "no visible artifacts" is sometimes a subjective measure. The presented figures provide an indication for the amount of data that is generated when encoding the same content in different formats, in terms of orders of magnitudes.

Table 5.1 shows the results of the two runs. When only the chalkboard is encoded it can be seen that, although the original E-Chalk board event file is encoded in user-readable ASCII format without any entropy coding, it is still a very efficient format. Giving up user readability and applying entropy encoding would bring the storage requirements of a 97-minute lecture down to less than half a megabyte (for example, a zipped version of the file resulted in a file size of 358 kB). Representing the board strokes using MPEG-4 BIFS (as described in Section 5.4) results in storage demands of about the same order of magnitude. The output of screen capture codecs is only a bit larger for this lecture. In general, the quality does not match the other formats described here. Especially images have many artifacts, and animations are sometimes messed up when the screen is captured in the middle of a redraw. Using state-of-the-art movie codecs results in file sizes that are several orders of magnitude higher. It is evident that for big parts of the video file the perceptually relevant information between two frames actually only consists of a change in several pixels. However, even if the representation is sampled down to ten frames per second, the techniques used by traditional video encoders do not yield acceptable compression results.

In the second run of the experiment, the same lecture was encoded again but with a semi-transparent, overlaid instructor. As described in Chapter 9, the instructor is recorded using a resolution of 640×480 and then extracted. Both, the E-Chalk Java client as well as MPEG-4 players that are able to play back BIFS content, receive the segmented video in the original resolution and scale

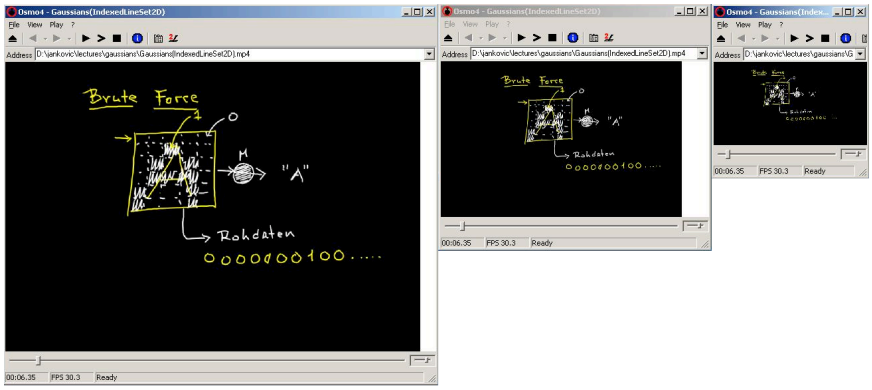

Figure 5.12: MPEG-4 players generally allow scaling replay content to any size. The example shows the Osmo-Player scaled to three different sizes.

it up to fit the board replay. For replay using Windows Media ScreenCapture codec or regular MPEG-4 movie players, the segmented instructor video has to be scaled up during conversion. As a result, more data has to be stored and transmitted for replay (because the codecs are not scaling-invariant). As can be seen from the figures in Table 5.1, adding an overlaid video of the instructor gives different results because the instructor video takes up the biggest part of the data. The E-Chalk format takes up most storage space because it is a very simple format (described in Chapter 8) that is aimed at computational efficiency. The results of encoding MPEG-4 BIFS plus video depends mainly on the video codec used. For this experiment, Microsoft MPEG-4 v2 was used. The resulting file is smaller than when using the movie codec also for board transmission because the video is encoded in 640×480 and the players scale the video up. The results of the Windows Media ScreenCapture codec were of intolerably low quality because it applies a very strong color quantization. The coding strategies of traditional video encoders, such as the MPEG-4 video profile, offer good tools for compressing data when an instructor is to be transmitted too. The table shows the results without audio track. The audio track adds another 8 to 46 MB to each file (about 10-64 kbit/s), depending on the codec that was used. In the result, each of the shown alternatives can be easily played back using a DSL or cable connection.

5.6 Conclusion

Most distance education projects still use traditional video encoders for both chalkboard and slide contents (compare Chapter 2). Although screen-capture codecs give better compression results for board-only lectures, vector-format storage is still more efficient, does not produce any compression artifacts, and preserves board stroke semantics. The required video players as well as the codecs are not always installed and force listeners to perform downloads and installations. The only reason to convert E-Chalk lectures without overlaid instructor into traditional video formats is enhanced compatibility with a broad range of players, especially on handheld devices, or to distribute lectures on DVD. E-Chalk lectures with overlaid lecturer can be better compressed using

movie codecs. In practice, however, the effect is negligible since for receiving a lecture with overlaid instructor, a DSL or cable connection is required, no matter which format is used. A separated transmission of the three streams still allows to switch off individual streams for connections that do not provide the required data-rates and other features that need lecture semantics, for example dimming the transparency of the lecturer or scrolling the board independently of the replay. MPEG-4 BIFS allows to encode E-Chalk lectures properly without any loss of event semantics. MPEG-4 BIFS players support scaling (see Figure 5.12) and MPEG-4 BIFS editing tools would even allow for post-processing of lectures. In theory, MPEG-4 BIFS would render both the Java-based E-Chalk client and Exymen's E-Chalk support obsolete. In practice, however, there are too few implementations of the standard and they still have too many technical problems. The Osmo-Player by GPAC comes as a source package and has to be compiled prior to first use because of licensing issues, and Envivio's plug-in is a commercial product and not freely available. The IBM toolkit appears promising at a first glance because M4Play is available as a Java Applet that would not require the user to perform any downloads or installations. However, a license fee has to be paid for regular use and distribution of the toolkit or parts of it. Because of the lack of support for random seek, the player is just not sophisticated enough to be an alternative to the E-Chalk client. At the time of writing this, a completely self-developed MPEG-4 encoder, server, and replay-client seems to be the only viable solution in order to use the MPEG-4 standard properly. The main disadvantage of the self-developed E-Chalk client is that it has to be maintained and kept compatible with any future browsers and Java versions. The advantage, on the other hand, is that the underlying formats can be kept simple and the operational requirements for the user can be kept low. The remote viewer can turn off individual streams, and the minimal bandwidth requirements can be fulfilled by analog modems. The Java client does not require any explicit download or installation, and random seek is efficiently supported. When E-Chalk's format changes, a new client can automatically be provided for the remote viewer without notice. However, playing back a lecture with overlaid instructor without dropping frames is still a problem for current home computers.

Thus, replay using traditional video formats makes sense for special applications such as supporting handheld devices. When an instructor is semi-transparently overlaid, video replay provides a workaround to allow higher replay frame rates. The self-developed E-Chalk client is the only way to cope with user demands (compare Section 5.1) for distance teaching and will remain the best option until MPEG-4 becomes elaborate enough to provide a practical alternative.