Interactive Technology & Smart Education (2007) 31-42

Navigational indices and content interlinkage on the fly

Peter Ziewer and Thomas Perst
Institut für Informatik/I2, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
Email: ziewer@in.tum.de, perst@in.tum.de

Lecture recording provides learning material for local and distance education. The TeleTeachingTool uses the very flexible screen recording technique to capture virtually any material displayed during a presentation. With its built-in annotation system teachers can add freehand notes and emphasize important parts. Unlike other screen recorders, our implementation offers slide-based navigation, full text search and annotated scripts, which are obtained by automated post-production. This article presents how automated analysis generates indices for slide-based navigation on the fly and how to achieve live interlinkage of annotations with slides so that annotations disappear when a slide is changed and are made visible again when returning to that slide later during presentation, although screen recorders generally do not provide an association of annotations with slides.

Keywords: Lecture Recoding, Screen Recording, Slide Navigation, Annotations, Automated Post-Processing

1. INTRODUCTION

Lecture/Presentation recording is preserving real live presentations for later playback and provides additional learning material not only for distance learning, which can be created rather cheaply and quickly and therefore is called *lightweight content creation* (Kandzia *et al*., 2004). Lectures and presentations are sequential and so are their video style recordings, the *asynchronous electronic lectures* (Brusilovsky, 2000). Distance students, or local students who have missed a live lecture, may watch the electronic version in full length, but in general pure sequential playback is not sufficient. Students rather want to locate and access certain topics and parts of special interest, which could be, for instance, the beginning of a chapter or a specific definition (Zupancic & Horz, 2002; Schütz, 2003). They possibly replay very interesting or not understood sequences more than once, but may skip other parts.

Most asynchronous electronic lectures support *timeline navigation*, i.e. navigating by specifying a certain point in time within the duration of the recorded session, which is generally provided in the form of a slider. The slider represents the timeline of the lecture and any slider position in-between relates to a position within the duration of the lecture. Hence, by use of the slider the user can skim through the lecture and thereby get a better overview. According to Lauer and Ottmann, such skimming works best if *visible scrolling* is supported, i.e. if the screen is updated in relation to the knob

position while dragging the knob instead of updating it after the knob has been released (Lauer & Ottmann, 2002).

Although timeline navigation is a useful feature, it is often not the best way to navigate. In fact, navigation by time is rather annoying if trying to access the beginning of a certain slide or topic, particularly if accessing takes a couple of seconds, e.g. caused by buffering streaming media. Even if immediate random access is supported, searching is not as comfortable as required. Dragging a slider back and forth until a certain position is found, is rather imprecise and therefore time consuming and annoying. Consider the tedious search of a certain song on a music tape or a passage on a video tape, always winding back and forth, and replaying a short fragment until the desired sequence is located. Besides assuming that a user knows what to search for, such an approach moreover requires not only to identify the target sequence, but also to identify even the unwanted sequences to be able to decide whether to wind forward or to rewind the tape. Thus, knowledge of the recording is a precondition for efficient searching. Accessing a certain song on a CD or a movie chapter on a DVD is much easier by use of a predefined table of contents, which exactly refers to significant access points within the timeline of the media. In order to improve the navigational features of asynchronous electronic lectures, *indices* are required as navigation marks, which address meaningful positions within a recording and thus provide *structured* recordings. Lauer and Ottmann claim that structured recordings can only be provided by the less flexible, but structure preserving approach of recording the *symbolic representation* of the source documents (Lauer & Ottmann, 2002). However, in previous work we introduced slide-based navigation as well as full text search for a lecture recorder, called *TeleTeachingTool* (TTT), which is based on the very flexible *screen recording* technology (Ziewer, 2004). The missing document structures are regained by automated post-processing of the pixel-based recordings. Furthermore, Lauer and Ottmann postulate that annotations must be associated with slides, so that annotations disappear when a slide is changed and are made visible again whenever returning to that slide later during presentation. The TeleTeachingTool can interlink annotations and indices according to the corresponding timestamps, but the indices are not available during the recording.

This article discusses how to achieve live access to annotated slides by adapting the automated indexing, originally designed for post-production only, to be performed on the fly while the presentation is still in progress, and how to interlink annotations not only with indices but also with content (mainly representing slides). At first, we will give an overview of the TeleTeachingTool environment and its features. Afterwards, we explain the automated computation of navigational indices and suggest how to adapt the algorithm to be performed during a live lecture. Furthermore, the article addresses how to replay earlier annotated slides. Finally, the association of slides and content is discussed.

2. RELATED WORK

Presentation recording systems like *Authoring on the Fly,* AOF for short, (AOF, 2006) or *Lecturnity* (imc AG, 2006), store the symbolic representation of the presented documents and thus preserve document structures and textual content, which provides useful and necessary advanced playback features such as slide-based navigation and full text search (Kandzia *et al.*, 2004; Brusilovsky, 2000). Furthermore, these two systems provide built-in annotation features and, due to the symbolic representation, annotations are associated with slides. However, the recording features of such symbolic recorders are commonly limited to slide-based presentations and commonly they support a single or very few document formats only. In contrast, lecture recorders based upon the screen recording technique, like *Camtasia* (Techsmith Corp., 2006), digitally grab the content of the presentation machine's desktop on a pixel basis and therefore are very flexible as they can capture any application for instance the presentation software and a browser, including pointer movements, animations and annotations, i.e. notes and sketches drawn with an electronic pen. However, Lauer and Ottmann criticize this flexible technique for omitting document structures and textual content (Lauer & Ottmann, 2002). Unlike symbolic recorders, screen recorders commonly provide only sequential playback or at most timeline navigation due to the missing structuring.

Manual post-processing is not suitable for a cost-effective *lightweight content production*, because "reliable chunking, synchronization and indexing [...] requires several hours of manual work for one hour of lecturing" (Brusilovsky, 2000). Li and Hopper introduced session recording for Virtual Network Computing (VNC), a remote desktop access environment (Li & Hopper, 1998; Richardson *et al.*, 1998). Li et al. discussed indexing and retrieval for recorded VNC sessions. They suggest storing and querying user input events for retrieval and to

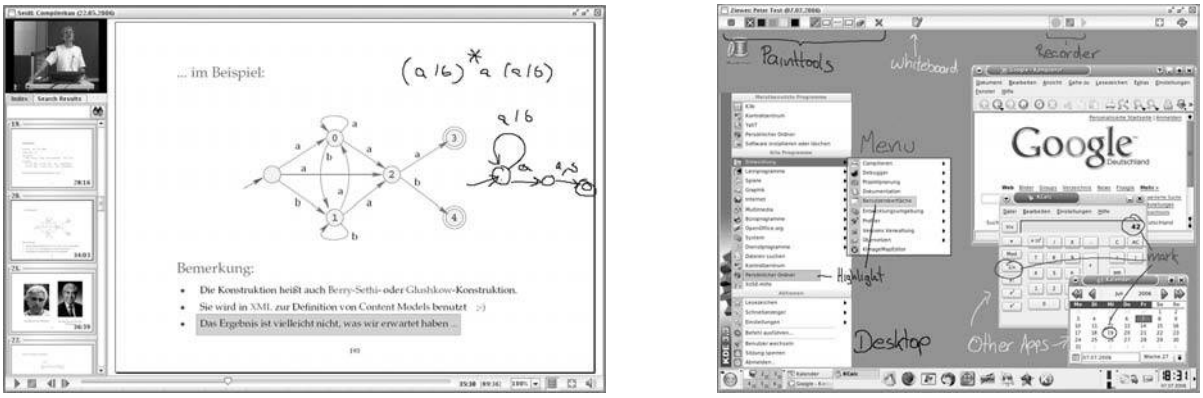

Figure 1 TTT Player with thumbnail overview (left) and TTT Recorder showing an annotated desktop (right)

analyze the screen updates, i.e. which pixels are modified between two subsequent frames, because a big screen update may suggest that an application window has been opened or closed (Li *et al*., 2000).

Considering our intention of recording presentations, big screen updates expose slide changes. Instead of delivering a meaningful set of indices, their system rather offers the possibility of querying slide changes on demand. They provide a method to select all frames changing more than, for example, 40% of the screen. Since it is difficult to decide a meaningful value for the selection, this approach might be useful for advanced users, but is not very intuitive, especially for novice students with less computer experience.

3. THE TELETEACHING TOOL

The *TeleTeachingTool* (TTT) was initially developed and implemented in 2001 as a cross-platform *lecture recording* and *broadcasting* environment, which offers flexible screen recording enhanced with audio and video streams. The TTT is implemented in Java and freely available to the public. It supports various operating systems and allows the parallel use of arbitrary applications, including the teacher's choice of presentation software, animations and browsers. The recorder can be seamlessly integrated into an existing teaching environment in a *transparent* way without influencing the teacher (Ziewer & Seidl, 2002).

Throughout the years, the TTT was extended and enhanced in close relation to our live experiences, gained at the Universität Trier and the Technische Universität München while recording several courses with over 400 lectures in total. Unlike other screen recorders, the TTT offers *slide-based navigation*, which is achieved by automated indexing, and *full text search* due to applying optical character recognition to pixel-based slide images (Ziewer, 2004). A structural overview of a lecture, as requested by (Lauer & Ottmann, 2002), is given in the form of a thumbnail overview (left hand side of the player in Figure 1), which provides a small preview image of each slide index. Clicking on the thumbnail causes an instantaneous replay of the corresponding slide or other indexed content. Additionally, accessing the respectively previous and next slide is supported via buttons (bottom left of the player). Retrieval is initiated by specifying a keyword in the search field (below the video of the teacher) and the results are also presented via thumbnails, but this time the overview only displays the matching indices.

Furthermore, the TTT includes a simple but effective *dynamic annotation system* (controls at the top of the recorder in Figure 1) offering the possibility of drawing freehand notes (Figure 1) and sketches as well as emphasizing presentation content by underlining or highlighting (bottom of the slide in the left picture). All annotations are recorded and are replayed dynamically at the appropriate time during playback. In conjunction with the built-in *whiteboard*, which provides a plain page on demand, the freehand drawing feature replaces the usage of common blackboards or overhead projectors in order to be recorded digitally. Annotations are not limited to slide presentations, but appear on a separate invisible layer on top of the entire recorded desktop. This has the benefit that annotations can be applied to any application (see right picture in Figure 1). Consider, for example, annotating a web page shown in a browser, highlighting some lines in an editor, or annotating your slides with the same tool.

Teachers may publish their slides, but typically these slides do not contain the annotations made during the presentation. Therefore, the TTT offers additional learning material in the form of an

Figure 2 Thumbnail overview of an automatically generated HTML script including annotations

annotated HTML script, which is automatically created by storing and linking slide images. For each index position, a copy of the framebuffer is stored, once full scaled and once scaled down as a thumbnail. The entry point presents a thumbnail overview as shown in Figure 2. Each thumbnail is linked to a page with the corresponding full scaled image and each of the slide images has links to the previous and the next index page as well as back to the overview. Our web server provides access to the recorded lectures and the corresponding HTML scripts. Furthermore, it offers an online full text search and the results are directly linked to the corresponding pages of the automatically generated HTML scripts.

In order to digitally grab and record the teacher's graphical desktop the TTT uses Virtual Network Computing, VNC for short, (Richardson et al., 1998), which is a remote display system that allows a graphical desktop environment to be controlled from anywhere on the internet. The technology underlying the VNC system is a simple protocol for remote access to a graphical user interface, the Remote Framebuffer (RFB) protocol (Richardson, 2005). It works at the framebuffer level and thus is totally independent of operating/windowing systems and applications. A *framebuffer update message* represents a change from one valid framebuffer state to another and contains rectangles of encoded pixel data to be placed at a given x,y position in the client's copy of the framebuffer. The opposite direction is handled by sending the user input events of the VNC client as *key event messages* or *pointer event messages* to the VNC server. Hence, all applications run on the server side, but are controlled via input events by the client.

This pixel-based approach in combination with the availability of several VNC implementations for all common operating systems, makes VNC a suitable basis for a flexible, cross-platform lecture recording system. VNC session recording is achieved by logging timestamped RFB protocol messages for later playback (Li & Hopper, 1998b; Li & Hopper, 1998). The timestamps enable synchronization during replay. The TTT extends the concept of VNC session recording by adding the functionality to record, transmit and replay audio and video streams and also enables live broadcasting of VNC sessions in a scalable fashion (Ziewer & Seidl, 2002). In its original design, the RFB protocol neither provides any message delimiters nor does it support any other possibility to separate messages, except to parse message after message. In the designed context of remote desktop access this is sufficient, but regarding later replay including random access and other navigational features, (Ziewer, 2004) adapted the protocol for TTT to enable direct access to rectangle headers and to facilitate the reading or skipping of complete messages during post-processing and playback. Furthermore, additional message types were introduced to integrate TTT annotations[1] .

4. NAVIGATIONAL INDICES

Minneman identifies four classes of navigational indices (Minneman, 1995):

1. *intentional annotations* to explicitly mark particular points of interest during a lecture;
2. *post-hoc indices* that are created manually during post-production;
3. *side-effect indices* containing activities, e.g. key events, that may provide appropriate indices because they are automatically timestamped and logged; and

4. *derived indices* produced by automated analyses of recorded data.

Later these classes were applied to the context of VNC session recording (Li *et al*., 2000). They store timestamped framebuffer update messages as well as user event messages. Any message provides a *potential* indexing mark. They suggest intentionally indexing headlines. As the teacher must manually highlight each headline, this approach influences the live presentation and therefore is not very commendable. They state that the side-effect indices caused by pointer button events are hardly meaningful. Furthermore, each pointer movement generates many pointer position events. We discourage logging key events because the benefit for retrieval is rather limited, but logging any entered password is safety critical. Li *et al*. suggest manual insertion of textual notes as post-hoc indices (Li *et al*., 2000), which unfortunately is very time consuming. Regarding the *lightweight* and *transparent* lecture recording approach, the generation and classification of indices should be automated as far as possible. Hence, only side-effect and derived indices provide an appropriate way of structuring electronic lectures. Offering derived indices by on *demand querying* (Li *et al*., 2000), i.e. by entering certain parameters, is not very intuitive, especially for novice students with less computer practice. A structured overview should rather offer a predetermined set of meaningful indices. Hence, we need to *automatically* derive and classify indices, and present them in the form of an intuitive user interface in order to provide meaningful and easy to use asynchronous electronic lectures. Regarding the classification, it is better to find fewer but reliably meaningful indices instead of presenting all the meaningful indices along with many falsely identified indices.

4.1 Slide Detection

Navigation *by slide* is a key feature each playback engine should offer (Lauer & Ottmann, 2002; Brusilovsky, 2000). Ziewer introduced automated detection of slide images for VNC-based screen recorders (Ziewer, 2004). A slide, or more precisely the representation of a slide, is mainly an image shown to an audience. Screen recorders store these slide images as pixel values. Switching slides during a presentation commonly results in a modification of large areas of the screen. In the VNC context any screen change leads to framebuffer update messages. Therefore the timestamp of any large framebuffer update is a potential index for a slide change. An efficient implementation requires fast access to the rectangle coordinates and dimensions in order to compute the affected pixels. Therefore, the TTT transforms the RFB messages before recording them and thus enables fast access to the rectangle headers. The number of pixels is accumulated for each given timestamp, where a slight delay of several milliseconds is tolerated. Comparing the number of pixels per timestamp with appropriate thresholds enables a prediction of the presented content. By translating the absolute quantities of pixels to relative values in relation to the maximum, which is given by the dimension of the framebuffer, we can specify thresholds that are independent of the framebuffer resolution.

Figure 3 shows the relative quantity of pixel values per framebuffer update for an exemplary lecture. The recurring values below 5% are caused by periodical update stripes, which are used to compute keyframes. The framebuffer is divided into 24 stripes and thus each stripe covers 1/24=4.1666%. These stripes can be filtered by ignoring any updates up to 4.2%. Any updates that exceed a given slide detection threshold are classified as potential slide indices. Our experiments with several recorded lectures revealed that a *slide detection threshold*, which just slightly exceeds the stripe filter, generates too many indices due to irritations, e.g. caused by pixel-based annotations or delayed framebuffer updates (see Figure 3 min. 20-30). We rather propose an empirically determined slide detection

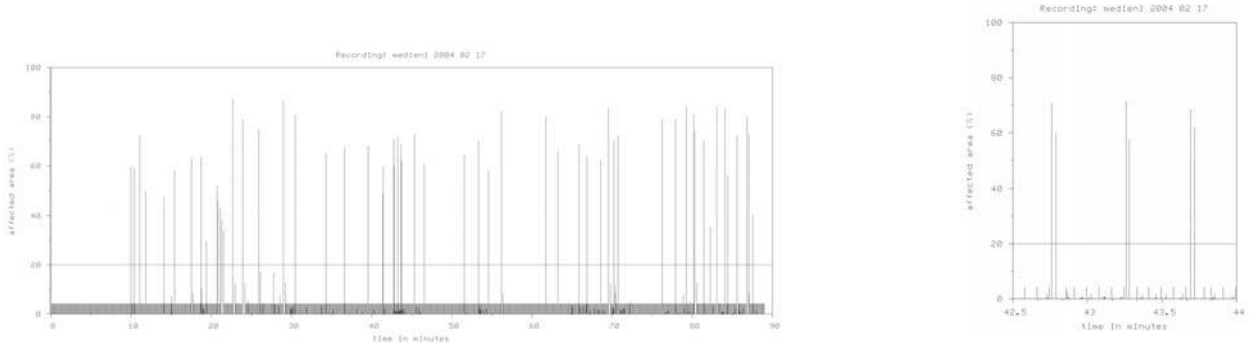

Figure 3 Affected areas of a recorded slide presentation

threshold of 20%, which is large enough to filter irritations, but still recognizes most of the slides.

The zoomed graph on the right hand side in Figure 3 reveals another phenomenon. There are some occurrences of framebuffer updates that evidently exceed 20% and which are followed by another large update within a very short interval of less than 2 seconds. Surveying the recorded session revealed that slides with scanned statistics were shown. However, at first the presentation software showed only the slide with headings, but needed some time to display the scans. Due to the short delay, the framebuffer updates have distinct timestamps and thus would result in different indices although they were induced by the same slide. A VNC server under heavy load may also cause a slightly delayed transmission of updates which belong together. As we are interested in the *final* appearance of each slide, we can drop potential indices that exceed the threshold but are overruled by another index shortly after. A *slide drop threshold* of five seconds is suitable for dropping previous slide indices, because showing a slide for less than five seconds will hardly be meaningful. Furthermore, this error correction eliminates skipped slides, i.e. slides that were skipped by the teacher in order to search a certain slide or skipping back to the previously shown slide after accidentally switching to the next slide. Analyzing several recordings revealed a slide detection hit rate of 95%–100% for slide presentations, which is competitive with the perfect values of symbolic recordings.

4.2 Animation Detection

A flexible, pixel-based recording environment provides not only the possibility to record mainly static slides or more precisely slide images, but also supports the recording of *dynamic content*, such as animations, simulations or executed programming examples, which we will all call animation to simplify matters. Slide overlays or sequences of slides, which may be used to explain certain issues in a step by step fashion are also classified as animation. Unlike static slides, dynamic content does not generate distinct peaks, but rather causes many updates (possibly exceeding the slide detection threshold) as long as the motion lasts. Hence, animations cause sequences of potential indices. However, a meaningful index is the first one, which refers to the beginning of the animation, and possibly the last one, which is less meaningful as an access index but reveals the duration of the animation.

During our "Compiler Construction" lectures we often use a simulator for visualizing the memory management of program execution, in particular the stack, heap and register operations. Setting, copying and deleting values result in animated movements and fading effects of the objects, which represent the corresponding memory cells. Figure 4 shows a screenshot of the simulator and the resulting graph of affected areas for a recorded simulator usage. It evidently exposes the dynamic presentation content between minutes four and six. Even the short pause, which the teacher made to explain the presented simulation, is recognizable by the gap around minute five. After the simulator presentation, the teacher started a slide presentation (Figure 4; around minute seven) and the later peaks are caused by switching to the subsequent slides.

Framebuffer updates that follow each other at a high rate are potentially caused by dynamic content. Examining the gaps between the updates also reveals the end of such a sequence of updates and thus enables multiple animations to be distinguished from each other. An animation starts whenever the delay between two subsequent framebuffer updates falls below a certain threshold and it lasts as long as the threshold is not exceeded. This animation detection threshold must either be lower than five sec-

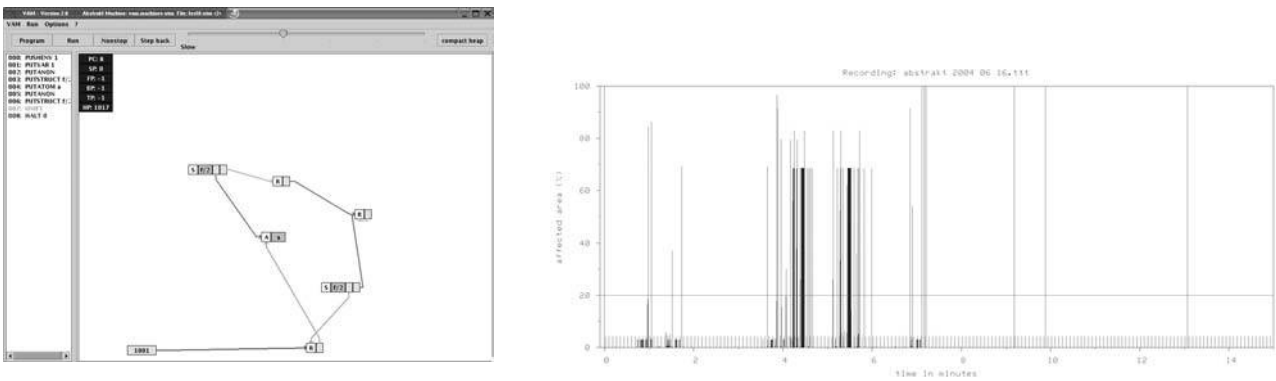

Figure 4 Screenshot of an animated simulator (left)

onds or the updates caused by the periodical update stripes must be filtered, because otherwise the delay between two subsequent updates will never exceed the mark of five seconds and thus results in an algorithm assuming one endless animation for each recording. The filtering is easily achieved by applying a *stripe filter threshold* slightly exceeding a stripe's size of 4.1666%. As a side effect of the stripe filtering, movements of the mouse cursor are also filtered, because moving the pointing device is typically not the kind of dynamic content which is of much interest. Surveying the results that are caused by recording the simulation (Figure 4), yields delays between subsequent large updates of less than a second as well as others of several seconds. If the animation consists of permanently changing areas, the delay between two updates is very short. However, an animation may contain short pauses as is the case for the step by step execution of our simulator. If the applied animation detection threshold is too low, too many indices are generated, a high value, on the other hand, may assume two subsequent slides to be an animation instead. Hence, the threshold must be lower than the presumed minimum distance between two slides. For the given example, a threshold of 10 seconds is suitable to classify two animations, one lasting from 3:51-4:40 min and the other from 5:07-5:50 min. Examinations of several recordings revealed that reducing the threshold further to 5 seconds results in too many indices and applying an animation detection threshold of 15 seconds already classifies some meaningful slide indices as a single animation.

4.3 Slides vs. Animations

Animation detection cannot be treated independently of *slide detection*. Applying the animation detection, for example, after the slide detection, may cause the animation detection to fail because the slide error correction may drop messages that are possibly useful for the animation detection. Furthermore, both detection methods have a contrary behavior regarding how they classify potential indices of a sequence as important. While the slide algorithm drops all indices except the last one, the animation detection keeps only the first index. To avoid these classification conflicts, we have to carefully distinguish animations from slides. Therefore, we introduce the *sequence duration threshold* of 15 seconds, which defines the minimum time span that a sequence of framebuffer updates has to exceed to be classified as animation. Shorter sequences are classified as skipped or split slides.

4.4 Visual Representation of Indices

Once the indices have been computed, the TTT represents them in the form of small preview images – the thumbnail overview – and an annotated HTML script. For each index, we store a screenshot according to the corresponding timestamp. While a scaled down copy is used as thumbnail, a full scaled copy can be used to generate a script directly from the recorded session without any access to the presented source materials such as slides. Since an index refers to a single timestamp but also represents the part of the recorded session from that particular timestamp up to the next index, the preview image should give a representative overview of that particular lecture period. Therefore, annotations made during that lecture period should be included in the thumbnail, because they focus on the key topics of the presented content, or even introduce new topics, e.g. by drawing sketches. As TTT annotations are stored as timestamped messages, we therefore collect all annotations appearing between the current and the following index or a remove-all-annotations event, whatever appears first, and apply them to the corresponding screenshot.

5. ON THE FLY ANALYSES

In order to interlink annotations with slides during a live lecture, we must adapt the detection algorithms to be performed on the fly while the presentation is still in progress. A suitable index computation algorithm must be able to determine meaningful indices by processing a stream of sequential messages in a realtime fashion.

Essentially, the automated post-processing consists of three phases:

1. Derive potential indices
2. Classify indices
3. Compute thumbnails / screenshots

The *first phase*, the computation of the sizes of the framebuffer updates in order to derive *potential* indices, can be transferred to online processing straightforwardly. As each message must be parsed by the TTT for recording and transmission purposes anyway, each rectangle header is read and the affected area can be calculated. Combining updates of almost identical timestamps – as is done during post-processing – causes a short delay of several hundred milliseconds only.

In the *second phase* the identified potential indices are classified to determine a meaningful selection.

The offline classification algorithm analyzes sequences of indices following each other at short delays. However, if messages are received from a stream instead of being available at the start, it is not always evident when a sequence terminates. The classification depends on the applied thresholds. We have suggested an animation detection threshold of 10 seconds, which is combined with a sequence duration threshold of 15 seconds. A potential index exceeding a given slide detection threshold can be classified at least after the maximum duration of a sequence that is still classified as slide skipping - all longer sequences must be classified as animations. Hence, a classification can be achieved after 15 seconds or earlier. A slightly delayed on the fly index computation is acceptable, because immediate usage of newly created back references is not very reasonable. They either refer to the currently displayed slide or to one which was shown recently for a very short time span only and hence cannot be very important and will contain only a few, if any, annotations.

A sequence that ends within the sequence duration threshold is assumed to be caused by skipped slides, opening a new application window or delayed messages due to heavy server or network load and thus the last index is rated to be valuable. A sequence is classified to be an animation if it exceeds the threshold. Animations can last longer, but only the start point and thus the first index is classified as important. As soon as the algorithm assumes that an animation is in progress, it can drop any further potential indices exceeding the slide detection threshold until the end of the animation is detected by a decreasing rate of framebuffer updates. Keeping in mind that we want to allow teachers to access annotated slides during the presentation, it is doubtful whether past animations should be accessible at all. If not, animations are simply ignored. They could also be treated in the same way as the short sequences, meaning that the last index in the sequence will be classified as suitable and thus the final framebuffer at the end of an animation with all annotations would be accessible. However, as it is unsolved how to guess the content and intention of an animation, it is not possible to determine if a suitable snapshot should be achieved at the entry point, the end or any position inbetween.

A thumbnail overview is a meaningful representation of indices. It can be updated dynamically whenever a new index is detected or an existing one should be replaced due to a higher classified index. As a perpetually changing index overview may confuse the teacher, replacing should be reduced to a minimum. This is achieved by delayed updating, which perfectly fits with the delayed index detection and classification suggested above. However, the teacher might expect an instant feedback whenever showing a new slide. Therefore a new thumbnail is added immediately whenever the teacher switches to the next slide, but any potential indices appearing shortly afterwards are not displayed until fully classified. This gives an immediate feedback during an ordinary presentation with an adequate amount of time between slides, but does not confuse presenters due to bustling activity in the thumbnail index during animations or while skipping slides. Similarly, each created annotation is added to the current thumbnail with a short delay, because there is no necessity to display them immediately as they are also visible in the main window and the teacher is obviously still occupied with annotating the current slide. As the intention is to access previously made annotations, it is advisable not only to index slide changes, but also to make use of the side-effect indices that are caused by removing all annotations. As result each slide can have several sets of annotations, which refer to different remarks of the teacher and can be accessed individually if represented by one thumbnail per set.

The *third phase* of the post-processing algorithm computes thumbnails and screenshots by fast replay of all update messages and copying the framebuffer's content for each timestamp that represents an index. The online algorithm must store screenshots during index computation, because the framebuffer is modified by every subsequent update and reclaiming overwritten pixel values demands the session to be in memory and the usage of a second framebuffer, which is inefficient. Instead, we store a scaled screenshot for each potential index and delete it if the index is rejected afterwards. In order to avoid performance problems caused by storing many screenshots within a few seconds, it is advisable not to store a screenshot immediately after deriving a potential index, but to wait until the next framebuffer message arrives. This offers the possibility of observing the header of the next update, which may reveal that the new message should be included in the screenshot due to an identical timestamp, or alternatively may result in the generation of a more suitable index to replace the current one. However, each update can contain several rectangles. Since the RFB protocol does not enable access to rectangle headers without parsing all preceding rectangles, either only the first rectangle can be observed or rectangles must be parsed and buffered but not immediately displayed. Screenshot generation is reduced further whenever the detection

algorithm has classified the currently read sequence as animation and thus all potential indices can be ignored until the end of the sequence is determined.

5.1 Live Replay

During offline playback a recorded presentation is replayed *dynamically* in the same way as it was presented in the lecture hall including the teacher's verbal narration. The narration is obviously not needed if accessing a previous index during a live lecture. Dynamic replay of recorded application usage may be meaningful to show the behavior of an application again. However, in most cases it is likely easier and less confusing to rerun the application once more instead of replaying the recorded version, because replaying does not allow interaction with the recorded applications and the index might not refer exactly to the position the teacher had in mind. This is also the case for animations. Furthermore, if accessing an annotated slide, teachers expect annotations to be displayed instantaneously rather than to appear after a while. As TTT annotations are handled on a separate layer, they can be reapplied in the same way as if they were created anew. Since there is no difference between replayed and newly created annotations, the teacher can mix and edit all annotations as the need arises.

Dynamic replay demands to keep the whole recorded session in memory, because reading from a file while still recording to it is error-prone and also the replay itself must be recorded again. In most cases the much easier approach of replaying *static* screenshots is sufficient. Commonly, the predominant part of each lecture consists of a slide presentation and other applications or animations are only shown on demand. Regarding the results of the classification algorithm allows annotated screenshots to be shown if indices are classified as slides, and the dynamic replay of animations or other content elsewise.

6. INTERLINKAGE OF ANNOTATIONS AND SLIDES

TTT annotations are not bound to the presentation software but are applied to the desktop as a whole and hence any application can be annotated. Recall that Lauer and Ottmann postulate annotations should be associated with slides so that annotations disappear when a slide is changed and made visible again when returning to that slide later during the presentation (Lauer & Ottmann, 2002). The first aspect is solved within the TTT by applying *automated removal of annotations* triggered by the keys commonly used to switch slides such as the arrow keys or the page up/down keys. Furthermore, annotations can be linked to indices according to their timestamps. An interlinkage to indices or any other timestamps can be achieved by aggregating all annotations in the period between two subsequent indices as is done to gain the annotations for the visual representation of indices, i.e. thumbnails or script pages. On the fly interlinkage of annotations with already computed indices is achieved by buffering annotation events. Accessing a slide via the thumbnail overview redisplays previously made annotations. However, this is only a *loose* interlinkage, because indices are only referencing slide changes without any knowledge of slide content. A slide shown twice during the presentation causes two independent slide indices. A real association between annotations and slides or any other content would even enable corresponding annotations to be recalled whenever skipping back to previously annotated slides within the presentation software instead of navigating by use of the thumbnail overview.

6.1 Content Interlinkage

In order to achieve *real* interlinkage it is necessary to determine if the currently displayed framebuffer matches any previously shown content. Comparing the current framebuffer with all previous states is not applicable as every update message modifies the framebuffer content and a session of 90 minutes comprises of several thousand updates. However, only grave modifications are important such as switching to another slide or opening a new application. But those are identified by the indexing algorithm and typically limited to several dozen occurrences. Therefore framebuffer comparisons are only needed whenever a received update message is identified as a potential index and the number of comparison partners is limited to the already identified indices. Detecting exact matches using checksums, e.g. CRC32, is a relatively easy task. Different checksums correspond to different framebuffers and, assuming a suitable checksum algorithm, matching checksums should point to equal content at high probability.

Unfortunately, such a comparison will be problematic unless contents match perfectly, which is not necessarily the case. Sources of inaccuracy can be a clock or a performance monitor, which are displayed within an application or the task bar, as well

as animated banners of web pages. Also the frequently changing pointer position is a disruptive factor if part of the framebuffer and not treated separately by VNC's cursor encodings. TTT's own annotations are stored on a separate layer, but annotations generated by any presentation software influence the framebuffer as well. Therefore only a high degree of covering instead of a perfect match should be used as the comparison factor. Examinations of the computer science course "Informatik III" from the winter term 2005/06 by Prof. Schlichter (25 recordings of approx. 90 min.), revealed a content matching threshold of 1.1% differing pixels as suitable to determine the identity of slides. Applying the same threshold to recordings of the courses "Compilerbau" and "Abstrakte Maschinen" both from the summer term 2006 given by Prof. Seidl showed less perfect matches due to the heavy usage of slide overlays during the presentations. Such overlays are very similar as they partly contain the same content, but nevertheless should be distinguished. Lowering the content matching threshold to a value below 0.2% eliminated the problem. Surveying several other recordings confirmed the lower threshold to be suitable for most lectures. However the detection rate for the lectures of Prof. Schlichter is remarkably better when the higher value is applied, which is caused by using a web browser showing HTML-based slides instead of dedicated presentation software. Slide navigation is done via links and followed links are displayed in another color, which results in a higher number of differing pixel values if showing the same slide twice, once before links have been clicked and then afterwards when returning to that slide. Until further research exposes an adaptive threshold computation, a preset suitable for most cases (0.2%) but adjustable for special occurrences is practicable. As specifying a perfect threshold is not necessary for the algorithm, a view additional steps between 0.2% and 1.2% are sufficient and typically a threshold stays valid for a certain presentation style and thus has to be designated only once per teacher or lecture series. Lowering the color depth before performing comparisons or regarding almost matching pixel values, i.e. pixels of similar colors, can also reduce irritations.

A pixel by pixel comparison of several framebuffer states, i.e. screenshots, is not very efficient due to the heavy memory usage and the high number of comparison operations required. However, the number of *effective* pixel values is very limited. Analyzing the screenshots of our automatically generated scripts showed that for most slides with a single colored background approximately 90-95% of the pixel values are set to the same color (Figure 5 to the left). Even very complex slides rarely contain less than 40% of background colored pixels (right hand side of the figure). This leads to very high compression rates even for simple and therefore fast compression schemes such as run-length-encoding. Furthermore, we suggest a comparison algorithm that rejects unequal slides as fast as possible, preferably by a single value comparison, because the vast majority of comparison partners represent unequal slides. Examination of several dozen recordings confirmed the number of background pixels to be a suitable criterion. Slides cannot match each other if the difference in background pixels exceeds the previously mentioned content matching threshold of 0.2% or 1.1%, because differing background pixels are a subset of all differing pixels.

While constructing a color histogram that is used to identify the background color and to count pixels, each pixel must be accessed once, but that is also the case for any other comparison algorithm. Since the comparison of the number of background pixels rejects most of the not matching indices at high probability, the complete comparison of all pixel values is typically carried out only if the number of background pixels almost matches. Therefore the main case of comparing non-matching framebuffer contents can be achieved in a time of O(framebuffer width * height) to create the histogram plus a negligible number of indices single value comparisons instead of O(number of indices * width * height) pixel comparisons. Note that quick rejection fails if color cycling or background images are used instead of a mainly solid background, but their usage for VNC environments is discouraged

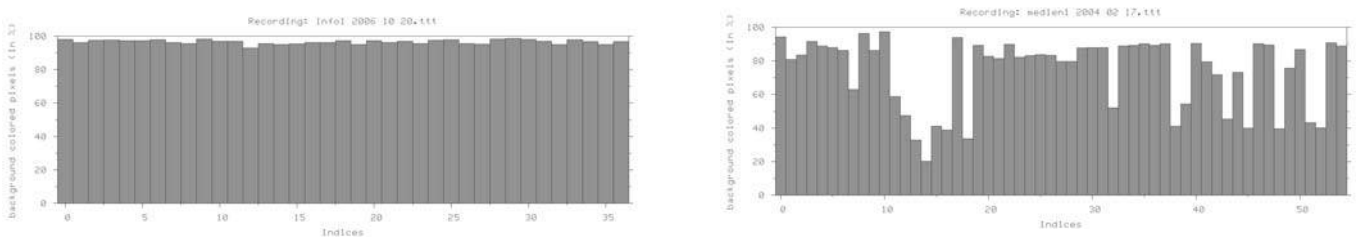

Figure 5 Background pixels of presentations showing mainly text (left) or very complex slides (right)

anyway, because they result in rather bad compression ratios and thus high bandwidth usage. In order to circumvent this general VNC restriction appropriate encoding schemes must be developed for the VNC environment first.

7. FUTURE WORK

Through the examination of the background color of recorded lectures we have detected that the color histogram provides information about the framebuffer content (Figure 5). For instance, if the most frequently used color – typically the background color – covers more than 90% of the pixels, a text-only slide can be assumed, but values of approximately 55-85% lead to more complex slides. A desktop with icons and windows results in a coverage of 30-50% in the most frequently used color and if no color covers more than 5% of the pixel values, a fullscreen video or a high colored picture, e.g. a photo, may have been presented. Surveying the second most frequently used color of complex slides reveals that a value of more than 10% indicates a table or diagram, but lower values most probably point to a slide containing a high colored picture. However, more research is needed to achieve suitable thresholds for such content prediction and to integrate appropriate search and navigational features in a reasonable way.

We suggested *quick rejection* to reduce the number of complete screenshot comparisons needed for content interlinkage. Nevertheless, entire framebuffers must be compared sometimes and thus must be available in memory. Another approach could be to compute and compare similarity hashes for screenshots, but a suitable hash function needs to be ascertained first. Furthermore, content interlinkage would benefit from the development of an *adaptive* threshold computation. Additionally, analysis of dynamic content should be improved.

The HTML script is a useful representation of a slide presentation for our web archive of recorded lectures, but the automated script generation should be extended to produce printer friendly documents, such as PDF, as well. This can be achieved by generating pages including a single screenshot for each index.

8. CONCLUSION

Screen recording offers a flexible technique for lecture recording as it allows virtually any material displayed during a presentation to be captured. *Automated indexing* compensates for many drawbacks caused by the missing structure or symbolic representation of content. Our TeleTeachingTool is a VNC-based screen recorder, which offers automated slide detection to generate useful navigational indices. If recording slide presentations, the automated slide detection for pixel-based recordings reveals slide indices that are almost as good as those of symbolic recordings. Thus, we have eliminated one main drawback of the screen recording approach.

Until now, association of annotations with slides was only feasible by use of symbolic representation. However, we have transferred the indexing algorithm to on the fly usage in order to interlink annotations with indices during a live presentation. Moreover, an interlinkage with slide images and thus presentation content is achieved by comparing screenshots, which represent indices. Therefore, we suggested ideas to perform an efficient comparison of framebuffer contents.

The easy to use navigation via a graphical overview of slide indices, which was previously available for later playback only, can also be adapted for online usage while the presentation is in progress. The thumbnails that represent indices and the annotations are updated as required by the index detection, but with a short delay, which is sufficient to avoid irritations resulting from a constantly changing thumbnail overview. As we assume that full text search is rarely used during live presentation, and because optical character recognition is a complex task, this feature remains for post-production and playback usage.

REFERENCES

AOF (2006) Authoring on the Fly product site. Available at http://ad.informatik.uni-freiburg.de/aof/index.html (Retrieved November 2006).

Brusilovsky, P. (2000) Web lectures: Electronic presentations in Web-based instruction. *Syllabus* 13(5), pp. 18–23.

imc AG (2006) Lecturnity product site. Available at http://www.lecturnity.de/ (Retrieved November 2006).

Kandzia, P.-T., Kraus, G. and Ottmann, T. (2004) Der Universitäre Lehrverbund Informatik - eine Bilanz. *GI Softwaretechnik-Trends* 24(1), pp. 54–61.

Lauer, T. and Ottmann, T. (2002) Means and Methods in Automatic Courseware Production: Experience and Technical Challenges. *Proceedings of the World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education (E-Learn 2002)*, number 1, Montréal, Canada. AACE Press, pp. 553–560.

Li, S. F. and Hopper, A. (1998) A Framework to Integrate Synchronous and Asynchronous Collaboration. *Proceedings of the 7th IEEE Workshops on Enablings Technologies: Infrastructure for Collaborative Enterprises*

(WETICE), Stanford, CA, IEEE Computer Society Press, pp. 96–103.

Li, S. F. and Hopper, A. (1998b) What You See Is What I Saw: Applications of Stateless Client Systems in Asynchronous CSCW. *Proceedings of the Fourth Joint Conference on Information Sciences (JCIS'98)*, volume 3, Research Triangle Park, North Carolina, pp. 10–15.

Li, S. F., Spiteri, M. D., Bates, J. and Hopper, A. (2000) Capturing and Indexing Computer-based Activities With Virtual Network Computing. *Proceedings of the 2000 ACM Symposium on Applied Computing,* volume 2, Como, Italy, pp. 601–603.

Minneman, S. L., Harrison, S. R., Janssen, B., Kurtenbach, G., Moran, T. P., Smith, I. E. and van Melle , W. (1995) A Confederation of Tools for Capturing and Accessing Collaborative Activity}. *Proceedings of the The Third ACM International Multimedia Conference and Exhibition (ACM MULTIMEDIA '95)*, San Francisco, CA, ACM Press, pp. 523–534.

Richardson, T. (2005) The RFB Protocol. Available at http://realvnc.com/docs/rfbproto.pdf (Retrieved November 2006). RealVNC Ltd.

Richardson, T., Stafford-Fraser, Q., Wood, K. R. and Hopper, A. (1998) Virtual Network Computing. *IEEE Internet Computing,* 2(1), pp. 33–38.

Schütz, F. (2003) Producing eLearning materials on the fly – only a great dream? *Proceedings of the Second International Conference on Multimedia and ICTs in Education (m-ICTE 2003)*, Badajoz, Spain.

TechSmith Corporation (2006) Camtasia Studio product site. Available at http://www.techsmith.com/camtasia.asp (Retrieved November 2006).

Ziewer, P. (2004) Navigational Indices and Full-Text Search by Automated Analyses of Screen Recorded Data. *Proceedings of the World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education (E-Learn 2004)*, Washington, D.C., AACE Press, pp. 3055–3062.

Ziewer, P. and Seidl, H. (2002) *Transparent Teleteaching (ASCILITE)*, volume 2, Auckland, New Zealand, UNITEC Institute of Technology, pp. 749–758.

Zupancic, B. and Horz, H. (2002) Lecture recording and its use in a traditional university course. *Proceedings of the 7th annual conference on Innovation and Technology in Computer Science Education (ITiCSE '02)*, Aarhus, Denmark, ACM Press, pp. 24–28.

NOTES

1. Further information, the TTT software and our lecture archive are available at http://ttt.uni-trier.de

Peter Ziewer got his diploma in computer science in 2001 at the Universität Trier. In his diploma thesis he provided an environment for visualizing of abstract machines, the VAM system. Afterwards he worked as an associate researcher at University of Trier and was engaged within the Project ULI (Universitärer Lehrverbund Informatik) whose aim was a (partial) virtualizing of the computer science studies – mainly by recording live lectures. Currently he has a research position at the Technische Universität München. He is the developer of the TeleTeachingTool, an environment for recording, broadcasting and replaying lectures and presentations.

Thomas Perst got his diploma in mathematical computer science in 2001. In his thesis he provided a macro system for XML. As an associate researcher at the Universität Trier he gave undergraduate courses on compiler construction and document processing. Currently he has a research position at the Technische Universität München. His research is concerned with statically guaranteeing that XML transformations are correct w.r.t. their input and output type.