Fakultät für Informatik
der Technischen Universität München

Flexible and Automated Production of Full-Fledged Electronic Lectures

Peter Ziewer

Institut für Informatik
Lehrstuhl Informatik II

Flexible and Automated Production of Full-Fledged Electronic Lectures

Peter Ziewer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Arndt Bode
Prüfer der Dissertation:
1. Univ.-Prof. Dr. Helmut Seidl
2. Univ.-Prof. Dr. Johann Schlichter

Die Dissertation wurde am 30. November 2006 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 24. Mai 2007 angenommen.

Abstract

This thesis describes the *automated production* of *multimedia-based learning materials*. Recording of real live lectures enables a *lightweight* and *cost-effective* way of creating *electronic lectures*. The *flexible screen grabbing technology* can capture virtually any material presented during a lecture and furthermore can be *integrated* seamlessly into an existing teaching environment in a *transparent* manner, so that the teacher is not aware of the recording process.

Throughout this thesis the design and development of a *flexible* and *easy-to-use lecturing environment* on the basis of *Virtual Network Computing* (VNC) is explained. The VNC infrastructure and protocol is adapted to build up an environment that offers *scalable transmission of live lectures* and is capable of supplying a large number of distance students in parallel. Furthermore, the suggested system provides comfortable *lecture recording* in order to produce *electronic lectures* for *asynchronous replay* at any time later. These electronic lectures preserve the *verbal narration* of the teacher and *any material* or applications presented during a lecture including additional *annotations* (e.g. adding freehand drawings by use of an electronic pen).

Unlike other *pixel-based recording environments*, the suggested system offers *slide-based navigation* and *full text search*. This is achieved by *automated post-processing* which generates appropriate *indexing structures* and *search bases* by analyzing the recorded lectures. Several analysing concepts are introduced and compared by evaluating recorded lectures of different presentation styles.

Furthermore, an implementation of the suggested concepts and ideas – the *TeleTeachingTool* – is presented and *usage scenarios* are given in order to approve the *usability* and the *advantage* of the system.

Acknowledgment

I would like to thank

Thomas Perst and **Pete Bankhead**

for their patient efforts in proofreading and
their helpful suggestions and comments.

Contents

1

Introduction

In the last decade the availability of powerful multimedia enhanced computer systems and broadband network connections has increased dramatically, having a large influence on almost all areas of human culture, not only the scientific and working life but also the social behavior and everyday life. Traditional universities reflect this trend and the possibilities of new technologies by widening their well known campus-based teaching and learning scenarios to embrace new media technologies. In recent years various new terms have been introduced, for instance *e-learning* (or *eLearning*), *tele teaching*, *virtual universities*, *distance learning*, et cetera, with sometimes overlapping meanings but also addressing very different subtopics. There are various different *e-learning* scenarios, all of which have in common a *computer-based* enhancement of teaching and/or learning areas, mainly in sense of more *flexibility in time* (self-studying) or *space* (circumventing a spatial separation of participants). In detail there can be a vast diversity of intended meanings containing, for example, *video conferencing* as a possibility for synchronous communication over large distances, *internet discussion groups* or *forums* for an asynchronous communication in an open community, *computer-based training* (CBT) or *web-based training* (WBT), i.e. learning by executing special training programs on a computer, or *learning management systems* (LMS), which provide the management and delivery of learning content and resources to students. Often the different subtopics are combined, for instance, a *learning management system* can deliver *computer-based training* modules or provide access to a forum.

Our involvement in the area of *e-learning* started with the project *Universitärer Lehrverbund Informatik* (ULI) [ULI, 2006], a cooperative project of 18 partners located at one Swiss and 10 German universities, which was sponsored by the *Zukunftsinvestitionsprogramm* (Future Investment Program) of the *Bundesministerium für Bildung und Forschung* (BMBF, German federal department for education and research). The project started in early 2001 and ended in 2004. Its aim was a (partial) *virtualisation* of the computer science study for the following two reasons:

- "For a growing number of students, a full-time attendance study is difficult or impossible to assist in due to family or occupation reasons. A partially virtual curriculum with courses that are not dependent on time and place can enable these students to participate in an up to standard Computer Science study.

- For students from the FernUniversität (distance teaching university) Hagen, this cooperation offers the possibility to make use of the other universities' variety of offers and take part in their courses." [ULI, 2006]

During this virtualization process the task of most of the individual project partners was to create learning materials. Generally the different universities did not develop complex CBT/WBTs due to the high "production costs for rich-media content ... [which] range between 50,000 and 100,000 EUR per weekly lecture hour" [Lauer and Ottmann, 2002]. Typically the rather *cost-efficient recording of real live lectures* and presentations was preferred. Such *lecture recording* (also called *presentation recording*) provides versatile *digital learning material* at negligible additional costs and therefore is called *lightweight content production* [Kandzia et al., 2004]. Hence, the *virtualization* was acquired in a stand-alone manner but rather closely connected to existing teaching and learning forms, which were extended by computer-aided elements.

The special challenge in our sub-project of ULI was that at the *Universität Trier* we intended to record our lectures for *asynchronous replay* and additionally wanted to provide a *synchronous transmission* to our sub-project partners at the *Universität des Saarlandes*, Saarbrücken. Furthermore, the materials that are presented in our lectures should not be limited to slide presentations but additional material such as visualizations or programming examples should also be supported. Since we did not find a suitable piece of software fitting these requirements, we started the development of our own solution, called *TeleTeachingTool* (TTT), which is presented in this thesis. We rejected other applications for various reasons: they were either limited to a single operating system or presentation application, did not support recording and transmitting in parallel, did not support live transmission with a short round-trip time (needed for live feedback), did not properly work or were unstable, or supported no high-quality transmission or recording of the presentation content.

The *TeleTeachingTool* originally emerged from various independent applications, which should be combined in one single system. In the beginning we used *video conferencing* techniques and applications, in particular the *Robust Audio Tool* (RAT) [RAT, 2006] and the *Videoconferencing Tool* (VIC) [VIC, 2006], which provided synchronous communication in audio and video, but with rather limited support for transmitting a desktop presentation (for instance by use of a network editor that enabled the simultaneous access to one document for multiple participants). We added *desktop transmission*, which was archived *pixel-based* and thus independent of the presented content, which provides a *high degree of flexibility*. At first, we transmitted *screenshots* of the presented desktop periodically by using system calls. Due to the poor performance and high bandwidth consumption of this approach, we rather soon decided to switch to *virtual network computing* (VNC) [Richardson et al., 1998], which provides remote access to a virtual desktop via a network. Both project partners, i.e. Saarbrücken and Trier, were connected to the same desktop and thus saw the same presentation, and furthermore were connected by use of video conferencing applications. The *TeleTeachingTool* replaced these individual components and thus provided an integrated environment. In fact, the core of the *TeleTeachingTool* is a modified *VNC client*, which is implemented in *Java* and thus offers a *platform independent basis*.

Throughout the years the *TeleTeachingTool* was improved and extended in close relation to our own requirements and experiments as well as in relation to external suggestions and related research publications. In the beginning, we have focused on the teacher's convenience by developing a system which enables recording and transmission of live lectures but without influencing or limiting the teacher in her/his choice of presentation applications and materials. Thus, we postulated as a main design goal that the recording and transmission process should be *transparent* to the teacher. In order to respect the special needs of digital presentation recording, we furthermore added additional *annotation and whiteboard components*, which enable presented elements to be emphasized and thus focus the attention of the audience, and enable notes and sketches to be added on demand. Additionally, our research addressed the *scalability* of lecture transmissions in order to support a high number of simultaneously connected online students.

Recording live lectures typically produces *electronic lectures* of about 90 minutes each. Typically a course consists of about 10–25 lectures resulting in about 15–35 hours of learning material. Hence, offering only *sequential playback* of recorded lectures is not sufficient. In order to offer students the possibility of locating and accessing certain predefined access points (such as chapters or slides) or to search for a particular topic, *electronic lectures* must rather support *navigational* and *retrieval features*. Commonly a *symbolic representation of content*, which preserves document structures (e.g. table of content, slides), textual content (e.g. sequences of ASCII characters) and images (e.g. vector graphics), is needed in order to support, for instance, *full text search* or *slide-based navigation* (i.e. direct access to each slide) [Lauer and Ottmann, 2002]. For *pixel-based recordings* the *navigational* and *retrieval features* are rather limited. On the other hand, a *pixel-based presentation recording approach* offers a much higher degree of *flexibility* as any presentation content can be preserved. Recording environments that preserve the *symbolic representation* of the presented content are very limited in their *flexibility*, because such systems typically need access to the source documents and support only a few document types and often require the use of a certain presentation application. Hence, we typically have a *flexible* recording environment that is rather limited in its *navigational* and *retrieval capabilities* or a system that produces *full-fledged electronic lectures* but restricts the teacher while presenting.

In order to generate *full-fledged electronic lectures* by use of a *flexible recording environment* this thesis states *two main goals*. The first one is the design and development of a *flexible, easy-to-use recording and transmission environment*, which does not restrict teachers in their content production and presentation process, but rather can be seamlessly integrated into existing teaching environments in a *transparent* manner.

The second main goal is the *automated production* of *electronic lectures* by adding *navigational structures* and *retrieval features* to *pixel-based recordings* in order to create *full-fledged electronic lectures* and thus extend the usability of the *flexible recording technique*. In order to keep the benefit of the cost-efficient *lightweight lecture recording approach*, the production of such *full-fledged electronic lectures* must be *automated* as far as possible.

1.1 Outline

After this introduction, we will start in Chapter 2 with naming and describing the *terminology* that is used throughout this work and will describe and compare different *approaches of lecture recording*. Furthermore, we will discuss the *requirements and features* that are necessary to reach our main goals by building a *criteria catalog* that can be used to compare and classify lecture recording systems.

Virtual network computing (VNC) and its *remote framebuffer* (RFB) protocol as the basis of our recording system are described in Chapter 3. Furthermore, the benefits and limitations of VNC in regard to the e-learning aspects are discussed. The next two chapters address how we modified the VNC environment to fit our requirements, but ensuring compability with the original VNC server components so that our system can access a remote VNC desktop. In particular, we will discuss how to achieve a *distance learning environment* by *improving the scalability* in order to support a large number of simultaneously connected students (Chapter 4) and, as VNC is originally only designed to remotely control a desktop via a network connection, how to record live VNC sessions (Chapter 5). While discussing the recording aspects, we will compare different *recording formats* regarding their suitability for recording and later replay.

Chapter 6 addresses *annotations*, i.e. presentation related electronic note taking and focusing the attention of the audience, as a meaningful feature of a computer-aided lecture presentation environment and how annotation features can be integrated into our VNC recording environment.

The following two chapters consider the second main goal of this thesis. Chapter 7 describes how we acquire *navigational indices* by *automated post-processing* of *pixel-based recordings* that were produced by the *flexible screen recording technology*. Different solutions for content analysis and prediction (e.g. *slide detection*) are described, evaluated and compared in order to find well-suited algorithms that enable an *automated structuring of pixel-based electronic lectures* as the basis for appropriate *navigational features*. Furthermore, we will address how to present the navigational structures and how to navigate and thus access the parts of an electronic lecture that relate to certain content. The *retrievability of pixel-based recordings* is addressed in Chapter 8. At first we provide a applicable possibility for creating *search-bases* that are suitable to perform *full text searches* within *screen recorded lectures*. Furthermore, we extend the searchability to perform *cross lecture searches* and address how to present and access the *search results*. Additionally, the handling of meaningful lecture related *metadata* is discussed.

Afterwards, the *TeleTeachingTool* and its *components, features, usage and capabilities* are described in Chapter 9. We will give *usage scenarios* for the recording, transmitting and post-processing of electronic lectures. Finally, a *conclusion* and suggestions for *future work* will be given in Chapter 10.

2

Presentation Recording and Transmission

The scope of this thesis is a subtopic of the vast area of *e-learning*, namely the *recording and transmitting of real live presentations* in order to create *electronic lectures*. Moreover, this work focuses on presentation-like teaching styles, which are common not only in computer science but in many other natural sciences and economics. A lecturer presents some information in the form of slides or by writing it up on a blackboard, accompanied with verbal narration to explain the contents. In other domains such as languages or humanities the requirements and thus the teaching styles can be very different.

This chapter specifies what is considered to be an *electronic lecture* here and also states and explains other terms used throughout this thesis. A discussion about benefits, requirements and restrictions of the transition from *traditional* to *electronic* teaching styles will be followed an explanation and comparison of different approaches to *lecture recording*. The aim is to elaborate a suitable *catalog* of desirable *features* for recording and playback, allowing the comparison and evaluation of lecture recording systems and research approaches within the given context. Finally, an overview of existing systems is given, including our own implementation, the *TeleTeachingTool*, with regard to the ascertained criteria.

2.1 Electronic Lectures

In the sense of Brusilovsky, an *electronic lecture* "preserve[s] a lecture as an element of web-based education replacing real lectures" [Brusilovsky, 2000]. This is a very vague concept, so we have to take a closer look at what we consider here to be an *electronic lecture*. Other terms used within this thesis are stated and explained as well. Furthermore, this section will point out why such lectures are beneficial for both distance and local students, and what requirements must be fulfilled to achieve a *transition from traditional to electronic lectures*.

2.1.1 Terms and Definitions

[Brusilovsky and Miller, 2000] differentiate *hierarchically* and *sequentially structured* learning materials, called ***electronic textbooks*** and ***electronic presentations***, respectively.

An ***electronic textbook*** offers a *hierarchically structured* representation of material, which usually consists of text, augmented with figures. It is commonly provided in the form of HTML[1]. The content is organized similarly to a *printed textbook* with its subdivision into chapters, sections and subsections. The hierarchical structuring implies special ***hierarchical navigation***, which includes links to all subordinate sections, a link to the higher level section, and a link to the beginning of the electronic document. Furthermore, ***sequential navigation*** provides access to the next and previous pages. There may be a linked *table of contents* and an *index* as well.

Brusilovsky and Miller distinguish two subclasses of sequentially structured ***electronic presentations***: ***electronic lectures*** and ***guided tours***. A ***guided tour*** is a sequence of (perhaps previously developed) content (possibly of multiple authors), which is accompanied by a *narration*. The narration is usually provided in the form of text, called ***textual narration***. This type of presentation is modeled after a *guided museum tour*.

An ***electronic lecture*** is "a sequence of slides extended with audio or audio/video narration" [Brusilovsky and Miller, 2000]. In our sense the concept of mainly ***static slides*** should be loosened to include ***dynamic elements***. [Effelsberg and Geyer, 1998] request the possibility of using (and recording) media other than text and still images as crucial components of modern computer-based teaching, because "motion and interaction really make the difference between paperbased teachware and computer-based teachware" [Effelsberg and Geyer, 1998]. ***Dynamic elements*** can be *animations*, *simulations* or any *other applications* used in addition to some presentation software. In other words, talking about electronic lectures we include any material and elements used during modern *multimedia enhanced presentations*. In the following discussion we will use the term ***slide*** not only in the meaning of conventional ***static slides***, but also to refer to ***presentation content*** in general containing any material, media or application presented by a teacher. Moreover, since this thesis is not concerned with guided tours, we use the terms ***electronic lecture*** and ***electronic presentations*** synonymously.

[Brusilovsky, 2000] distinguishes *synchronous* and *asynchronous* (electronic) lectures. ***Synchronous lectures*** provide access to real lecture halls from a distance. Live presentations are transmitted via any network or infrastructure to another location, either as ***unidirectional*** broadcast only or ***bidirectionally***, allowing two-sided communication offering distance students active participation (e.g. asking questions). ***Asynchronous lectures*** are intended to be viewed at any time (later), perhaps by students, who could not attend real lectures, or to rework the presented content. Synchronously presented lectures can also be recorded and thus turned into *asynchronous lectures*.

[1] Hyper Text Markup Language (HTML)

Common digital media types are audio and video streams. In case of electronic lectures ***audio*** preserves the ***verbal narration*** of teachers and, if required, questions or explanations of students as well. If talking about ***video*** we think of a ***live video*** filmed in the lecture hall mainly showing the teacher. We do not intend to film presentation content, because simply videotaping lectures does not lead to acceptable quality of the filmed blackboard or slides [Lauer and Ottmann, 2002]. Besides audio-only and video enhanced electronic lectures, Brusilovsky furthermore classifies ***high-quality video***, which "requires special recording equipment, preferably a studio with a blue screen and several hours of processing time for [...] one hour of lectures" [Brusilovsky, 2000]. The efforts and costs of producing such high-quality video lectures is similar to the production of CBTs/WBTs and "... range between 50,000 and 100,000 EUR per weekly lecture hour" [Lauer and Ottmann, 2002].

This thesis rather focuses on so-called ***lightweight course production*** [Kandzia et al., 2004], which is "to record a quite regular live lecture in order to obtain versatile digital material with negligible additional cost" [Kandzia and Maass, 2001]. Nevertheless such ***low-cost recordings*** can be reworked, edited, enhanced and combined with other material and thus be the basis for enhanced WBTs. Our aim is preserving a live presentation to achieve *valuable* additional learning material, but not at the *cost* of the real lecture. The process of recording is called ***lecture recording*** or ***presentation recording***. Those terms are not only used in to mean "recording to a file for later asynchronous replay", but also to refer to "the act of grabbing a presentation's content (for synchronous transmission and/or for recording to a file)", which can be called ***presentation grabbing*** as well. If only the *synchronous* or *asynchronous* case is meant, we will explicitly note which one or use non-ambiguous terms, such as (the noun) ***recording*** and ***file*** or ***transmission*** and ***broadcast***, respectively. Furthermore, this thesis will mainly concentrate on recording a presentation's content (in the sense of what is displayed on the presentation computer or presented to the live audience via video projector) and not on audio and video (in the sense of a camera filming the lecturer), because audio and video recording stays almost the same for the different presentation recording systems. At best, presentation grabbing/recording should be ***transparent*** [Ziewer and Seidl, 2002], which means that lecturers need not even be aware of the recording process at all. *Transparency* concerns not only the presentation, but also the content creation process. In other words, the recording should not restrict or influence *content creation and presentation*, allowing lecturers to perform their preferred traditional teaching styles, especially including their favorite presentation software. Suggestions to achieve ***transparent recording*** are discussed in Section 2.3.

Textual narration may be sufficient for edited *guided tours* (as mentioned above) or accordingly designed *CBTs/WBTs*. For recording real *live presentations* we rather assume the lecturer's *verbal narration* to be recorded, because "without this, an essential and substantial part of the information will be missing" [Lauer and Ottmann, 2002]. On the other hand Lauer and Ottmann state that for recording traditional presentations "the importance of integrating live video should ... not be overestimated" [Lauer and Ottmann, 2002]. Although the video provides a *visual impression* as well as a feeling of the instructor's presence, it usually conveys comparatively little information when considered alongside the amount of data it

consumes (but this might be different for special purpose presentations, e.g. showing experiments). [Schütz, 2003] declares the video to be useful to build up a personal relationship to a lecturer, but is not looked at in detail after a while. Furthermore he suggests that "because of bandwidth and use of processor time, no really important information ought to be recorded with the video". These conclusions fit with our own experiences.

We also agree with Lauer and Ottmann, who requested ***tightly synchronous replay*** of any recorded media streams and data, because "simply capturing starting times of events (such as launching an application) can result in streams drifting apart". In other words, media streams are not only starting synchronously, but synchronization is also checked and adjusted (if needed) during playback. In order to explain the importance of synchronization, [Effelsberg and Geyer, 1998] suggest an example of a teacher explaining an algorithm where data packets flow over a graph representing a network. The lecturer starts with nothing but the topology of the graph, with nodes and edges. During the explanation of the algorithm, the graph is dynamically annotated with colored arrows representing the packet flow. Hence, the dynamics of the algorithm are explained by annotating the graph and thus, without reproducing the dynamics of the annotations in the recording, essential information is lost.

In order to synchronize replay [Brusilovsky, 2000] requests "video/audio streams [...] to be divided into the smallest meaningful chunks, which usually correspond to [...] a piece of a slide" and "synchronization means that each audio or video chunk has to be associated with a corresponding portion of the slide presentation". Recall that Brusilovsky considers a presentation to be a *sequence of static slides*, but we additionally allow *dynamic content*, which acts more like a stream. We do not recommend *physical* partitioning of media files according to determined chunks as this would cause a huge amount of I/O handling during accessing processes. In fact ***timestamps***, which are non-ambiguous due to the sequential data style, are used to refer to certain points within data streams. Special timestamps corresponding to (the beginning of) interesting parts of an electronic lecture are called ***indices***. Indices and timestamps are essential elements of ***navigation*** within electronic lectures as they, for example, enable access to slides, which is known as ***slide-based navigation***. Another possibility for navigation is ***timeline navigation***, which commonly is available via a *slider* representing the time scale of an electronic lecture from beginning to its end. If any arbitrary time within the time scale can be accessed, this is called ***random access***. In fact suitable *navigational and retrieval features* must be provided by electronic lectures, because it is almost impossible to read and navigate video like lectures [Abowd et al., 1998]. [Brusilovsky, 2000] suggests sequential playback without fine-grained "chunking" (i.e. sub-dividing) to be sufficient for local students, because a teacher is available to solve problems. Nevertheless, we assume navigational and retrieval features to be a crucial factor in order to provide really useful learning material particularly for local students, who have attended the live lectures and use recorded presentations to rework certain topics. These students are mainly not interested in watching recorded presentations as a whole, but rather want to *identify* and *access* only those topics they have not understood. This assumption is confirmed by analysis of students behavior, which showed that navigating through course content is preferred over sequential playback

[Zupancic and Horz, 2002, Schütz, 2003]. ***Manual postproduction*** is not seen as an appropriate solution for indexing as this would conflict with our *lightweight* course production approach. Indexing and retrieval structures rather must be *automatically* retained during the recording process or achieved by ***automated postproduction*** (after the recording process has finished or on demand during playback).

An important feature of electronic learning materials are ***annotations***. A comprehensive overview about various types of annotations is given by [Schütz, 2002], who lists, for instance, *textual notes*, *cross links* (within a document or to other documents), adding *audio or movie clips*, *freehand drawing* (by use of an electronic pen, *emphasizing* of certain elements in order to focus the attention of the audience, et cetera. Such annotation can either be placed by teachers or by students and furthermore can be placed during a live lecture or afterwards. The most common case is that a teacher annotates the presented slides during the lecture by adding aditional comments or sketches and furthermore highlights the most important aspects. [Lienhard and Lauer, 2002, Lienhard and Zupancic, 2003] suggest *annotation layering* in order to handle students' annotations besides those of a teacher.

2.1.2 Usage Scenarios and Benefits

Recall Brusilovsky's definition of an *electronic lecture* as "[preserving] a lecture as an element of web-based education replacing real lectures" [Brusilovsky, 2000]. Certainly, for distance students electronic lectures offer the possibility of being a substitute for traditional lectures, but we do not intend to replace real lectures under any circumstances. There are two approaches to access electronic lectures: the *synchronous* and the *asynchronous* one.

Synchronous electronic lectures are *live transmissions of real lectures* via a network (mainly the internet, but other infrastructures are not excluded) from one lecture hall to another and/or to students' homes. Distance students can participate from anywhere in the world by accessing the corresponding streams. Generally, such transmissions are *unidirectional*, because otherwise a sophisticated *access management* system is required in order to grant *speaking permissions* and avoid the confusion caused by simultaneously speaking participants. The advantage of *synchronous* transmissions (even of unidirectional ones) is that they provide *immediate access*, without any loss of time. All *asynchronous* electronic lectures more or less suffer from a delay caused by postproduction and publication of the recorded lectures. At least the data must be copied to a server and a web page has to be updated to enable downloading. For perfectly automated publication processes such a delay might be a few minutes only, but in the majority of cases it takes some hours or even days from the time of recording until the asynchronous lecture is accessible for students. That can be critical for weekly exercises and assignments.

Another *synchronous scenario* is a dedicated *two-point transmission* between lecture halls as we did during the *ULI project* [ULI, 2006] (see Introduction, page 1) in collaboration with the Universität des Saarlandes. A lecture was given at one of the two locations alternately and transmitted to the other lecture hall. A feedback channel transmitted a live video of the distant audience and was also used to enable *bidirec-*

tional communication (mainly to ask questions). We also extended the scenario by integrating unidirectional distance students (e.g. participating from the homes) as described before. Another variation can be a network of multiple participants similar to *video conferencing scenarios*.

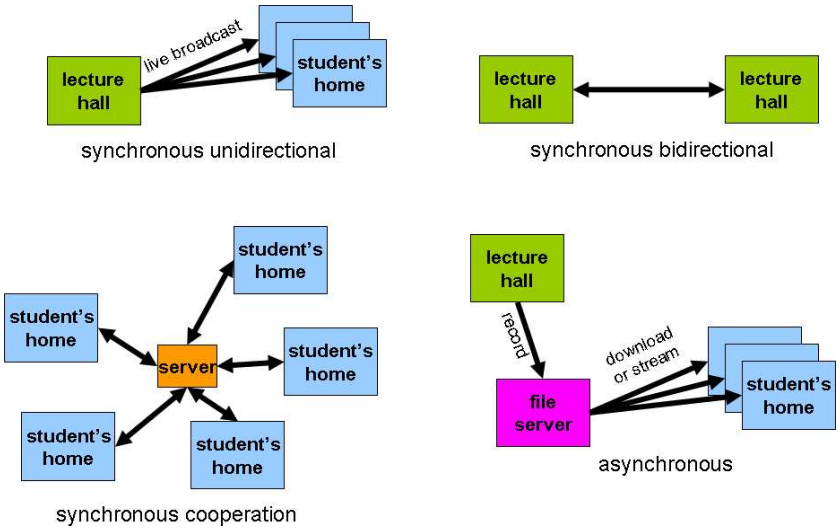


Fig. 2.1. Electronic lecture scenarios

Preserving traditional presentations for asynchronous replay is useful for occasionally missed lectures as well as for students who are not able to participate in time, which can be the case for part-time students in employment or those with parental duties. Asynchronous electronic lectures allow access to courses *anytime* and *anywhere*. Furthermore, appropriate asynchronous electronic lectures provide meaningful *additional learning materials* for local students as well, even if they attend(ed) live lectures. During a live lecture important information can be missed due to the distracting process of note taking. While reworking a lecture, replay of the recording offers the *real wording* of the teacher again. Appropriate *retrieval* and *navigational* features like *full text search* or *slide based navigation*, respectively, can be used to locate and recall special elements and topics in any recorded lecture.

Brusilovsky claims that "many faculty consider electronic lectures the best substitute for classroom lectures claiming that neither textbooks, nor handouts can adequately replace an up-to-date lecture done by a leading researcher or professional" [Brusilovsky, 2000]. Moreover, such electronic lectures preserve the live atmosphere of the classroom and the teaching style of a lecturer for distance students. [LaRose et al., 1997] claim that web lectures are at least as efficient as regular lectures. Electronic lectures can be utilized independent of time and place. Other advantages are "fast and efficient navigation through a lecture, fast retrieval of multimedia content, and course and lecture indexing" [Jackson et al., 2000].

Some typical scenarios for the use of electronic lecture are given in Figure 2.1. However, there are many other and combinations of the shown scenarios are possible,

for instance recording and transmitting, or two (or more) bidirectionally connected lecture halls which also supply students' homes in an unidirectional fashion.

2.1.3 From Traditional to Digital Lectures

In order to achieve valuable electronic lectures by presentation recording, all meaningful elements of a live presentations must be preserved. [Lauer and Ottmann, 2002] declare recording of all relevant media streams occurring in a live session as their ultimate goal of presentation recording. We agree with them that simply videotaping lectures does not lead to acceptable quality of the filmed blackboard or slides (unless high resolution video recordings are practicable in future) and, according to [Effelsberg and Geyer, 1998], that document cameras are inappropriate technology in an all-digital environment. Video taping of computer screens is also discouraged, because the technical limitations of such recording make the screen content very difficult to read during playback [Ishii and Miyake, 1991] and *video compression encodings*, such as MPEG-2 [Mitchell et al., 1996], are not suitable for compressing artificial high-contrast images such as slides due to causing artifacts and displaying blocks instead of a high-quality reproduction of the compressed content [Hogrefe et al., 2003]. Rather, any content and all elements of interest must be captured and digitized in an appropriate quality. Therefore we have to identify such elements and suggest how to translate them into the "*digital world*".

Audio and Video:

Obvious *essential elements* of a presentation are *slides* and *verbal narration* of a teacher. A live video showing the teacher is not mandatory [Lauer and Ottmann, 2002, Schütz, 2003], but nevertheless worth recording, as it can preserve gestures and facial expressions. Digitizing and recording audio and video streams are an everyday business for today's computers, but there should be a well considered decision about which formats to use and which codecs[2] should be applied to achieve compression (as uncompressed recording would lead to several Mbytes or, in the case of video, even Gbytes of data). "The audio has to be of high quality, because it has to transport a huge amount of information" [Schütz, 2003]. Listening to low quality audio over a period of more than a few minutes is straining and exhausting (think of a bad mobile phone connection with flaws and gaps). The most common audio format, *MPEG-1 Audio Layer 3* well known under its abbreviation *MP3*, is capable of compressing a 90 minutes talk to approximately 20 Mbytes in a suitable quality (22050 Hz, 32 kbps, 16bit, mono). Other codecs like *Ogg Vorbis* or *Advanced Audio Coding* (*AAC*) achieve similar results.

The diversity of video codecs is much higher, which is not surprising as video compression is more complex. *MPEG4/Divx* achieves very good compression rates at high quality, but the compression algorithm is time consuming even using powerful computers, which makes this codec unsuitable for online and real time coding. Other

[2] Codec: a COmpression/DECompression algorithm capable of performing encoding and decoding on a digital data stream or signal

codecs, e.g. *H.261* and its successors including the now widespread *H.323* or *H.264*, are optimized for exactly that purpose of real time coding as they are designed for videoconferencing. Streaming media codecs, such as *Real Video* or *Windows Media Video*, even fulfill other design requirements such as supporting different bandwidth settings. Choosing a suitable video format and codec depends on the intended purpose, which can be bidirectional synchronous transmission (requiring short delays), streaming online content on demand or electronic presentations for asynchronous offline usage.

Recall that we do not intend to film the presentation content (e.g. slides), due to insufficient quality. Thus, no special high tech cameras are required to record video as they will always exceed the intended output quality. Technicians operating cameras may disturb the natural flow of a live presentation and, due to the additional costs, are not in accordance with our *low-cost lightweight approach*. In most cases a small fixed camera is satisfactory. Remotely controlled cameras or so-called *pan-tilt devices*, which recognize movements and automatically adjust themselves, can be used as well. Either digital video input via *Firewire* interface (also known as *I.Link* or *IEEE 1394*) or analog input offered by some graphics cards or cheap grabbing devices (e.g. common BT848/BT878 TV cards) is adequate for digitizing live video. Audio recording can be seamlessly integrated by using already existing hardware. Large lecture halls are equipped with at least one microphone and an adequate system of loudspeakers anyway. In such case it is sufficient to connect the recording system to the existing equipment. For smaller presentation rooms, where no voice amplifying is required, obviously a dedicated microphone must be added. It depends on the preferences of the lecturer if fixed microphones, headsets or other wireless microphones should be used.

Overhead Slides:

As document cameras are inappropriate technology in an all-digital environment [Effelsberg and Geyer, 1998], we demand presentations to be computer-bound. However, the use of computer-based presentation software is more and more replacing traditional blackboards or overhead slides anyway, which is especially the case for computer science courses. *Digital slides* can easily be edited and adapted and thus offer a higher degree of reusability. Static computer-based slides can be conserved in their *original format* or in the form of *pixel-based slide images*. The way in which content is stored has a crucial impact not only on the recording process, but also on the playback features that can be offered by electronic lectures. Storing presentation content is a core element of this thesis and therefore demands a more detailed discussion, which is given in a separate section (Section 2.2).

Blackboard and Overhead Notes:

Digitizing traditional blackboard-and-chalk lectures in an acceptable quality can be achieved only if a large amount of personal and technical effort is invested both during and after the lecture [Lauer and Ottmann, 2002], which does not conform with our lightweight approach. We suggest two approaches (which can be combined) of transition towards computer bound lectures: *computer-based slides* or an *electronic*

whiteboard. Teachers tend to just copy their notes from their draft to the blackboard (or overhead). In such a case the use of slides is appropriate. The content can be prepared in advance and is then only presented instead of completely written during a lecture, which results in less time for writing and more time for explaining. Especially drawing complex sketches to explain simple issues is effective but inefficient. Think of drawing graphs to explain graph manipulating algorithms. Small graphs consisting of only a few nodes and edges may not be meaningful, but drawing pairs or sequences of larger ones, which only differ in some nodes or edges, is time consuming. Drawing two graphs on a blackboard to show their differences is even more inefficient, but can easily be achieved with digital slides. In addition, electronic slides do not suffer from illegible handwriting.

The essential difference to handouts or books lies in the teacher's hands by means of her/his more or less detailed explanations or the possibility of reacting to the audience. However, chalking up step-by-step notes *dynamically* can be very useful for special purposes such as deriving mathematical proofs. Sometimes dynamics can be achieved be so-called *overlays*, which are a series of slides, where each slide contains a little more information than the previous one. Overlays are subsequentially displayed until the complete slide is visible, allowing verbal explanations of parts without showing too much in advance. For mathematical proofs or creating content while interacting with students this approach is not applicable, because slide creation would be very intricate or not possible at all. In such cases the blackboard can be replaced by an electronic equivalent, often called an *electronic whiteboard*. As everybody can easily figure out by trying to write her/his name within some simple drawing software, the commonly used mouse is not an appropriate input device to replace chalk or overhead markers. Special hardware is rather required enabling teachers to write and draw with pen-style input devices. Such hardware can range from monitor-like sizes (e.g. *tablet PCs* or *Wacom's Cintiq interactive pen display series* [Wacom, 2006]) to screen diagonals of several meters (e.g. *SMART Board interactive whiteboards* [Smartboard, 2006]). Since everybody is familiar with handling a pen, the usage of such input devices should be obvious and intuitive. Personal preferences can be served by small overhead-like or larger blackboard-like realizations. We have to admit that the amount of notes which can be handled by the limited area of such boards is restricted, but hopefully will increase in future such as is described by the *digital lecture hall* approach [Muehlhaeuser and Trompler, 2002], e.g. by use of several beamers which display a history of the last few slides.

The use of such *note taking* should not be limited to empty pages, but rather can (and should) be applied to slide content as well. Additional information can be added if necessary during the presentations. Prepared slides can be annotated while giving explanations (recall the graph flow algorithm example given earlier in this chapter). The focus of the audience can be attracted to essential keywords and important details by emphasizing (e.g. highlighting or underlining) them. This is very similar to annotating overhead slides in the non-digital world with conventional non-permanent pens, but without the nasty cleaning afterwards (if slides are intended to be reused).

Emphases:

The usage of traditional emphasizing/pointing devices such as laser pointers, sticks or pens lying on overhead slides, will get lost during the digitizing process. At the most, the live video showing the teacher may capture pointing actions. However, we stated that the video should be an optional element and, moreover, encoding important information in the video should be avoided [Schütz, 2003], as students may get confused about which media to concentrate on. During a live presentation their main focus is bound to the lecturer, led by her/him with some pointing device and only briefly interrupted whenever a new slide appears[3]. Replaying an electronic lecture (irrespective of whether it is synchronous or asynchronous) the students' main source of information is not the small live video, but the recorded computer-based presentation. Thus, their attention is focused on slides and other presentation content. Trying to figure out by watching a small and/or low quality video to which area of the slide the lecturer is pointing and then finding the corresponding piece of information on the recorded high quality slide is obviously not reasonable (see Figure 2.2). Particularly as the slides might be unreadable in the video and the pointer is only a barely visible tiny red laser dot.

Fig. 2.2. Lecturer pointing to something on an illegible slide

Emphasizing content to attract students' attention should rather take place within the presentation slides. We already suggested underlining by use of freehand drawing as a possibility for gaining focus in the previous paragraph. Advanced annotation objects such as textmarker-style drawings or boxes can also emphasize content. Alternatively, a dedicated pointer such as a large arrow can be used. In an early stage of our e-learning research we have experimented with such an enlarged mouse pointer for the purpose of marking, but we experienced this to be not very useful. The approach works fine as long as the teacher places the pointer carefully. But whenever the lecturer moves the mouse only as computer mouse without being aware of the special emphasizing function, she/he causes the large pointer to refer to an arbitrary object not intended to be emphasized.

[3] Therefore it is good practice to stop verbal narration for a short moment to allow students to survey the newly presented slide.

Dynamic Content and External Applications:

Dynamic elements in the form of animations or simulations as well as annotations (like freehand notes) are very meaningful and should be preserved in a suitable way to allow dynamic replay. However, animations and additional applications used during a presentation have no counterpart in the traditional blackboard or overhead teaching scenario. They are digital per se. Preserving and dynamically replaying such content is discussed in Section 2.2.

traditional lecture	**digitizable analogon**	**electronic lecture**
verbal narration	microphone output	digital audio, e.g. MP3
lecturer's presence	video camera output	digital video
overhead slides	computer-based slides	digital slides or pixel data
blackboard or overhead notes	electronic whiteboard	digital annotations or pixel data
stick or laser pointer	computer-based pointer or annotations	digital annotations or pixel data
—	dynamic elements (animation, simulation, ...)	synchronous start or pixel data
—	additional applications (browser, java, ...)	synchronous start or pixel data

Table 2.1. Elements of traditional and digital presentations

Table 2.1 sums up the aforementioned suggestions and shows an overview of how to transfer elements of *traditional* presentations to a *digitizable* counterpart in order to be preserved as part of an electronic lecture. Different recording approaches will be explained and compared in Section 2.2. A detailed explanation of the recording process used in our approach is given in Chapter 5.

2.2 Symbolic Representation vs Screen Recording

Section 2.1.3 showed how to transfer certain elements of traditional lectures to the digital world, but leaving out details concerning how to access and preserve the presented content (such as slides, annotations, animations and external applications). This section addresses the recording, or more precisely the grabbing process, with focus on presentation content. Recall that the term *recording* is also used in the sense of "grabbing" (for synchronous as well as for asynchronous electronic lectures). Audio/video grabbing stays the same for different recording approaches and has already been discussed in the previous sections as far as is necessary for that stage of the thesis. Furthermore, we address computer-based digital recording only, because "simply videotaping lectures [...] lead[s] to unacceptable quality of the filmed blackboard or slides" [Lauer and Ottmann, 2002]. Note that the availability of increased computing power, appropriate storage capacities, very fast network connections and high-resolution cameras, may enable the filming of a blackboard presentation at a suitable quality at some point in future, but todays technology is either not affordable or not suitable to do so.

[Lauer and Ottmann, 2002] distinguish three different alternatives for presentation recording software:

1. existing presentation systems with added recording functionality;
2. screen grabbing / output grabbing;
3. specifically designed tools for presenting and recording course contents.

Although designing a new piece of software offers more possibilities for addressing special requirements, we do not see the necessity of separating between adding recording features to an existing presentation software or designing a new presentation software supporting such features from scratch. Recording software should rather be classified by the concept of the produced output as this heavily impacts the *representation of content*, which is also addressed as key criterion by [Lauer and Ottmann, 2002]. Thus, we distinguish two groups of lecture recording applications, which differ in the way they access and store presentations:

1. *symbolic recorders* preserving *symbolic information*;
2. *screen recorders* storing *pixel-based* data.

The screen grabbing class of Lauer and Ottmann stores lecture content as pixel data and matches our class of *screen recorders*. Those recorders grab and conserve the *output* of a presentation independent of the underlying presentation software or the formats of the presented documents. On the other hand, *symbolic recorders* are closely connected to (and normally integrated with) the presentation software and therefore can preserve all the features of the original presentation by storing the symbolic information of the presented *input* document plus events occurring during

the presentation process. Regarding this, the classification can also be seen as *input* versus *output grabbing*.

Some presentation software recorders store static pictures of slides only, these pictures are then pixel-based. Nevertheless, such recorders gain symbolic information from the adherent presentation software and preserve them as part of the produced recording. Such hybrid solutions are therefore counted as members of the class of symbolic recorders, because they have access to symbolic content during the recording process. That is also reasonable as many presentation software applications support input documents composed of sequences of pixel images anyway.

Symbolic Representation

Symbolic representation formats, such as PDF, store graphical content as vector data and textual content as sequences of characters with information about coloring, font style and size. In order to generate a visible (or printable) output, the symbolic information has to be interpreted. The interpretation can be modified e.g. using different colors for visually impaired people or another font can be used if the specified one is not available or not wanted. Furthermore, symbolic representation offers scaling. Vector graphics can be scaled and font sizes can be adapted to fit the resolution of the viewer's machine. Symbolic information will be interpreted not only during live presentation, but also during any replay, which enables individual scaling (and other personal interpretation options). Symbolic recordings preserve document structures, which not only allows slides to be distinguished but, equipped with adequate timestamps, also enables access to individual slides and thus provides *slide-based navigation*. The textual content can be searched and edited, which allows information retrieval (full text search), correction of scribal errors or insertion of additional explanations. Furthermore, symbolically represented recordings normally result in smaller file sizes.

As the symbolic data must be accessible in order to be preserved, the recording process must be closely connected to the presenting process. Therefore, the recorder functionality is commonly integrated into the presentation software (either as add-on or by design). Thus, recorders can access presented documents and applied actions, e.g. switching to another slide. During the recording process (a copy of) the source of the presentation (the input document) is tagged with synchronization data, e.g. timestamps referring to actions. During replay, actions can be applied to the sources again. For instance a list of pairs, each consisting of a slide reference and a timestamp, can be used to replay corresponding slides at appropriate positions in the timeline in synchronization with the audio playback.

However, the technique of symbolic recording is rather restricted concerning the input formats and applications to be recorded. Teachers are restricted to using a specific piece of presentation software, because the recorder is a part of that software and, if only one platform dependent implementation is available, is also bound to a single operating system. As a consequence, the supported input formats are restricted by the presentation software as well. *Authoring on the Fly* (AOF) [AOF, 2006, Bacher et al., 1997] uses a proprietary format, which demands slides to be created with special applications (*AOFwb* or *mlb*) only. Import filters (e.g.

for *PowerPoint*) improve the usability, but cannot compensate for all incompatibilities (e.g. animations). Furthermore, such recording systems are typically designed to record the presentation only. The parallel use of multiple software applications (e.g. presenter and browser) during a lecture is rarely supported or not possible at all. AOF allows the running of external applications, but the recorder only preserves which application was started and when. During playback the application is started at the appropriate time, but runs asynchronously thenceforward without further synchronization. Thus, a recording does not really preserve what happened during live presentation, but rather the playback engine only initiates a rerun of the presented application or, to be more precise, an application of the same name as the presented application. Moreover, the application must be available during playback time, which demands that any special applications are copied with the electronic presentation and installed on students' machines. Even more severe is that the recorder does not preserve the input applied by the teacher to the external application. Consider a teacher opening a browser and manually entering an URL[4] to show a certain web page. During replay a web browser will open, but displaying the default start page only. Entering the appropriate web address must be performed by the student, who probably will not know the address. All other input performed by the teacher such as selecting links or entering text in form fields will also be lost. Students would have the reapply any clicks and any textual input the teacher has performed during presentation. Additional problematic issues are deleted or modified web pages, outdated links or the unavailability of a network connection during replay. Similar problems occur for any other kind of interactive applications. Thus, only inputless stand-alone applications or self-performing documents associated to corresponding playback applications, such as Flash animations, that can be distributed to students (technically and legally) can be used in a meaningful way.

Dynamic elements and annotations, which are handled by the presentation software itself, can be accessed and preserved by an integrated recorder, at least theoretically. Advanced presentation software, such as MS PowerPoint, offers a vast variety of dynamic elements (e.g. animations, slide crossfading or even JavaScript[5] programs), but additionally integrated recorder plug-ins rarely support these features. Sometimes not even freehand annotations are preserved by the plug-ins. On the other hand, especially designed symbolic recorders with built-in presenters mostly offer a built-in annotation system and provide dynamic annotation replay, but are often limited in their capability to integrate other dynamic elements.

In summary, *symbolic representation* offers good editing, navigational and retrieval features, which makes it very useful for postprocessing and playback demands, but rather restricts lecturers in the content creation phase as well as during the presentation process. Displaying and recording dynamic elements may not be fully supported and a meaningful usage of external applications is almost impossible due to the loss of applied inputs. As a consequence, lecturers may be forced to change their teaching styles, which contravenes our *transparent* approach (Section 2.3). Reuse of previously

[4] Uniform Resource Locator (URL): a unique address of a document or another resource on the World Wide Web

[5] JavaScript is a scripting programming language compliant to the ECMAScript Language Specification [Ecma, 1999]

created but now incompatible sets of slides is impossible or needs at least a (perhaps manual) transformation process.

Screen Recording

On the other hand, consider *screen recorders*, which store the *graphical output* (in the form of *pixel data*) of a presentation and replay it exactly as it was displayed on the screen during the live presentation. Screen recording provides lossless high quality access to a presentation machine's desktop. Storing pixel data is very *flexible* as it allows everything happening upon a presenter's screen to be preserved, independently of the applications used. Teachers are free to run any presentation software, which makes any input document formats accessible. There is (almost[6]) no limit concerning the recording of additional applications. In contrast to the *symbolic recording* approach, *screen recording* offers real replay of what happened during the presentation without demanding data transfer or the installation of software which was/is used during presentation. Furthermore, any annotation system of any presentation software can be recorded and all dynamic elements like animations, annotations as well as pointer movements are retained.

Unfortunately, this flexibility is also the primary drawback of this approach. Screen recorders are independent of presentation software and annotation systems and therefore have no access to document structures, textual content or any events caused by the presentation software. The inaccessibility of input documents and events inhibit comfortable navigational, post-processing and retrieval features. Conventional screen recorders only offer video-like editing options (e.g. cutting and concatenating), sequential playback and at best timeline navigation.

In order to conserve a lecture, a screen recorder needs to *digitally* access the graphical output. One possibility is *screen grabbing*, which periodically grabs the complete desktop as a *bitmap* (e.g. by copying the graphic card's buffer). Storing uncompressed bitmaps at high frame rates (bitmaps or frames per second = fps) obviously creates huge amounts of data. Grabbing a screen of 1024×768 pixels at 32 bit color depth and 20 fps leads to 3600 Mbyte per minute. Standard picture compressing algorithms may not be sufficient to reduce the data to a manageable amount. Rather, screen recorders demand compression algorithms designed for motion pictures. Standard video codecs provide much higher compression rates by allowing lossy compression. However, to provide high quality multimedia-based learning materials, not only the quality of the contents and its visual representation in respect of pedagogical issues are crucial, but also its visual representation regarding image quality. A low quality video transmission of the speaking teacher is no problem. We recognize who is talking, his/her gestures and movements even if some details are missing. Applying lossy compression to the content of slide, on the other hand, can result in illegible text or indistinct sketches and tables as demonstrated by Figure 2.3, where the lossy area was achieved by applying JPEG compression with very low quality settings, or Figure 2.2, which shows a video screen shot of a filmed presentation. Even if the degree of loss is not that bad and the slides are only blurred instead of being unreadable, it is at least hard to follow the lectures, because reading low quality slides over a period

[6] Full screen movie replay during representation generally exceeds recording capacities

of 90 minutes is awkward and tiring (even if the lecture is interesting). As standard video compression algorithms are designed to encode real world videos, they are adequate and unproblematic for live videos, movies, video conferencing, etc., but "does in most cases neither lead to acceptable quality nor to a reasonable size of the compressed file" [Lauer and Ottmann, 2002] if applied to the contents of a graphical desktop.

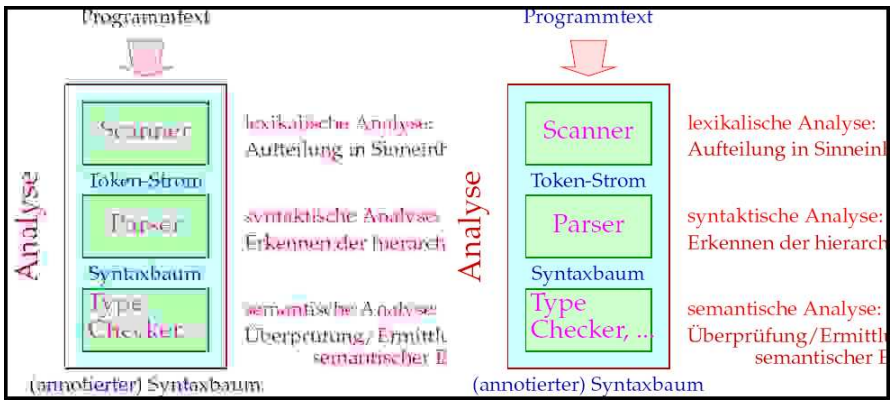


Fig. 2.3. Lossy vs lossless compression

During ordinary slide presentations the screen content is rarely altered. Changes occur only whenever the teacher switches slides, except for maybe a clock or the mouse pointer, which causes small parts of the screen to change frequently. Hence, two bitmaps representing two sequential frames will rarely differ very much from each other, which offers large potential for compression by storing frame differences only. The *TechSmith Screen Capture Codec* [TSCC, 2006] used by the commercial screen recorder *Camtasia Studio* [Camtasia, 2006] is especially designed for lossless and efficient compression of screen content. Unfortunately, it is only available for MS Windows systems and still results in huge files if a high frame rate is applied, as is necessary to enable smooth playback. Lowering the frame rate reduces the memory consumption, but also reduces the quality. Instead of a smooth playback, pointer movements and animations look jerky. Dynamics are (partly) lost. This is the same for all other codecs that support fixed frame rates only. Dynamically adapting frame rates, which allow the usage of many frames whenever necessary to preserve dynamics but reduced frame rates elsewhere, would be preferable but are rarely supported. Most compression algorithms support different, but fixed, frame rates, selectable in advance of recording only.

Another approach for conserving the graphical output can be *output grabbing* in a more general sense. Instead of recording bitmaps it is also possible to store transmission protocol information, as is done by *VNC session recorders* (see Chapter 5) or in the University of Mannheim's *Interactive Media on Demand* (IMoD) system [Hilt et al., 2001] based on their *Real Time Protocol for Distributed Interactive Media* (RTP/I) [Mauve et al., 2001]. Protocol information is tagged with synchronization data and replayed similarly to the symbolic representation approach. However, depending on the recorded protocol, no document content and structures are stored,

but rather messages describing screen content. Such event-based recording is not constrained by a predetermined fixed frame rate, but rather stores data on demand, which offers more flexibility concerning the trade-off between preserving dynamics and achieving small file sizes.

	Symbolic Representation	***Screen Recording***
Presentation Software	dedicated presentation software with integrated recorder	any
Input Formats	restricted by presentation software	any
External Applications	limited or not supported	any
Annotations	built-in annotation system of presentation software	any (recorder built-in and presentation software)
Dynamic Elements	dynamic annotations; others are rarely supported	any (limited by frame rate)
Retrieval	searchable (full text search)	—
Navigation	slide-based	timeline only
Editing	content editable	video-like editing only (cut and concat)
Scalability	scalable fonts and vector graphics	—

Table 2.2. Symbolic Representation vs Screen Recording

Table 2.2 presents an overview of the comparison between the two recording approaches. In summary, we have *flexible recording*, i.e. a flexible environment that supports the recording of arbitrary content and applications, versus *structured recordings*, i.e. asynchronous electronic lectures in the form of structured files, which enable enhanced post-processing and playback features. However, the optimal solution should enable both a flexible and transparent easy-to-use recording process producing electronic lectures with effective navigational and retrieval options.

One attempt to circumvent this dilemma is offered by *Lecturnity* [Lecturnity, 2006], a commercial descendant of AOF [AOF, 2006]. This recorder stores symbolical data, but offers an integrated screen grabbing utility to capture external applications on demand. In fact there are two recorders integrated in one environment. Unfortunately, whenever teachers leave the slide presentation (recorded by the symbolic recorder), then they have to explicitly initiate the screen grabbing process by starting the screen recording component and selecting which application has to be recorded. It is certainly much easier to use the coupled screen recorder than two stand-alone solutions. However, as the usage of other applications is possible without recording them, teachers tend to forget to start the recording process. In such cases only the verbal narration, but no application, is recorded. Furthermore, this approach has still limited navigational and retrieval options for screen recorded parts and is meant for teachers who rarely use additional applications.

Another approach is given by *tele-TASK* [Schillings and Meinel, 2002, teleTASK, 2006]. This screen recorder delivers a special plug-in for MS Pow-

erPoint, which is connected to the recorder. Thus, the recorder can preserve events from the presentation software, such as switching to another slide, and hence, enriches the produced electronic lecture with slide-based navigation. Unfortunately, this approach reduces flexibility as it demands presentation software dependent plug-ins (currently only PowerPoint is supported. Furthermore, transparency is reduced, because the teacher must install and remember to start the plug-in prior to recording.

Our research followed an alternative approach. Starting with a flexible screen recording system, the produced electronic lectures are enriched with structure to provide navigation functionality (Chapter 7) and integrate retrieval options (Chapter 8) by automated post-production independent of the presentation (or any other) software which was used.

2.3 Lightweight Production and Transparent Recording

While discussing *electronic lectures* (Section 2.1) we have stated that *lightweight course production*, which is "to record a quite regular live lecture in order to obtain versatile digital material with negligible additional cost" [Kandzia and Maass, 2001], is a preferable solution for producing digital learning material in a cost-effective way. High quality studio productions demand a lot of preparation to select and create teaching contents. Special technical equipment and many manual resources are needed during production and post-production. "Experience gained in several large-scale projects in Europe has shown that production costs for rich-media content in the range between 50,000 and 100,000 EUR per weekly lecture hour are no exceptions" [Lauer and Ottmann, 2002]. This is mostly not affordable, especially recalling that University teaching should stay up-to-date with research, but updating content is not easy to achieve in such a high quality production process. *Lightweight recording* of regular *live lectures* in order to create *electronic lectures* tremendously reduces production costs, but still offers meaningful digital learning material. Furthermore, lightweight recording enables large *archives of electronic lectures* to be built up in a short time and thus delivers a basis for virtualizing teaching processes.

As *lightweight course production* is kind of an add-on, it must not negatively influence live lectures. The recording functionality should rather be *seamlessly* integrated into the existing lecturing environment. At best, presentation grabbing/recording should be *transparent* to lecturers [Ziewer and Seidl, 2002]. In other words teachers are not aware of the recording process at all, but are free to perform their preferred traditional teaching styles. Thus, the presenting process stays the same, no matter if a lecture is recorded or not. A *transparent recording technique* not only does not disturb teachers willing to apply modern teaching approaches, but also will reach a larger group of teachers because recording is not dominated by laborious preparation and complex operation. The familiar act of presenting is entirely sufficient to achieve electronic lectures. Such an approach conforms to [Muehlhaeuser and Trompler, 2002], who advocate a *smooth transition* from traditional to digital teaching in order not to create powerful teaching environments for small elites and leaving behind the big mass of average teachers. This approach also corresponds to [Schütz, 2003] who said that "the main goal ... is not to produce the best e-learning content, but to produce it very easily". Certainly, some requirements and restrictions are inevitable. In order to achieve valuable electronic lectures only *computer-based* presentations can be recorded. Therefore the elements of traditional teaching must be transferred to *digital analogons* as described in Section 2.1.3. Now we will discuss possibilities of seamlessly integrating recording techniques and equipment into live lectures, in order to achieve a *transparent recording environment*. Nevertheless, it is not our intention to give a solution fitting all possible circumstance but rather to offer hints, because local conditions, available hardware and applied recording techniques can differ considerably from one another in numerous aspects.

Audio

Audio recording can be seamlessly integrated by using pre-existing hardware. Large lecture halls are commonly equipped with at least one microphone and an adequate

system of loudspeakers. In this case it is sufficient to connect the recording system to the existing equipment. Technically this can be done by the use of a simple Y-splitter cable or better by connecting the recording system to a line-out of an amplifier or an interposed mixer. If wireless microphones are used, the recorder can be connected to a second receiver, which is adjusted to the same frequency as the static equipment of the lecture hall and thus receives the same input. The wireless scenario is even easier to integrate as it does not matter where the static equipment and the recording devices are located. Some lecture halls provide a special room for technical equipment in the back, others have the amplifiers installed at the front desk or maybe even integrated within the speaker devices. Additionally, recording approaches differ in their requirements and possibilities. Some recording systems are integrated within the presentation software and recording takes place at the presentation machine. Other scenarios consist of several machines, which can be locally distributed.

For smaller presentation rooms, where no voice amplifying is required, a dedicated microphone must be added, obviously. It depends on the preferences of the lecturer if fixed microphones, headsets or other wireless microphones should be applied. A static microphone requires the teacher to stay within its reach. Wireless devices offer more flexibility, but must be attached to the lecturer. Hand-held microphones or microphones which are attached to the teacher by a wire, are discouraged as they heavily influence a lecture because they restrict teachers in their movements.

A simple solution with one microphone is sufficient for recording the teacher's talk. Questions from students are a little problematic. Highly equipped lecture rooms with integrated microphones for each student are only available in rare research scenarios. Therefore, questions as well as any other verbal output from students are not captured by microphones and thus get lost during the recording process. As a consequence, the teacher's answer can be meaningless. However, this problem also occurs for any lectures taking place in large lecture halls even without recording them. An unamplified voice is hardly audible in a lecture hall with a capacity of several hundred students. Handing a microphone to a student prior to asking questions is cumbersome and will often limit the number of questions asked. If a question is important for the entire audience, the teacher should rather repeat it or at least phrase the answers in a way so that it will make sense without exactly knowing the question.

Video

The video equipment provided by most lecture rooms is limited to visual presentation (video projector, overhead projector or diascope) not offering video recording. However, no special high tech cameras are required as they will always exceed the intended output quality. Rather, any standard video cameras are sufficient, often even small internet cameras.

More significant is the placing of the camera so that it does not disturb the lecturer. A large camera placed on a tripod just in front of the teacher and a technician working behind the tripod panning the camera according to any small movements of the teacher will obviously be a disruptive factor. A technician also does not fit

into our low-cost lightweight production approach, which claims to limit manual resources. In most cases a small fixed camera is satisfactory as it is not intended to produce high quality video material. Remotely controlled cameras or so-called pan-tilt devices, which recognize movements and adjust themselves automatically, may be used if camera panning is required.

Sometimes additional video devices such as VHS[7] players are used during a lecture. In order to achieve higher quality in the recording, the players should rather be connected to the video input of the recording machine if possible, instead of projecting and filming them via video camera. In one of our settings, we use a (hardware) video mixer which makes it possible to switch the input of the lecture recorder.

Presentation Content

Transparent recording in the sense of presentation content grabbing must access and preserve whatever is presented to a local audience in a real lecture hall. Computer-based presentation systems commonly consist of a computer, typically a laptop, connected to a video projector. The output projected to the wall exactly matches the output visible on screen. Furthermore, the usage of some slide-based presentation software in fullscreen mode is common practice. To achieve absolute transparency this should stay the same. Integrating the recording process into a presentation environment can be handled by different approaches differing in their placement relative to the presenting computer (i.e. the computer that is locally present in the lecture hall and is connected to the video projector):

I. Output Recording: In order to preserve what is projected onto the wall, it would be straightforward to film the wall and thus the output of the video projector. But such videotaping is discouraged due to the loss of quality as already mentioned in the previous sections. In our digital setting, we rather would record the output of the presentation computer as this is the input for the video projector. Such output recording can be achieved by use of special digitizing hardware, such as AdderLink IP[8] [AdderLink, 2006], which is connected to the graphical VGA[9] output of the presentation machine in the same way as the video projector is. Due to the hardware costs it might not be the cheapest solution, but it is the most flexible one because it allows the output of any computer to be captured without installing any piece of software or drivers.

Software recording generally results in lower costs. A software solution for screen recording accesses the displayed screen content on the presenting computer via the operating system or graphics card. As output recording is independent of the underlying presentation software, it can store pixel data only. The received pixels must be compressed and stored in a suitable way (see Chapter 2.2).

II. Parallel Recording: Presentation software with integrated recording functionality also offers software solutions for lecture recording. While displaying presentation

[7] VHS abbr. for Vertical Helical Scan but commonly used as abbr. for Video Home System: a recording and playing standard for video cassette recorders
[8] AdderLink IP is a VNC compatible KVM-via-IP solution
[9] Video Graphics Array (VGA): an analog computer display standard

content on screen (and thus via video projector to a local audience), the presentation software will also store the content plus additional media streams (audio and/or video) for later replay. Such background recording does not influence the presentation process as long as the recording process does not consume too much processing time, which possibly could slow down the presentation. As discussed in Chapter 2.2, such presentation software recorders can access and store the symbolic representation of input documents and thereby preserve structuring as well as textual content.

III. Input Recording: Another approach is available for distributed infrastructures, such as a videoconferencing environment or a set-up where the presenting machine accesses the presented content located on another computer via a network. A recorder accesses the data transfer between the content providing and the presenting machines (e.g. interposed as proxy). The data transfer can be logged and replayed later in the same way. Moreover, it can be interpreted in the same way a presentation software would do and then handled as in the parallel solution (just without presenting the content). Note that those different tasks are not necessarily distributed to different computers. The recording software can be a process running on either the recording or the content providing computer. Even those two tasks (and therefore all three) can be performed on a single computer in today's multitasking systems.

Other distributed scenarios do not transmit presentation content to the presenting computer, but redirect the output of a presentation to the presenting machine. This means, the presenting machine located in the lecture hall and operated by the teacher is not performing the presentation. In fact, it works as a client accessing a server which runs the presentation software. Thus, the output of the presentation is transferred from the server to the computer in the lecture hall, which presents it to the teacher and the local audience. The input of the presenting computer is the output of the machine that performs the presentation. Recording such data transmission obviously does not offer access to input documents or events caused by the presentation software. Hence, only pixel data is stored.

Controlling

In some circumstances a loosening of the transparent concept is inevitable. This is the case whenever the integration of additional features demands some controlling possibilities. Obviously some user interface for interacting with the system has to be integrated into the recording or presenting software, respectively. However, the presentation system should stay as transparent as possible and the ease-of-use of new features should have first priority in order not to disrupt the natural flow of a live presentation. Therefore, simple but effective user interfaces must be developed and integrated only. Too many separate windows or different user interfaces for slightly different scenarios confuses teachers and students [Effelsberg and Geyer, 1998]. Respecting the *smooth transition* approach, no lecturer should be forced to use additional features, but should freely decide if and when to do so. Individual configuration of a common adequate standard user interface can provide more flexibility by allowing controlling elements to be added or removed.

Examples of additional features are controlling the recording process, which obviously is not needed in presentation-only environments, or annotation systems, which

can be meaningful for the local live presentation as well (recall the graph flow algorithm example from Section 2.1.1). If recording is not controlled by a technician in the background, but rather teachers themselves initiate and stop the recording process, at least one start/stop button or keyboard shortcut must be available. Omitting buttons and using shortcuts only has the benefit of not disturbing the visual representation of slides, but starting and stopping the recording process can easily be forgotten. If some graphical user interface is present anyway (e.g. for annotations or slide control), an additional button does not matter. If the recording process is performed by use of dedicated hardware, buttons are probably provided in hardware instead.

Fig. 2.4. MS PowerPoint 2003's annotation controls (closed)

For annotation systems there must be an option to choose and place annotations, and usually one can select an annotation mode and maybe the coloring. At best, interactions should be accessible via a single click. The menu style annotation controls of MS PowerPoint has almost no affect upon the visual presentation itself (Figure 2.4), but unfortunately always requires two clicks to switch between paint modes (one to open the controls and one to select the mode) and the color selection demands even a second level menu (Figure 2.5).

Fig. 2.5. MS PowerPoint 2003's annotation controls (opened)

We rather suggest that annotation tools should be accessible in a comfortable way, at best by a single user interaction. A toolbar with the most frequently use functions will suitable. A hardware solution is offered by *SMART Technologies*. Their whiteboards

are supplied with several electronic pens and the teacher can use different pens in order to chose different colors [Smartboard, 2006].

Navigational and retrieval features are certainly crucial usage criteria of electronic lectures, but must not demand additional work to be done *during* a live presentation by the lecturer. [Li et al., 2000a] suggest to select any headlines during a presentation in order to produce navigational marks which can be accessed during later replay. However, this might irritate the teacher and also the students. The teachers' tasks and thus the user interface should rather be limited to whatever is required for presenting and recording purposes only.

In fact the controlling user interface is meaningful to the teacher and within the recording context only, but unfortunately will also be presented to the local audience. Special hardware, so-called scan converters, or even many of todays video projectors allow clipping of video output in order to display a selected section only. If the graphical user interface of the presentation/recording software is located at the border of the presentation area (which is advisable in order not to overlap and hide something), they can be clipped out. This is obviously not possible using electronic whiteboard hardware (e.g. Smartboard), where teachers and students must see the same, because the board is used as the input device. The electronic lectures which are produced do not suffer from unwanted graphical user interfaces, because only the presentation content will be recorded (except for hardware output recording, which will receive the same data via the graphics card as a local video projector).

2.4 Criteria and Features

In the previous sections we have regarded three criteria, *lightweight content production*, *transparent lecture recording* and *smooth transition* (from traditional to e-teaching), giving guidelines to produce electronic lectures in a fast, easy and comfortable (for the teacher) way. As those are no *true-or-false criteria*, the term *strategies* might be more appropriate. To summarize, those strategies and guidelines are:

I. **Lightweight Production**

Producing electronic lectures should be fast and inexpensive, i.e.:

a) producing electronic lectures by recording real live lectures,

b) reducing manual and technical resources,

c) utilizing highest possible degree of automation.

II. **Transparency**

The recording process should not (negatively) influence the natural flow of recorded live lectures, rather teachers should be unaware of the recording process. This can be achieved by:

a) supporting the teacher's choice of presentation software and documents,

b) recording in the background,

c) seamless hardware integration (concerning type and placement),

d) if additional user interfaces are inevitable, keep them concise.

III. **Smooth Transition**

The recording environment should aid teachers by offering beneficial features, but not force their appliance:

a) ease-of-use (at best in a familiar style),

b) enabling quick-start for untrained (with regard to lecture recording) teachers without additional preparation,

c) step-by-step integration of additional features at teacher's pace.

In addition to these base strategies giving ideas and hints for how to achieve cost-saving and teacher friendly content production processes, a detailed criteria and feature catalog for both production and replay of electronic lectures is needed. The aim is to compare existing environments and approaches exposing flaws and restrictions of today's lecture recording technology. In the following chapters we will then offer theoretical ideas and practicable implementations to loosen restrictions and

eliminate flaws and, consequently, suggest solutions to achieve more suitable and convenient electronic lectures.

There are *essential features*, which are indispensable to achieve meaningful replay, and *desirable features* offering very useful aspects to enrich the value of an electronic lecture. Rating the features depends on user preferences and some features might conflict with others. Hence, we cannot design a 1 to 10 scale which could be used to create the perfect 10 electronic lecture recording and playback environment, but rather achieve hints and suggestions for some classification. Hardware aspects are left out here, because they are almost the same for any kind of presentation recording environment.

Obvious *essential features* for slide-based electronic lectures (stated by [Lauer and Ottmann, 2002, Brusilovsky, 2000]) are:

LO1 **Verbal narration:**

audio recording of the teacher's voice

LO2 **Static slides:**

storing slides displayed during presentation and slide replay synchronized to the verbal narration

Recording slides without verbal narration delivers no more than a simple script (maybe with additional annotations) and therefore we will not call it an electronic lecture. Audiobooks intended for listening purposes only or referring exactly to certain textual learning material, e.g. by referencing by page and section numbers, are meaningful without recorded slides. In the case of recording live lectures, a verbal narration without interlinkage to visual content is impossible (or at least very hard) to follow, because it is not intended to be used in such a way.

Beyond this elementary functionality, [Lauer and Ottmann, 2002] list nine criteria for the evaluation and comparison of presentation recording systems:

LO3 **Representation of contents:**

preserving symbolic information of the original content wherever possible, because it offers:

a) information retrieval (e.g. full text search)

b) scalability

c) later manipulation of content

d) reduced amount of data compared to video formats

LO4 **Dynamic capture and replay of annotations:**

a) preserving the dynamics of teacher's handwriting and other graphical annotations

b) associate annotations with slides (so that annotations disappear when a slide is changed and made visible again when returning to that slide later during presentation)

LO5 **Structured overview of recordings:**

a) automatically created overview (e.g. via thumbnails) or table of content showing the structure of the presentation

b) offering comfortable navigation by accessing structured elements

LO6 **Visible scrolling during replay:**

a) browsing through an electronic lecture by dragging a slider along a scrollbar with instant update of the display during dragging (and not afterwards, when the knob is released)

b) real-time random access without noticeable delay (required to achieve fast visible scrolling)

LO7 **File format and size of recorded documents:**

a) small file sizes appropriate for downloading

b) platform-independent format

c) standard web-based format

d) streaming ability

e) lossless compression of slides and presentation content (to achieve sufficient quality)

LO8 **Formats of material which can be captured:**

a) recording any given file format

b) or application used

LO9 **Capture and integration of live video:**

video picture of the teacher

LO10 **Post-editing facilities:**

although opposing the lightweight strategy some editing features might be useful

a) cut and join parts

b) manipulating slide content (can best be achieved if represented symbolically)

LO11 **Turning records into CD-Rom or web-based course:**

easy and automated way of combining recordings with other materials as part of integrated e-learning modules to organize a course in a web-based learning management system or on distributable storage media

[Lauer and Ottmann, 2002] addressed asynchronous electronic lectures only, but [Kandzia et al., 2004] recall and summarize those criteria and add:

LO12 **Synchronous transmission:**

Transmitting live lectures either to other lecture halls and/or to online students

Although we agree with most of the suggested aspects, we will rework and extend them. Extending the list is needed to add further synchronous aspects. Rework mainly concerns the intertwining of symbolically represented content and features, such as scalability, structuring and retrieval. The representation of content certainly has a crucial impact on some features, but the features are what we demand in the first place. Enabling any of those features without using symbolically represented content is sufficient as well. Therefore our criteria catalog should rather list information retrieval LO3a and scalability LO3b as individual features independent of the content representation aspect. The same holds for structured recordings (already listed separately), manipulation of content (listed twice LO3c and LO10b) and data amount (also listed twice LO3d and LO7a).

In addition to dynamic annotations, the aspect of dynamics should be extended to capturing and replaying any dynamic elements including pointer movements, animations and interactions with arbitrary software (covered indirectly by LO8).

Real-time random access LO6b is a condition precedent to enable visible scrolling LO6a, but is beneficial for any kind of navigation (e.g. accessing slides) and hence a desirable feature of its own.

[Mertens and Rolf, 2003] suggest the following features of an ideal recording tool (partly derived from [Lauer and Ottmann, 2002, Brusilovsky, 2000]):

MR1 **Advanced navigation**

such as structured overview (LO5) and visible scrolling (LO6)

MR2 **Animations**

reproduction of animation (effects) as part of a slide presentation

MR2 represents animations, which are provided as part of a slide presentation within a presentation software. These could be simple fade-in/fade-out effects applied for slide changing, but also any other sophisticated animations used not only for the effect, but rather for educational issues. Mertens and Rolf include reproduction of such animations (or animation effects), because systems like Lecturnity [Lecturnity, 2006]

convert slides to proprietary formats, which may not support all kinds of animations and features of the original document. Mertens and Rolf suggest decomposing an animated slide into a sequence of several (sub)slides to reproduce such animations. However, this is also a conversion of slides, which may support more effects, but again may be incomplete. The drawback of converting slides is that the conversion algorithm is most probably outdated for any new release of the original slide format. Nevertheless, reproducing animated slides instead of supporting static slides only is a desirable feature for a lecture recording environment.

MR3 **Capture of hand-written annotations**

MR4 **Full text search**

Annotations (MR3) and full text search (MR4) are analogous to LO4a and LO3a, respectively.

MR5 **Line based addressability**

accessing lines rather then slides (as in [Brusilovsky, 2000])

MR6 **Line based synchronization**

synchronize audio and video streams according to accessed lines

Navigational features (MR1) are covered more precisely by Lauer and Ottmann (LO5 and LO6), but *line-based navigation* (MR5 and MR6) (also suggested by [Brusilovsky, 2000]) delivers an additional aspect, which is hardly fulfilled by any of today's recording systems. Line-based addressability and synchronization can be merged to a single feature because one is useless without the other. In fact, line-based navigation and slide-based navigation are subtopics of a structured overview/access (LO5).

MR7 **Metadata**

incorporation of metadata in the recorded document (e.g. keywords, course description, language of course) to make recorded documents accessible for content management

MR8 **Post-editability**

in the meaning of LO10

MR9 **Screen recording**

whenever a lecturer switches to another program

Demanding a screen recording feature mixes representation of content and the intended feature of an ability to record additional applications other than the presentation software (analogous to LO8b). Admittedly pixel-based recording is the best

suited and maybe the only full featured solution to achieve this criterion. If screen recording is also applied to the presentation software, the animation feature (MR2) is covered per se.

MR10 **Searchability by conventional search-engines**

generate electronic lectures, which are indexable by web-search-engines

The searchability by conventional search engines offers an additional and meaningful retrieval aspect not addressed by Lauer and Ottmann. Additionally, [Jackson et al., 2000] state searchability over all types of media, i.e., text, graphics, audio and video, as a desirable retrieval feature, although it is not quite clear what to search for in a video, which mainly shows a speaking teacher. Furthermore, they suggest searchability not for a single electronic lecture only, but for an entire library of courses as well, which, for sure, is a meaningful aspect. Such search functionality is somehow a loosening of MR10, because retrieval is enabled for large data bases, but not necessarily by use of conventional search engines.

MR11 **Supplemental delivery of annotations, links and references at any time**

enable adding and altering of additional information and learning material related to certain parts of an electronic lecture by teachers and students (e.g. wiki-like)

The integration of metadata (MR7) and supplemental elements (MR11) are not listed by Lauer and Ottmann. However, the group of Prof. Ottmann (the developers of the *Authoring On The Fly* (AOF) system [AOF, 2006]) discusses the integration of student notes [Lienhard and Lauer, 2002, Lienhard and Zupancic, 2003], which can also be seen as supplemental elements.

Brusilovsky, who has suggested the integration of supplemental elements [Brusilovsky, 2000], also states retrievability of those elements, e.g. searching keywords, comments or links to additional resources (which he calls *annotations* different from our term). A very similar idea is offered by [Jackson et al., 2000], who suggest a *supplementary information* window with extra readings or links. Another feature suggestion by [Jackson et al., 2000] is to offer students the possibility of interacting with their learning environment. They suggest electronic note-taking in the form of text and graphical annotations, post-its (to tag information) as well as captioning short audio or video clips, all of which can be filed under the vague feature of offering supplemental elements (MR11). A fine-grained classification of supplemental elements cannot be achieved[10] due to unequal requirements and very diverse appliance of such additional elements throughout different e-learning systems. In fact, psychological studies and evaluations concerning acceptance by students/teachers and learning benefits are needed to provide a classification.

[10] Unless split into well-defined subtopics, but such discussion is not required throughout this thesis

2.4.1 Criteria Catalog

Merging the different aspects and feature lists with our own experiences gained throughout the last few years, we suggest the following catalog of criteria for evaluation and comparison of presentation recording systems:

C1 **Verbal Narration**

recording the teacher's voice in high quality is mandatory

C2 **Live Video**

a small, *low quality video* is sufficient if showing the teacher only; *high quality video* is demanded for instance to preserve experiments filmed via camera; video may be omitted to reduce bandwidth usage or file size

C3 **Presentation Content**

concerning degree of *full-featured* support and preserved *dynamics*

a) Format of presentation documents

handling *presentation formats* (e.g. html, pdf, PowerPoint incl. dynamic elements), *static slide images* (e.g. BMP, JPEG, GIF) or at best, *any input documents*

b) Supported presentation software

dedicated presentation software (familiar to teacher), *recorder built-in* presentation feature (unknown to teacher) or ideally *any* presentation software

c) Supported additional applications

ability to preserve *arbitrary applications* (e.g. browser) in addition to a slide presentation (incl. menus and pointer movements)

The distinction between supported document formats and presentation software is recommended, because supporting certain input formats regards the process of creating teaching material, but the second aspect addresses the act of presenting itself. Although a presentation recorder supports a certain document format, it does not necessarily support the teacher's choice of presentation software. Consider a teacher who produces teaching material with the often used MS PowerPoint, but uses the recording software Lecturnity [Lecturnity, 2006]. The teacher propably is familiar with PowerPoint not only for creating but also for presenting slides, but must use Lecturnity's integrated presenting functionality. This may annoy some teachers and thus possibly prevent them recording their lectures. On the other hand, supporting the teacher's choice of presentation software offers transparency and ease-of-use during recording as the teacher is familiar with the handling. Of course supporting the teacher's commonly used presentation software implies that her/his presentation documents are supported as well (unless the teacher made some terrible wrong de-

cisions). Furthermore, a recording environment may support a certain input format not directly, but demand some conversion of the original input documents.

C4 **Annotations**

a) Type of annotations

such as freehand notes, graphical objects, textual and audible annotations, references and other supplemental elements

b) Dynamic capture and replay of annotations

preserving the dynamics of handwritten and other graphical annotations (used for emphasizing or note-taking by teachers during live lecture)

c) Annotations associated with slides (or other elements)

so that annotations disappear when a slide is changed and are made visible again when returning to that slide later during presentation

d) Student note-taking

supporting student notes and annotations live or during replay regarding kind of annotations (e.g. references, textual, graphical, audible), maybe distributed among students (and teachers)

C5 **Metadata**

incorporation of metadata in the recorded document (e.g. keywords, language of course, course description) to make recorded documents accessible for content management, also regarding the format of metadata

C6 **Post-processing**

a) Video-like editing

cutting and concatenating

b) Content editing

editable slide content and/or annotations

c) Creation of distributable media

production of lecture archives, web-based courses or distributable media (e.g. CD/DVD)

Criteria C1-C6 regard content and production mainly. At least, sequential replay must be possible to make use of the created electronic lectures, but this alone would not be sufficient for a top-quality learning environment. Rather advanced navigation and retrieval aspects must be addressed:

C7 Navigation

a) Structured electronic lectures

regarding granularity (e.g. *slide-*, *line-*, *element-based navigation*) and representation (e.g. thumbnail overview, table of contents) of identifiable and accessable elements

b) Random Access

possibility to access any position (e.g. *time-line navigation*) regarding access time (*real-time random access* without noticeable delay)

c) Visible Scrolling

browsing through an electronic lecture by dragging a slider along a scrollbar with instant update of the display during dragging (and not afterwards, when the knob is released)

Real-time random access is demanded to achieve meaningful visible scrolling and is suitable for other navigation features such as slide-based navigation. However, accessing (the beginning of) slides can be achieved without supporting random access (e.g. storing separate files for each slide).

C8 Information Retrieval

a) Searchable content

which content/elements is/are searchable (for instance full text search of slides, keywords, annotations, audio or video search)

b) Range of searchability

supporting retrieval within single electronic lectures, databases of archived lectures or via conventional web-search engines

Also more technical issues of electronic lectures are a source of classification:

C9 Format of Produced Electronic Lectures

a) Lossless reproduction

replay content without loss in quality

b) Scalability

scaling content without loss in quality to fit to different screen resolutions

c) Streamability

d) Format

standard formats are preferable to proprietary ones

e) File size and bandwidth

supported bandwidths for streaming; file size for downloading

C10 **Platform Independency**

platform independent recording, post-production and replay

Many issues affect synchronous and asynchronous electronic lectures in the same way. Navigational and retrieval features concern asynchronous aspects. In addition, special requirements of synchronous lectures are handled in the last item of our criteria catalog:

C11 **Synchronous Electronic Lectures**

a) Addressed participants

uni- or bidirectional (audio/video feedback channel) live transmission to distant lecture halls or online students (at their homes)

b) Scalability

number of participants that can be handled

c) Late join

providing late-comers with a consistent state

d) Error tolerance

e) Synchronization

regarding delay (real-time or buffering) and support of multi user inputs (e.g. distributed whiteboards)

We do not claim our list to be complete in all aspects or to be suitable for all kinds of (maybe specialized) evaluations, but rather it offers a good overview most readers can probably agree with. The suggested criteria catalog is derived from the recommendations of various other research results [Lauer and Ottmann, 2002, Brusilovsky, 2000, Kandzia et al., 2004, Mertens and Rolf, 2003, Jackson et al., 2000], which have been merged with our own experiences from recent years. Most items are congruent with features listed by other researchers, but some have been reformulated, combined or segmented to be, in our opinion, more appropriate. In particular we tried to abstract the required feature (which we want to reach in the first place) from a certain way (of possibly many) to achieve that feature. Consequently this catalog reveals the most important features and criteria useful to evaluate and compare lecture recording systems in different aspects.

Note, that the importance of the criteria may be different according to the intendet e-learning scenario. For example, retrievability is very important to locate certain

topics in a huge database of asynchronous electronic lecture, but is typically rather unimportant for any synchronous scenarios.

3

VNC: Virtual Network Computing

Virtual Network Computing (VNC) [Richardson et al., 1998] is a remote desktop environment providing access from a client machine to a desktop on a server machine. VNC was developed at the Olivetti & Oracle Research Lab (ORL), which was acquired by AT&T in 1999 and since then called AT&T Laboratories Cambridge. As part of AT&T's global restructuring of research, the industrially funded part of the Cambridge Laboratory was closed in 2002. However, the original inventors of VNC, the team around Andy Hopper, professor of Computer Technology at the University of Cambridge, continued to develop and maintain VNC under a newly founded company called RealVNC [RealVNC, 2006]. Besides RealVNC there are other full-featured VNC implementations, for instance TightVNC [TightVNC, 2006], UltraVNC [UltraVNC, 2006] or OSXvnc [OSXvnc, 2006], which offer various enhancements like efficient compression algorithms, encryption or file transfer. There are also numerous VNC client implementations or special purpose servers (e.g. exporting a single application or a graphical user interface only). Except for some special featured versions of RealVNC, all these implementations are freely available, both as executables and sources.

3.1 The VNC Environment

In the VNC environment a server machine, called *VNC server*, supplies an entire desktop environment that can be accessed via network from any machine using a thin software client, called *VNC viewer*. The technology underlying the VNC system is a simple protocol for remote access to a graphical user interface. It is called *Remote Framebuffer* (RFB) protocol [Richardson, 2005]. Unlike other remote display protocols such as the X Window system, the RFB protocol is totally independent of operating systems, windowing systems and applications. Thus, it allows cross platform usage between arbitrary operating systems, e.g. accessing a Microsoft Windows desktop from a Linux machine or an Apple Macintosh from Sun Solaris. Besides various VNC implementations for several operation systems, a client implemented as a Java applet offers access to any VNC Server from within any Java capable web browser. While remote desktop access is the main usage of VNC, other scenarios

accessing only a single application, e.g. the controls of an audio player application, are possible.

3.2 Remote Framebuffer Protocol (RFB)

The *Remote Framebuffer* (RFB) protocol is currently available in its version 3.8 [Richardson, 2005], which is backwards compatible to the only previously published versions 3.3 and 3.7. It is a simple protocol for remote access to a graphical user interface. The protocol works, as the name suggests, at the framebuffer level and is based on a single graphics primitive: *Put a rectangle of pixel data at a given x,y position.* A *framebuffer update* represents a change from one valid framebuffer state to another. The protocol is *demand-driven* by the client. That is, an update is only sent by a server in response to an explicit update request from a client, assuring that pixel data is transferred only if the client is ready to process. This demand-driven communication in conjunction with the client's properties concerning color depth and encoding schemes, gives the protocol an adaptive quality. A slow client and/or network results in fewer framebuffers. Furthermore, the RFB protocol is a *thin client protocol* that makes very few requirements of the client. In particular, the protocol makes clients *stateless.* A client can disconnect at any time and reconnect even from another machine and will find the graphical user interface in the same state as left.

The protocol consists of two stages: an *initial handshaking* phase to establish the connection followed by the *normal protocol interaction.* During the *handshaking phase* the two participants agree on a protocol version, handle the authorization and negotiate the format and encoding with which the pixel data will be sent. There are various pixel formats and encoding schemes to compress the pixel data, giving a large degree of flexibility to trade-off various parameters such as network bandwidth, client drawing speed, and server processing speed. A server must respect the client's properties concerning pixel format, color depth, and encodings.

In the second phase, the *normal protocol interaction*, the two participants communicate by exchanging messages. The protocol defines several *client-to-server* and *server-to-client messages*, mainly to notify each other about input events and framebuffer changes, respectively. If the RFB protocol is set upon a stream-oriented data transfer, such as TCP/IP[1] (as is the case for VNC), skipping messages or reading messages without parsing them is impossible. This is the case because messages (even of the same type) have variable lengths, but neither length tags nor message delimiters are provided by the RFB protocol. Hence, messages are intended to be read sequentially and in total.

The input side of the RFB protocol is based on a *standard workstation model* of a keyboard and a multi-button pointing device like a mouse. A client sends *input events* to its server whenever a user presses a key or pointer button or moves a pointing device. So all input events generated by a client are passed on to the server and thus to applications running at server side. The server's task is to determine

[1] Transmission Control Protocol [Postel, 1981b] / Internet Protocol [Postel, 1981a]

pixel changes within the framebuffer (representing the applications). Whenever a client requests to update a rectangular area of the framebuffer, the server transforms and encodes pixels of the specified area according to the client's properties and returns the resulting framebuffer update. A *framebuffer update* consists of a sequence of rectangles of pixel data, which the client should copy into its framebuffer. The rectangles in an update are usually disjoint but this is not necessarily the case. The server usually responds to a request by sending a framebuffer update. Note however that there may be an indefinite period between a request and an update and that a single framebuffer update may be sent in reply to several requests, combining the requested areas.

Update requests can be *incremental* or *non-incremental*. In the case of a non-incremental update request any pixel within the specified rectangle must be updated. Otherwise the server may send only subregions covering those parts of the framebuffer that contain modified pixels. The server can either send exactly the requested area (same as non-incremental), the area specified by the outer bounds of all modified pixels within the requested rectangle, or a set of adequate rectangular subregions.

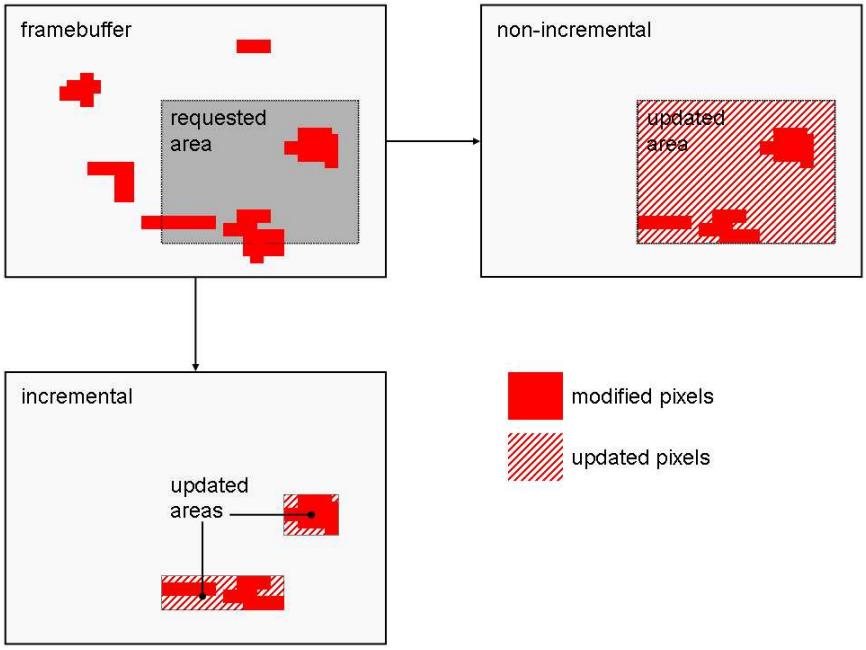


Fig. 3.1. Incremental and non-incremental framebuffer updates

Figure 3.1 shows an example of a framebuffer with some modified pixels (top left). The area requested by the client is marked by a rectangle. The non-incremental update (left) contains many pixels that have not been modified (since the last update). For the incremental request, the server has chosen two rectangular areas (bottom), which contain all modified and only a few unnecessary pixels. It would have been valid to transfer modified pixels only, but that would have demanded at least seven

rectangles and thus possibly resulted in an overhead caused by rectangle headers. The granularity of partitioning is a trade-off between processing time and the achieved compression ratio and varies among several server implementations.

3.2.1 Common VNC workflow

The RFB protocol specification [Richardson, 2005] does not exactly designate how to use messages. It demands that the communication must start with the initial handshaking phase and proceed to the normal interaction stage afterwards. At this stage, the client can theoretically send whichever messages it wants, and may receive messages from the server as a result. Practically the common VNC client and server implementations follow a certain workflow (see Figure 3.2). At first, the client

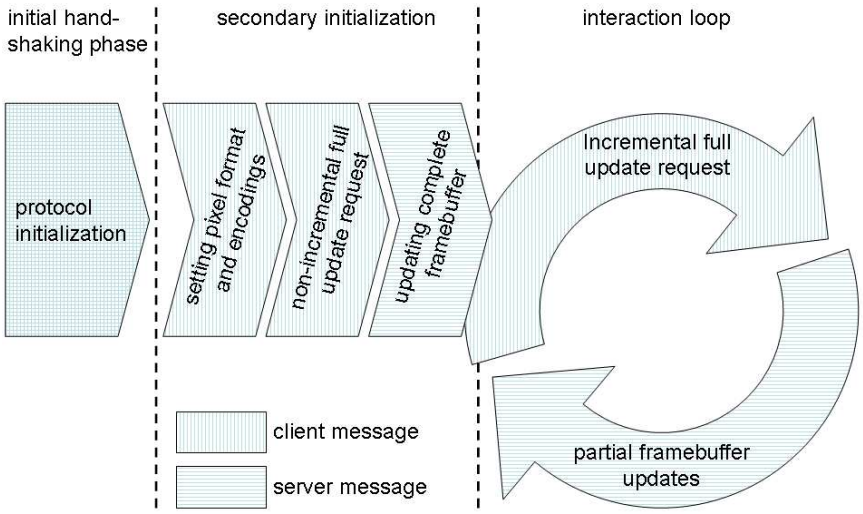


Fig. 3.2. Workflow of common VNC implementations

overrides the server's pixel format settings and specifies which encodings are accepted by sending a *SetPixelFormat* and a *SetEncodings* message, respectively. This is done before any other message is sent, and in particular before any framebuffer update request. Setting properties before any pixel data is interchanged assures that the client receives only pixel data it can decode, which is crucial because messages that cannot be parsed or messages of unknown types cannot be skipped due to missing message delimiters. As the next step in the workflow, the client requests a non-incremental update of the complete framebuffer by sending an appropriate *FramebufferUpdateRequest*. The server responds by encoding all pixel values of its framebuffer and transferring the resulting *FramebufferUpdate* message to the client, which allows the client to initialize its copy of the framebuffer. Now the client's and server's framebuffers are synchronized. This can be seen as second or intermediate initialization following the protocol's compulsory initial handshaking phase.

Afterwards the workflow enters an infinite loop of client's requests and according updates from the server. Again the client requests the complete framebuffer to be updated, but now in an incremental way. Thus the server has the option to send smaller rectangular parts only, which are sufficient to keep the client's copy of the framebuffer up to date. If no pixel values have been altered since the last update, the response is delayed until a modification of the server's framebuffer occurs. After the client has received, decoded and displayed a framebuffer update, the next iteration of the loop is executed, starting with another incremental update request of the complete framebuffer. So to speak, the common implementations make use of full area update requests only, one initial non-incremental request and always incremental ones thereafter. During the loop, input events (*KeyEvent* and *PointerEvent*) are sent whenever they occur.

3.2.2 RFB Message Types

The RFB protocol distinguishes *client to server* and *server to client* messages. All messages begin with a message-type byte, followed by any message-specific data.

Client-to-server messages

The protocol specifies six client to server message types:

type	message name
0	SetPixelFormat
2	SetEncodings
3	FramebufferUpdateRequest
4	KeyEvent
5	PointerEvent
6	ClientCutText

A client can set its properties by sending messages of the type *SetPixelFormat* and *SetEncodings*, which specify color depth and other format issues and a list of accepted encodings, respectively. As those mainly occur immediately after the handshaking phase, they can be seen as part of the initialization.

*FramebufferUpdateRequest*s can be incremental or non-incremental and specify the bounds (coordinates, width and height) of a rectangular area of the framebuffer, which is supposed to be updated by the server.

KeyEvent and *PointerEvent* forward user input events to the server. A *KeyEvent* consists of a key symbol and a flag which specifies whether a key was pressed or released. A *PointerEvent* delivers the actual coordinates of the pointer and a mask representing which buttons are currently pressed. Such events are used for both pointer movements and button presses. A movement results in events with different coordinates, but without any change of the button mask. A button press and release is indicated by two consecutive events with the corresponding flag of the mask once set and once not. If the pointing device has not been moved in between, the

coordinates stay unchanged, otherwise the two messages indicate the button press and the movement. For instance, the two consecutive events PointerEvent[(100,50) with button 1 pressed] and PointerEvent[(102,55) with button 1 released] indicate that the button 1 was pressed and released but the mouse was also moved from the coordinates (100,50) to (102,55), which could either be counted as *drag-and-drop* (commonly *drag-and-drop* results in more position changes in between) or just a little inaccuracy while pressing the button.

Finally, there are *ClientCutText* messages, which transfer text from the client's clipboard to the server.

Server-to-client messages

In the opposite direction four server to client message types are available:

type	message name
0	FramebufferUpdate
1	SetColourMapEntries
2	Bell
3	ServerCutText

The *ServerCutText* message type is the counterpart of *ClientCutText* and is used to transfer the server's clipboard contents to the client. The *Bell* message type is very simple without any content other than the type byte and is sent to ring the system bell as user notification. A *SetColourMapEntries* message contains a sequence of color values to specify the color mapping to be applied to pixel values (not supported by some VNC implementations).

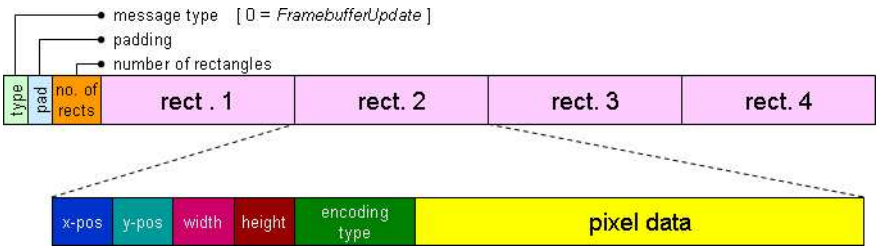


Fig. 3.3. Format of the *FramebufferUpdate* message type (including rectangles)

The most frequently used and most important message type is the *FramebufferUpdate* (Figure 3.3). An update messages is sent in response to a *FramebufferUpdateRequest* from a client. Each update consists of a number, which specifies how many rectangles are included, and a sequence of that many rectangles, all of which contain encoded pixel data supposed to be copied into the client's framebuffer. The common part of all rectangles specifies the (x, y)-position where the pixel data is to be placed, its width and height, and the applied encoding scheme. It is followed by pixel data encoded by the given scheme. As there are no delimiters provided by

the protocol, rectangles cannot be accessed individually, but only sequentially (by parsing all previous rectangles).

3.2.3 Pixel Format

A *pixel format* describes how pixel values represent individual colors. For *true color* formats bit-fields within the pixel value translate directly to red, green and blue intensities. Alternatively, a *color map* can be used, where an arbitrary mapping is applied to translate between pixel values and red, green and blue intensities. Further format parameters are *byte order* (little or big endian), *color depth* and *bits-per-pixel*. The distinction between color depth and bits-per-pixel may not be evident. In most cases they are equal, however it is common to store 24-bit color values within 32-bit fields to respect 4-byte borders.

3.2.4 Encoding Schemes

An *encoding scheme* specifies the compression algorithm that is applied to encode pixel data. The selected scheme influences various parameters such as network bandwidth, client drawing speed and server processing speed. The *Raw encoding*, that simply consists of *width*×*height* pixel values in left-to-right scanline order, can be processed very fast by both server and client, but obviously results in heavy bandwidth usage due to the lack of compression. This makes raw encoding suitable for local connections to the same machine with practically unlimited bandwidth, but unusable for slow modem connections. Advanced encoding schemes like *tight*[2] or *ZRLE*[3] offer good compression rates, but demand more processing power, which might be unsuitable for a VNC viewer running on a slow PDA. Except for the JPEG option of the *tight* encoding, all encodings used by the RFB protocol provide *lossless compression* of pixel values and therefore offer optimal image quality.

Several RFB encoding schemes use a two-dimensional extension of *run-length encoding* (RLE), called *rise-and-run-length* (RRE). The run-length encoding stores sequences of the same data value as a single data value and a count giving the length of that sequence. In its two-dimensional counterpart the count is substituted by a rectangle specifying a subregion of the data array to be filled with the given data value. Such an encoding is most effective for data that contains many regions filled with the same value, which is the case for most graphical user interfaces as they are mainly assembled by simple graphic images. The basic idea behind RRE is the partitioning of a rectangle of pixel data into rectangular subregions (subrectangles) each of which consists of pixels of a single value and the union of which comprises the original rectangular region. The *CoRRE*[4], *Hextile* and *ZRLE encoding schemes* are variations of RLE or RRE.

[2] special encoding used by the TightVNC implementation [TightVNC, 2006]
[3] ZRLE = Zlib Run-Length Encoding
[4] CoRRE = Compact Rise-and-Run-length Encoding

3.2.5 Hextile Encoding

The *Hextile encoding* is a variation of the RRE encoding and is of special interest for this work. A rectangle to be encoded is split into smaller subrectangles, called *tiles*. Limiting tiles to 16×16 pixels allows the dimensions of each subrectangle within a tile to be specified by 4 bits per value, 16 bits in total ((x, y)-position, width and height). Partitioning a rectangle into tiles is done in a predetermined way and therefore no position or size of each tile has to be explicitly specified. The encoded contents of the tiles simply follow one another starting at the top left going in left-to-right, top-to-bottom order. If the width of the entire rectangle is not an exact multiple of 16, then the width of the last tile in each row will be correspondingly smaller. Similarly, if the height of the entire rectangle is not an exact multiple of 16, then the height of each tile in the final row will also be smaller. An example of hextile ordering and sizes is shown in Figure 3.4. A framebuffer update of 93×68 pixel values is partitioned to 30 hextiles. The hextiles of the last column and the last row are smaller than 16×16 pixels.

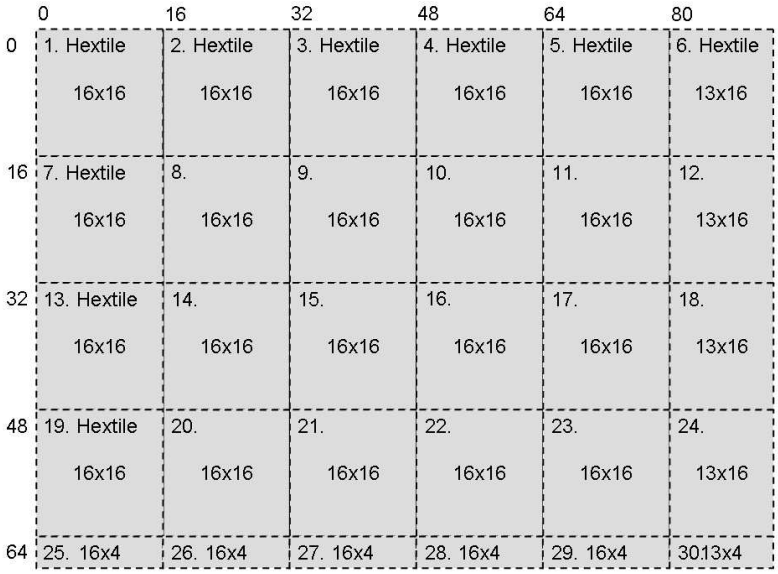


Fig. 3.4. Hextile partitioning and ordering of a rectangle of 93 × 68 pixels

Each tile begins with a *subencoding type mask* and is either encoded as raw pixel data or as a variation of RRE. If the *raw* flag is set, all other flags are irrelevant and pixel values follow in left-to-right scanline order. Otherwise each tile will — depending on the given flags — specify a background pixel value and/or a foreground pixel value for the entire tile. If any value is not explicitly specified for a given tile, the appropriate value of the previous tile is carried over. Tiles without any subrectangles are just solid background color. Otherwise a sequence of subrectangles is attached, which all are individually colored or filled with the foreground of the tile. Figure 3.5 shows a single hextile. As this hextile is two-colored, it is sufficient to specify one

foreground color for all subrectangles. The contained pixels can be encoded by specifying four subrectangles (given as $\{(x, y), width, heigh\}$): $\{(1, 1), 6, 2\}$, $\{(3, 3), 2, 8\}$, $\{(8, 3), 2, 10\}$ and $\{(10, 11), 4, 2\}$.

Predetermined tile ordering, 4 bit dimension values and carrying over previous colors result in a very compact compression.

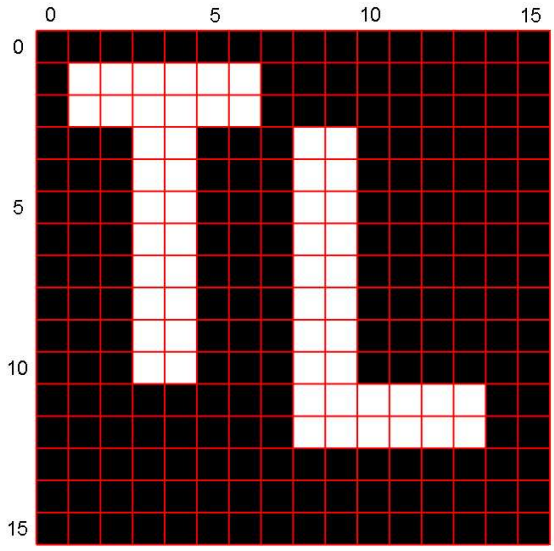


Fig. 3.5. Hextile with subrectlangles

3.2.6 Limitations

Accessing a desktop on the *framebuffer level* is not only independent of the operating and windowing system but furthermore is independent of the applications running on the accessed desktop. Hence, any application and thus also any documents can be presented regardless of the document format. Due to the *lossless compression* a perfect reproduction of the presented material is ascertained. However, *lossless compression* cannot achieve as good *compression ratios* as lossy encodings. Nevertheless, rather good *compression ratios* can be achieved, especially for slide presentation. However, there are a few suggestions regarding how to improve the *compression rates* or more precisely, what should be avoided (if possible) in a VNC session. In particular, elements that should be avoided are:

- color cycling (as shown in Figure 3.6, left);
- fine grained patterns (same figure, right);
- high colored high resolution images;
- large movies (e.g. scaled to fullscreen).

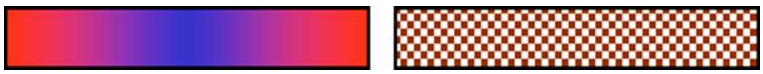


Fig. 3.6. Color cycling and fine grained patters

Note that it is unproblematic if small areas of the screen contain any of those elements, but the user should avoid to place a high resolution image as wallpaper of the desktop, presenting slides with a slide background that makes use of color cycling or presenting movies scaled to fullscreen. Solid coloring should be preferred whenever possible.

3.3 Distance Learning based upon VNC

In the final section of this chapter, we evaluate the suitability of the *Virtual Network Computing* system and the *Remote Framebuffer Protocol* as basis for creating a distance learning environment. We compare VNC's features with our criteria catalog (Section 2.4.1) and expose flaws and drawbacks, which must be eliminated or circumvented in order to produce synchronous and/or asynchronous electronic lectures.

Flexibility

VNC offers remote access to a graphical desktop and thus the desktop's content. Since desktop grabbing is done on a framebuffer basis, VNC is very flexible regarding presentation content. There is no restriction in the choice of applications that can be used, which are for educational issues the teacher's favorite presentation software, webbrowsers, additional visualization tools or whatever software is needed for a course. Furthermore, any dynamics of applications are preserved as well as pointer movements. Any application dependent annotations, which are visible on screen during lecture, are also preserved dynamically.

Display Quality

Most image and video compression methods are lossy to achieve smaller file sizes or lower bandwidth, thus making them unsuitable to compress the visual appearance of graphical desktops. In contrast, the encoding schemes of the RFB protocol are particularly designed to compress desktop data, which makes VNC a perfect choice regarding image quality. Providing the same color depth for server and client, all encodings (except *tight*) are lossless. The loss caused by downscaling to a lower color depth, e.g. from 24 to 16, is almost negligible, as in most cases it does not influence details in the displayed text, figures or sketches. Color cycling will appear less smooth using lower bit rates, but is mainly used as a background effect and not for pedagogical reasons. Due to resulting in less efficient compression ratios, the usage of color cycling effects is discouraged anyway. The same holds for high colored images used for the background of slides or as desktop wallpaper.

Recording and Replay

Although the RFB protocol is not originally intended for recording, it can easily be extended for that purpose. [Li and Hopper, 1998b] introduced a VNC session recorder, which is seamlessly integrated as proxy between a VNC server and a VNC viewer. The proxy logs RFB messages plus additional timestamps specifying the delay since the beginning of the recording. Afterwards the messages can be sequentially replayed at the same rate as when they were recorded. Further publications of the VNC development team address indexing [Li et al., 2000a] and retrieval [Li et al., 1999b], but by no means fulfilling today's requirements. Advanced playback features like slide based navigation or full text search remain unconsidered.

Synchronous Scenarios

VNC is designed for remote desktop access in a scenario where a single user interacts with applications running on a server via a network. However, a second user (a student), whose client is connected to the same VNC server, is able to observe any work done by the first one (the teacher). The student can survey any pointer movement, any button press, any menu selection and watch how the graphical user interface responds to the interactions made by the teacher. If allowed, the student himself/herself can even interact with applications under the guidance of the teacher. Providing an audio channel between those two participants, a one-on-one distance teaching environment is set-up.

While a VNC server can supply several clients with a copy of its framebuffer, it is not a very scalable infrastructure. In order to respect different properties and processing speeds, each client must be fed separately with an individually transformed and encoded copy of the same framebuffer. This is not a severe restriction regarding the intention of VNC to provide remote desktop access, because many users interacting with the same desktop at the same time is not very reasonable. However, in the context of distance learning a scalable infrastructure is demanded. Extend the scenario described above to a virtual classroom, where the teacher's desktop should be visible via network for many students. Controlling access is available to the teacher exclusively and students participate on a course by using view-only clients connected to the teacher's desktop. A VNC server might be able to handle up to 20 clients, but is unable to supply hundreds of students, because this would cause heavy network traffic and a high processing load to encode and transfer all pixel data.

Transparent Integration

The VNC technology can be seamlessly integrated into a computer based presentation scenario by starting a VNC server, which exports the presentation computers' graphical desktop. Every RFB compatible client is able to access the graphical output now, either for recording asynchronous electronic lectures or transmitting synchronous ones. The teaching process stays uninfluenced by the VNC server running in the background. VNC implementations for operating systems that support more than one graphical desktop may not export the standard desktop, but rather run an additional one. Such a background desktop can be accessed via a VNC client locally or remotely. The appearance of the VNC client can be optimized by setting

it to fullscreen mode. Thus, no additional borders create a distraction and the VNC desktop can be accessed like a standard desktop.

Due to the variety of VNC implementations available for all common operating systems, the described scenarios can be integrated in a cross-platform manner into existing heterogeneous infrastructures (common not only in the technical departments of third level education) and thus even respecting low budgets, as VNC is freely available.

3.3.1 Summary

Regarding the suitability of VNC as a basis for setting up a synchronous and asynchronous e-learning scenario, this technique offers a flexible high quality access to any remote desktop on a framebuffer basis, allowing a teacher to use any applications (including any presentation software) running on arbitrary operating systems during lectures or other courses. Thus, VNC is very flexible regarding presentation content (Criterion C3) and dynamic annotations (C4b). The RFB protocol (C9d) offers lossless (C9a) and efficient (C9e) compression of a graphical desktop. Due to the availability of free VNC implementations and the cross-platform design, VNC is applicable in a platform independent and cross platform manner (C10). The scenario can be integrated into any heterogeneous infrastructure and this even in a cost-effective and seamless way, obeying the Lightweight and Transparency strategies.

However, any support for handling audio and/or video streams is missing (Criteria C1 and C2). At least audio is essential, whether to record and playback the teacher's voice, to allow a one-way live transmission or to establish a communication between multiple participants. [Li and Hopper, 1998a] suggested the use of an additional telephone line. Furthermore, video conferencing software, such as the *Robust Audio Tool* (RAT) [RAT, 2006], could transmit the verbal narration. However, an integrated solution is rather preferable, because too many separate applications may confuse users [Effelsberg and Geyer, 1998].

As numerous protocols and encodings have been designed to record or transmit audio and video data it should be possible to find suitable ones, which can be combined with or integrated into VNC to fulfill the given requirements. Since accessing, transmitting and recording a VNC desktop is not affected by a particular way of integrating audio and video streams, we will discuss this topic later (while addressing implementation issues) in Section 9.6.

A more severe restriction to building up a distributed virtual classroom scenario is the very limited number of simultaneous users that a VNC server can handle. Chapter 4 describes how the VNC environment and its RFB protocol can be modified and extended to widen its scalability in a way that fulfills the requirements of setting up a virtual classroom environment (C11).

Furthermore we have to address VNC's recording facilities. Logging messages and the time of their occurrences allows VNC sessions to be recorded. Unfortunately,

playback is rather limited to sequential replay of these logs. Instead, we are interested in advanced navigational (C7) and retrieval (C8) features. How to achieve such features for pixel-based VNC recordings is discussed in Chapters 7 and 8. As advanced playback can benefit from suitable recording formats, we will describe in Chapter 5 how VNC session recording can be improved.

4

Scalable VNC

The *Virtual Network Computing* (VNC) architecture [Richardson et al., 1998] is designed and commonly used to access a remote desktop, which is provided by the VNC server, from a single distant VNC client. However, several clients can be connected to the same server. In this case, the desktop is shared among the clients, which means that each client will display and, unless set to the view-only mode, also access the same remote desktop. A client can request to be connected in the shared mode and, if granted by the server, all other clients that are connected in parallel stay connected as well. If a client requests exclusive access to the desktop, the server can either disconnect all other clients or reject the request and thus the incoming connection. As the client's properties are often set to request exclusive access by default, although this may not be explicitly demanded by its user, it may be advisable to set the server's always shared option (offered by most implementations). If this option is enabled, all incoming connections will be treated as shared, and thus not disconnect any existing connections, regardless of whether the connecting VNC client requests a shared or an exclusive access. It is a critical issue how to handle incoming connections in our intention of providing electronic lectures. Consider a VNC session recorder implemented as VNC client connected to a VNC server whose always shared option is disabled. The recording process will be terminated whenever another client connects, but its user has forgotten to set the shared option.

Furthermore, the latest VNC server implementations support two kinds of connections, one for view-only clients and one for full access. This is useful for distributed classroom scenarios. As clients are distinguished by different passwords, the view-only password can be handed to students without offering them access to the teacher's desktop. Hence, a VNC desktop presentation can be easily distributed to the world wide web, if combined with verbal narration, for instance by use of video conferencing software (such as *Robust Audio Tool* (RAT) [RAT, 2006] and *Videoconferencing Tool* (VIC) [VIC, 2006]). However, we rather prefer an integrated software solution instead of starting one application that displays a remote desktop and one to receive audio and maybe an additional one to process a video.

Although the VNC architecture allows multiple clients to share the same desktop, it is not very scalable, because all clients are served individually and therefore only a limited number of clients can be supplied. This is for two reasons:

1. **Point-to-Point Communication:** Existing VNC implementations are based on connection-oriented communication, which only allows one-to-one transmissions. In order to supply 100 clients with an update of 100 kbytes the server's outgoing traffic will increase to 10.000 kbytes. For a distance learning environment supplying a large number of students, a more suitable one-to-many or many-to-many communication is preferable. Furthermore, the number of simultaneous connections of the server machine may be limited by system properties.

2. **Individual Client Properties:** During initialization, each client specifies its pixel format and the set of encodings it can handle. The server must transform and encode each framebuffer update for each connection according to the properties of the respective client. Obviously, if many clients are connected simultaneously, individually supplying each client requires huge server processing power due to many parallel transforming and encoding tasks.

In order to circumvent the high processing needs a network could be designed, where clients do not directly connect to the teacher's VNC server, but rather communicate with proxies [Li and Hopper, 1998a, Li et al., 1999b, Li et al., 2000b]. One could create a scenario where the main VNC server supplies a manageable number of proxies, which are connected as VNC clients. These proxies act as (intermediate) VNC servers and distribute (a copy of) the framebuffer to the students' clients. Parallel proxies could handle more clients and a multi-staged cascade of such proxies could be established if necessary, but would cause a little extra delay for each level. The processing power necessary to individually serve all clients could be distributed among the proxies. In order to use existing software components instead of newly implementing the proxy, a VNC viewer running on a VNC server could be used. However, besides the huge amount of hardware needed and the complicated process to build-up such a network, the benefits regarding the bandwidth consumption are limited to the data exchange between two points on such a network. Nevertheless, a bandwidth problem will occur when all proxies are located within the same network segment, where all outgoing connections are handled by a single router, which therefore becomes the bottleneck.

[Li et al., 2000b] introduced a one-to-many communication for the VNC environment by seamlessly placing a proxy between the server and the clients to intercept and manipulate the message streams. The proxy merges and transmits the messages of all clients to the server. Unfortunately, clients' requests are merged in a way that the proxy waits until it receives one request from each client before forwarding a single (merged) request to the server. This ensures the rate of requests to be that of the slowest client, but unfortunately delays the communications for all other clients. Furthermore, that proxy forwards all framebuffer updates received by the VNC server to all clients. From the VNC server's point of view, there is only one single client, which is the proxy.

[Li et al., 2000b] experimented with four VNC clients that were connected via a proxy to one VNC server. The proxy forwarded server messages to multiple clients via multiple point-to-point connections. They measured the average data transmission rate for each client while performing certain tasks. During the tests one to four clients were connected simultaneously. This exposed a drop of the average data

transmission rate of up to 50% when four clients were connected instead of one. Furthermore, they noticed that the completion time of the task which caused the most framebuffer updates, increased from about 90 to 140 seconds due to "an unnecessary overload on some segments of the network that have to carry multiple identical flows [... and because ...] more clients usually cause more congestion and delay" [Li et al., 2000b]. One has to admit that the bad performance was not only caused by the number of clients, but also caused by the unnecessarily high bandwidth usage due to sending uncompressed framebuffer updates (*Raw* encoding) and the process of merging requests, which waits for the slowest client. Nevertheless, their solution does not fulfill our demands of scalability in order to support a large number of students. In fact, a better scaling infrastructure is needed in order to supply many clients with synchronous electronic lectures.

However, the idea of *extending the VNC architecture* by placing a *proxy* between existing VNC server and client implementations or implementing a new client (in fact a proxy is a client from the server's point of view) is preferable in comparison to modifying or re-implementing a *VNC server*. Due to the *thin client* ideology of the *VNC architecture*, it is much easier to implement a *client* (perhaps in a platform independent programming language). Furthermore, the dependencies of the server towards the windowing and operating system are unlike higher than the dependencies of the *thin client*. Hence, modifying the server's functionality while providing a *platform independent solution* requires to adapt all the implementations of several operating systems and do so, not only once, but for each future release. Implementing a platform independent *thin client* or modifying one of the open source clients preserving *compatibility with original VNC implementations* allows our components to be *integrated into standard VNC environments*. In fact, the development of the *TeleTeachingTool* was originally started by adapting and extending a platform independent Java implementation of a VNC client.

In order to improve the scalability of the VNC architecture to supply many students with synchronous electronic lectures, first we discuss how to limit individual handling of clients and merge the communication of parallel connections with the aim to reduce the processing load of the server. Secondly, we expose the requirements and restrictions of a switch from the originally used one-to-one to a more suitable one-to-many communication and explain how to adapt the VNC environment and its RFB protocol to work in a scalable one-to-many fashion.

4.1 Limiting Individual Properties

One obstacle preventing scalability is that, although all clients are displaying a (transformed) copy of the same framebuffer, all updates must be individually transformed, encoded and sent for each existing client according to its individual properties regarding the requested pixel format and which encodings are supported. Such individual handling of clients obviously requires huge server processing power due to many parallel transforming and encoding tasks if many clients are connected simultaneously. Considering that most VNC viewer implementations are very similar (or users even use the same implementation), the diversity of parameters used

practically is not as high as offered by the *Remote Framebuffer* (RFB) protocol [Richardson, 2005].

Recall the workflow of common VNC implementations (Figure 3.2) with its initial handshaking, secondary initialization and the infinite interaction loop of non-incremental update requests and responding framebuffer updates. Omitting the possibility of setting individual client's properties and presuming that all clients have similar preconditions regarding their processing power, displaying capabilities and network connections, nearly identical framebuffer updates are sent almost simultaneously. Having equal clients, which use the same pixel format and consume server messages at almost identical rates, a framebuffer update computed as a response to an update request from any client can be re-used for other clients, which presumably will send an identical request shortly after. This is especially the case as each client will always set the same update request during the interaction loop, which is in particular the request to update the complete framebuffer in an incremental fashion. Moreover, even all other clients will set the same request during their interaction loop. However, the high quantity of different pixel formats provided by the RFB protocol may prevent the use of such shared framebuffer updates, maybe only because two pixel formats differ in just one little parameter, which is most probably even insignificant.

4.1.1 Reducing Pixel Formats

The RFB protocol offers several parameters in order to individually adjust the pixel format. Besides setting the byte order (big-endian or little-endian), the RFB protocol allows not only the use of several color depths, but also the specification of how to extract red, green and blue intensities from a pixel value by giving the number of bits used for each color. This is done by a $max\ value = 2^n - 1$ where n is the number of bits used for the color and a $shift\ value$, which is the number of shifts needed to get the appropriate intensity in a pixel to the least significant bit. For 16 bit pixel values and without concerning the byte order, this already results in $\binom{18}{2} = 153$ different pixel formats[1]. Altogether for the common 8, 16 and 24 bit color depths and regarding both byte orders, the RFB protocol offers $\binom{10}{2} + 2 * \binom{18}{2} + 2 * \binom{26}{2} = 1001$ different pixel formats[2].

Not all pixel formats are necessarily reasonable. The commonly used RGB color model[3] provides no more than 8 bits per color, 24 bits in total. Hence, it is useless to specify more than 8 bits per color, because the finer nuances are not displayable. Moreover, imbalanced pixel formats are not very meaningful. Consider, for instance, the format for 8 bit pixel values, which specifies 7 bits for the red intensities, but only a single bit for green and even no bit for blue values. This format allows the brightness of red to be specified in $2^7 = 128$ different steps and also supports bright green and yellow tones, but no blue, brown, pink, violet, orange or turquoise colors,

[1] number of 3 partions of 16 bits - 16 white balls as bits and 2 black ones as markers

[2] byte order is irrelevant for 8 bit values

[3] the RGB color model is an additive model, which combines its primary colors Red, Green and Blue to reproduce colors

as displayed in Figures 4.1 and 4.2. In fact, reasonable formats will provide almost uniformly distributed numbers of bits for the red, green and blue color intensities.

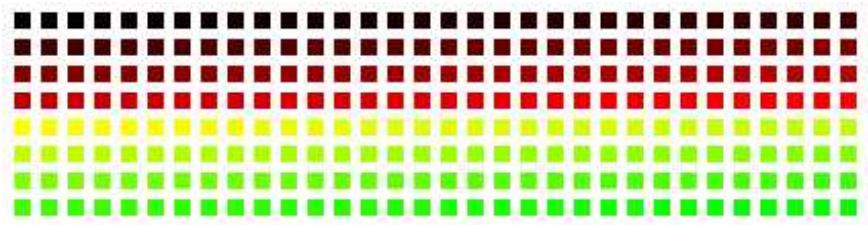


Fig. 4.1. All colors of the 7-1-0-bit RGB format

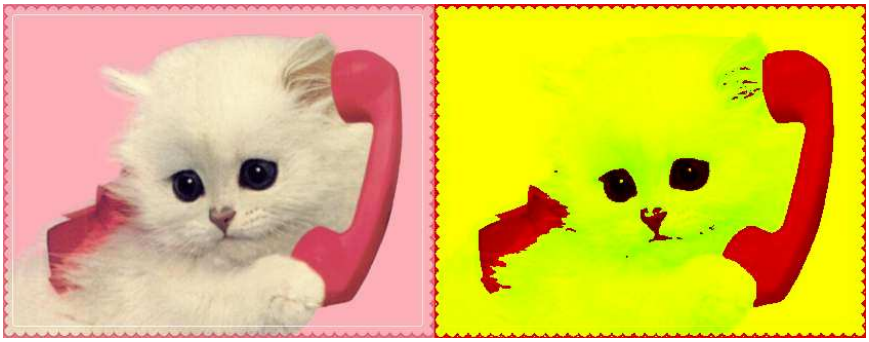


Fig. 4.2. True color vs. 7-1-0-bit RGB format

In consequence, applying one predetermined byte order and (almost) uniformly distributed color bits, no more than three different pixel formats are needed, one for each of the three standard color depths (see Figure 4.3).

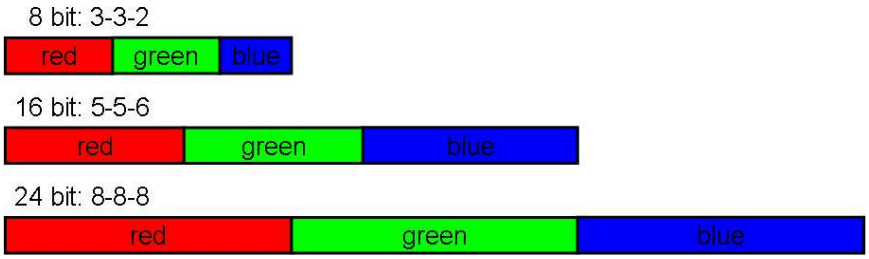


Fig. 4.3. RGB formats with uniformly distributed bits

4.1.2 Encoding Agreement

Besides sharing a single pixel format, obviously all clients must support the same encodings as well. However, the number of possible encodings is considerably lower than the number of pixel formats. The RFB protocol specification in its version 3.8 [Richardson, 2005] specifies six encodings for framebuffer updates (*Raw*, *CopyRect*, *RRE*, *CoRRE*, *Hextile* and *ZRLE*), even including a deprecated one (*CoRRE*), and two so-called pseudo-encodings (*Cursor* and *DesktopSize*), which encode the mouse cursor or switches to another desktop resolution. These are the encodings specified by the RealVNC team, which officially releases the RFB protocol specifications. Furthermore, three additional encodings (*zlib*, *tight* and *zlibhex*) and several pseudo-encodings are registered (created by developers of other VNC implementations).

In the original VNC workflow each client specifies which encodings it supports. In order to supply all clients with the same update messages, we must either *statically* ensure that all clients support the same set of encodings or the framework must *dynamically* agree on the least common subset during communication. A dynamic agreement can be achieved by seamlessly placing a proxy between the VNC server and its clients. The proxy suggested by [Li et al., 2000b] filters all *SetEncodings* messages and thus implicitly agrees on the *Raw* encoding, which must be supported by all clients regarding the protocol specification. Although not explicitly mentioned, obviously their clients must also agree to a single pixel format as we have suggested in the previous section.

Unfortunately, agreeing to the *Raw* encoding is very inefficient because it provides no compression of the pixel data. Rather the proxy should coalesce all incoming *SetEncodings* messages and send an appropriate *SetEncodings* message to the VNC server whenever a client connects which does not support all of the currently specified encodings. Such a *dynamic agreement* will serve fine unless the intersection of the encodings that are supported by individual clients is empty, which should rarely occur if using the same or similar VNC implementations with default settings.

Recalling our intention to create an e-learning environment with integrated verbal narration, we probably have to provide our students with a special purpose client implementation in order to support audio streams. Hence, we can additionally require this client implementation to support certain encoding schemas and thus the static agreement fits fine.

4.1.3 Combining Framebuffer Update Requests

The proxy provided by [Li et al., 2000b] merges the update requests of all clients to a single one. Unfortunately, the faster clients have to wait for the slower ones. This causes a lower rate of requests and thus fewer updates and a less smooth replay. Furthermore, if a single client freezes, the entire communication between the server and all other clients will be blocked due to missing a single request.

If the suggested proxy would forward all requests of all clients to the server, the higher number of requests could possibly cause a deadlock if the server implementation relies

on the common workflow of alternating requests and responses. But this should rarely be the case, since the protocol specification states that a single update may be sent in reply to several requests. However, the higher number of requests would increase the rate at which the server creates framebuffer updates. If ten clients request their framebuffer to be updated and all ten requests are forwarded to the VNC server, the server will read and answer the first request. As the other requests are buffered the server is free to immediately send nine further updates instead of waiting until the clients have received, decoded and displayed the first framebuffer update message.

One could also imagine a scenerio where the proxy generates a request after each forwarded framebuffer update message, or a scenario of a VNC server sending updates whenever its framebuffer is modified without waiting for a request. We have tested the first scenario and spotted that updates were sent faster than a real VNC client is able to consume and display them. This was the case whenever many large updates occurred within a short time span. For instance, we dragged a window for several seconds in circles over the desktop. The client received, decoded and displayed all framebuffer updates, but at a lower rate than they were generated by the server. In consequence, the surveyed window movement was much slower on the desktop that was displayed by the client. Then we stopped the dragging of the window and applied actions that caused fewer modifications to the framebuffer, for instance just moving the pointer or entering some text in a shell. As the client still was occupied with the decoding of the previously sent messages caused by dragging the window, the later updates were also delayed. As soon as the client had processed all updates caused by the dragging, it started displaying the later updates. However, as processing those smaller framebuffer updates was less CPU intensive, they where displayed at a faster rate than they actually occurred, which looked unexpectedly strange. The appearance was similar to dragging an elastic band. Pulling causes the band to flex and therefore pulling becomes slower the wider the band is stretched, but if the band tears, the movement will be suddenly and surprisingly fast. Therefore, it is advisable to set requests that are generated by a real VNC client, which means a client that really decodes and displays all framebuffer updates.

Neither forwarding all requests, sending updates without requests, nor merging requests of all clients (by waiting for one request per client) is a satisfying solution. Presuming that all clients are served with (copies of) the same framebuffer update messages, it is sufficient if only one client, in our classroom scenario supposedly the teacher's client, generates requests and the response will immediately be delivered to all other clients as well. Even if a client does not process the incoming data immediately for some reason, for example temporary network dropouts or being busy running other tasks, it will be in sync shortly after, because naturally the communication shows a lot of idle time, which enables the clients to process buffered messages. Note that this will not lead to the "elastic band" effect described above, because the frame rate will never decrease due to too much framebuffer updates. It rather gives to impression that the computer has stuck, which is (sadly) common to most users. The faster replay of buffered messages is relieving since it offers the impression the the blockade is now removed and the computer is working again.

Regarding the given facts, we suggest the requests are generated by a single client, which really displays the framebuffer updates, preferably the teacher's client, and the

resulting update messages are forwarded to all other clients. Alternatively, the proxy can ignore the requests of all clients and create framebuffer update requests on its own, but must respect the time needed by clients to decode and display framebuffer updates in order not to increase the rate of updates. Such a scenario can be useful considering a proxy that records the current VNC session without any connected client.

To summarize, in order to increase the degree of scalability by reducing transforming and encoding tasks and supplying all (or at least many) clients with (a copy of) the same framebuffer update, the clients must support predetermined properties instead of individual ones. This can be achieved limiting the unnecessarily large number of possible pixel formats to an unitary standard or at least to three or four standardized formats (e.g. for different color depths). Furthermore, all clients must agree on a collective set of encodings, which has to be supported by each client. The agreement can be achieved either dynamically or statically. Besides omitting individual properties, individual requesting is eliminated as well. A single privileged (teacher) client or an appropriate proxy will generate the framebuffer update requests instead.

4.2 One-To-Many Communication

Current VNC implementations are based on a connection-oriented communication protocol that only offers one-to-one data transfer and therefore causes heavy bandwidth usage if supplying many clients in parallel. The experiments described by [Li et al., 2000b] revealed a drop in the average data transmission rate of up to 50% when four clients were connected instead of one. This is mainly caused due to serving multiple clients with uncompressed pixel data via multiple point-to-point connections. In order to achieve a higher degree of scalability, more suitable one-to-many transmission schemas are needed.

4.2.1 Routing Schemes

In computer networking the term *routing* refers to selecting paths in a computer network along which to send data. The most common routing scheme is *unicast*, where a source sends packets separately to each recipient, which is similar to an ordinary mail delivery or, for two-sided connection-oriented protocols, a phone call. In order to send the same data to multiple recipients, the source must send multiple copies and each copy must traverse the network separately between the source and destination, which obviously is very inefficient.

The opposite is *broadcast* where one sender reaches many recipients analogous to a television or radio transmission. For computer networks, broadcasts are highly inefficient, because the entire network is flooded with data packets reaching all possible destinations, regardless of whether anyone is interested or not. Therefore, broadcasts are generally filtered (ignored) by routers and hence will work in the subnet of the sender only.

An optimized variation of broadcast is *multicast*, where data is send from one or more senders to a set of registered recipients. Unlike unicast, multicast does not send multiple copies. In fact the data traverses the network on a path towards all recipients as long as possible and only if the path must be split up to reach all clients, one copy of the data is routed along each new fragment. This assures that every (sub-)path will never transmit two copies of the same data.

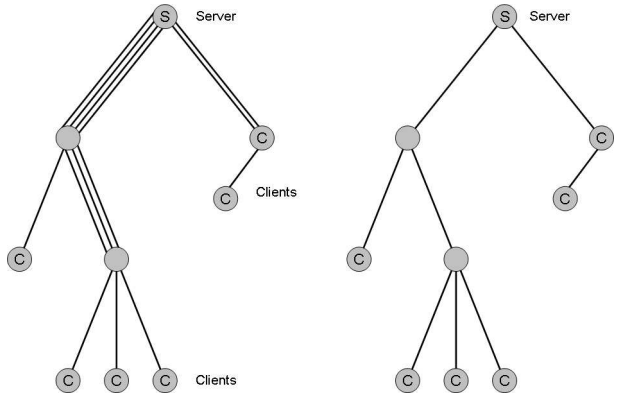


Fig. 4.4. Multiple point-to-point connections vs multicast connection

4.2.2 Communication Protocols

Although the RFB protocol defines distinct messages, the common VNC implementations are based on streaming data transfer. Client and server communicate via two reliable streams, one transfers the client-to-server messages, while the other handles the opposite direction. The streams buffer incoming messages until they are read by the corresponding consumer. The sender feeds the stream and the receiver consumes byte after byte while reading and parsing the incoming messages. A VNC server can start writing a framebuffer update message while encoding is still in progress and the receiving VNC client already starts displaying the update while decoding it, although it has not received the full message (maybe the server is still encoding!).

The *Transmission Control Protocol* (TCP) [Postel, 1981b] used by VNC implementations is a connection-oriented protocol offering the benefits of *reliable* and *in order* delivery of sender to receiver data for *stream oriented* services, but is relatively complex causing additional processing (e.g. for error detection and retransmission of lost packets). Unfortunately it only supports *one-to-many* data transfer. Also each client connection is established during initialization and then is held during the complete session, which might be a problem if the number of feasible simultaneous TCP connections of the server is limited.

For time-sensitive applications (such as video conferencing) TCP might not be appropriate as, due to in order packet delivery, the recipient cannot access the packets coming after a lost packet until the retransmitted copy of the lost packet is received.

For real-time applications it is more useful to get most of the data in a timely fashion than it is to get all of the data in order. Missing data may reduce the quality of the reproduced voice or motion picture, but delaying the replay while waiting for a single piece of data to arrive is unacceptable. Consider some short gaps in an audio transmission, which lead to a slightly distorted, but still understandable speaker. On the other hand, delays caused by ensuring in order packet delivery will probably result in a stumbling voice, which is hard to follow for the audience.

The *User Datagram Protocol* (UDP) [Postel, 1980] is a connectionless transport-layer protocol that does not provide the reliability and ordering guarantees that TCP does. Instead of building a stream, UDP transmits data by sending packets called *datagram*, which are sent in a best effort manner. Datagrams may arrive out of order or can be lost without notice. Without the overhead of checking if every packet actually arrived, UDP is faster and more efficient for many lightweight or time-sensitive purposes such as transmitting audio and video streams. UDP is not only more suitable for real time applications, but also supports the multicast routing scheme and thus the one-to-many and even many-to-many communication postulated for distance learning environments in order to enlarge scalability [Effelsberg and Geyer, 1998].

In order to enable multicast transmissions for VNC, a shift from TCP based reliable in order streaming communication towards unreliable packet oriented UDP transmission is inescapable. Therefore RFB messages must be packed into UDP datagrams instead of being delivered via streams. In consequence, we must respect the allowed maximum size of datagrams and regard the impacts of packet loss and out of order delivery

4.2.3 Size Limitations and Message Dependencies

The size of UDP datagrams is limited. The supported datagram size depends on system properties and therefore varies between different platforms. Our testings discovered an apparently "non deterministic" behavior regarding UDP packets larger than the platform's maximum size. Some systems dropped such packets as invalid, others just truncated them without any notification, but all systems supported at least packets up to 64 kbytes.

Just splitting the original message stream at arbitrary positions, for instance at 64 kbyte borders, to fill UDP packets will fail as soon as one packet is lost or received out of order, which can occur as UDP is a best effort and not a reliable protocol. Consider four RFB messages split into six datagrams as shown in Figure 4.5. A recipient that

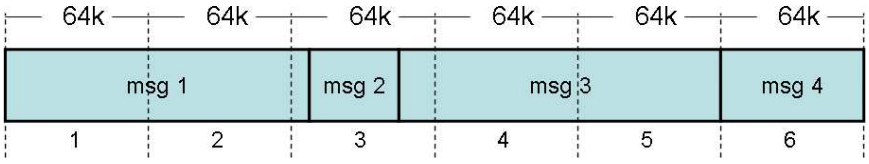


Fig. 4.5. Splitting messages to datagrams of 64 kbytes

has parsed datagram no. 1 and then receives no. 3 cannot finish message no. 1. Assuming that the datagram provides consecutive data of message no. 1, further parsing will fail in all probability. Numbering the datagrams by adding increasing sequence numbers allows the detection of packet loss. However, as it is impossible to determine the beginning of another message due to missing message delimiters, the recipient is even incapable of parsing message no. 2 although it is completely included in datagram no. 3. Datagram no. 6 is valid on its own as it exactly contains a single message. However, if the previous message is lost, it is almost impossible to detect whether the datagram begins with a message or not. Again the parser assumes the received data to be part of the unfinished previous message. The splitting of RFB messages to UDP datagrams will always be doomed whenever a datagram is lost due to missing message delimiters and size tags. In fact, each datagram must provide some information about which part of which message it contains. Regarding the possibility of packet loss it is even better if each datagram contains only complete messages, which are valid on their own, because this reduces datagram dependencies.

In order to ensure that UDP datagrams contain complete messages, it is necessary to determine the beginning and the end of each message. Messages of a fixed length, for instance the *KeyEvent*, *PointerEvent* or *Bell* messages, can be read in total once the message type is determined be reading the message tag. Detecting message borders of other than fixed size messages demands these messages to be completely parsed, because the RFB protocol neither explicitly specifies the length of the following data, nor does it offer delimiters to separate messages from each other. Unfortunately, this prevents the handling of complete messages and also skipping of unknown messages, which would enable a better extensibility of the RFB protocol. In the designed VNC context, lengths or delimiters are not needed as all messages are read and consumed in order. However, a proxy that only forwards messages must parse and buffer all messages to extract them from the incoming stream in order to pack messages into datagrams.

Packing RFB messages into UDP datagrams demands not only that messages are distinguished from each other, but also that the length of a message does not exceed the maximum datagram size, which we have limited to 64 kbytes as supported by all operating systems we have tested. The five fixed size message types have lengths of 1 to 20 bytes (see Table 4.1) and hence are unproblematic. A *SetEncodings* message

message type	**max. size** (in bytes)
SetPixelFormat	20
FramebufferUpdateRequest	10
KeyEvent	8
PointerEvent	6
Bell	1
ClientCutText	$8 + 2^{32}$, $\leq 64k$ if truncated
ServerCutText	$8 + 2^{32}$, $\leq 64k$ if truncated
SetEncodings	$4 + 4 * 2^{16}$, ≤ 1064 for registered encodings
SetColourMapEntries	$6 + 2^{16}$, rarely used (for many colors)
FramebufferUpdate	$> 64k$

Table 4.1. Maximum lengths of messages

can theoretically be $4 + 4 * number\ of\ encodings = 4 + 4 * 2^{16}$ bytes long. However, regarding that the specification lists nine registered encoding types and 256 values for pseudo-encodings, a maximum of 265 values and thus a maximum length of $4 + 4 * 265 = 1064$ bytes is practically relevant. Generally, a client will set only a few encodings and maybe some pseudo-encodings, which specify additional parameters for the encodings. *ClientCutText* and *ServerCutText* messages are used to transfer ISO 8859-1 (Latin-1) encoded ASCII text from a client's clipboard (or cut buffer) to that of the server or vice versa. The message length depends on the length of the text. As the protocol allows the length to be specified by 4 bytes, the theoretical length of text can be 2^{32} characters and therefore the length of a message can reach $8 + 2^{32}$ bytes. However, transferring that many text is not very meaningful and clipboards will rarely support such sizes. Limiting the length to 64 kbytes allows 65528 characters of text to be transferred, which should be sufficient. Hence, the text of a *Client-* or *ServerCutText* message will be truncated prior to packing them into an UDP datagram and the field that specifies the length of the following text must be adjusted accordingly. The *SetColourMapEntries* message type is rarely used. Most VNC server implementations do not support this message type at all. The theoretical maximum length is $6 + 6 * number\ of\ colors = 6 + 6 * 2^{16}$. However, defining that much colors is not very meaningful. An appropriate pixel format is able to cover (at least most of) the specified colors as well and should be used instead. Hence, most of the RFB messages fit into one UDP packet easily. However, framebuffer update messages may and most probably will exceed the maximum size of a datagram.

4.2.4 Splitting Framebuffer Updates

Framebuffer updates may be too large and will result in truncated messages if packed into UDP. The size restriction can be resolved by splitting those updates into smaller ones. In order to avoid dependencies, each of these pieces of the original update must be a valid RFB message of its own. All updates are built up of one or more rectangles. If an update message contains multiple rectangles it can be split into several updates containing only a subset of these rectangles. If a single rectangle is still too large to fit into an UDP packet as a whole, it must be split into smaller rectangles. Certainly it is always possible to decode the framebuffer update to raw pixel data and then (re-)encode smaller rectangles. Besides producing additional processing tasks, it is not straight forward to gain a suitable subdivision of the unencoded framebuffer. Rectangles of pixel data must be determined that will fit into datagrams after being encoded. Choosing very small rectangles will increase the overhead for message headers. A coarse-grained partitioning may result in messages that exceed the given maximum size again and therefore must be split again, which produces further processing loads.

Additionally it must be taken into account, that some framebuffer encodings depend on data earlier sent. So is the case for the *ZRLE* encoding, which stands for *Zlib Run-Length Encoding* and, as the name suggests, applies zlib compression [Deutsch and Gailly, 1996] to the pixel data. During one connection, all *ZRLE* encoded messages are compressed by use of a single zlib stream, which achieves a better compression ratio but unfortunately demands all *ZRLE* encoded data to be encoded

and decoded strictly in order [Richardson, 2005]. Decompression may require previously sent data and therefore may fail if any part of the zlib stream is missing, which may occur due to packet loss. The same is the case for the *zlib*, *tight* and *zlibhex* encodings. Therefore, encodings that apply zlib compression to data of sequential messages are unsuitable for unreliable UDP transmission.

Another encoding type that relies on previous data is the *CopyRect* encoding, which only specifies a rectangular area of the framebuffer to be copied to another position. Unlike encodings which are based on zlib streams, the *CopyRect* encoding will not fail if previous messages are lost. However, copying other than the expected content may lead to (even more) incorrect framebuffer content. In contrast, *Raw*, *RRE*, *CoRRE* and *Hextile* encoded framebuffer updates do not depend on previous messages or the current state of the framebuffer and thus are valid on their own.

Instead of decoding, subdividing and re-encoding large framebuffer updates, they can be split and repacked regarding their encoding scheme. A new message header must be generated for each part, but the data of the message can be reused by applying little adaption. The *Hextile* encoding is a suitable candidate.

4.2.5 Splitting Hextile Encoded Rectangles

Hextile encoded rectangles are subdivided into tiles of 16×16 pixels, which are encoded in left-to-right top-to-bottom order (see Section 3.2.5). As many rows of a height of 16 pixels are gathered as fit into one UDP datagram. Taking complete rows only ensures the result to be in a rectangular shape. The maximum size of each tile is limited by the size of a raw encoded tile, which consists of a one byte header followed by a sequence of pixel values for the 16×16 pixels: $1 + 16 * 16 * bpp$ (bytes per pixel). Assuming a color depth of 16 bit this are 513 bytes per tile resulting in a maximum of 127 tiles per 64 kbytes. This corresponds to a desktop width of 2032 pixels, which should be sufficient for common desktops. Even the width of 1008 pixels for 32 bit values should rarely cause problems, as the probability of reaching the maximum size for all tiles is very low. However, a single row of tiles can be further partitioned into a lower number of tiles until fitting into one UDP packet.

Hence, *Hextile* encoded framebuffer updates can be split as follows:

```
1   buffer rectangle header;
2   set marker on buffer;
3   for each tile {
4     read and buffer tile;
5     if tile exceeds datagram limit {
6       if marker points to beginning of buffer {
7         /* splitting single row */
8         create new framebuffer update message
9                     of all buffered tiles omitting last one;
10      } else {
11        /* split after complete row */
12        create new framebuffer update message
```

```
13                    of all buffered tiles up to marker;
14       }
15       send new message;
16       remove sent tiles from buffer;
17     }
18     if end of tile line is reached {
19       if current line of tiles was split {
20         /* next line will start at different x-position */
21         create new framebuffer update message
22                    of all buffered tiles;
23         send new message;
24         remove sent tiles from buffer;
25       }
26       if end of tile line is reached {
27         set marker on buffer;
28       }
29     }
30   }
```

Appropriate headers for messages and rectangles must be set for the newly created framebuffer update messages. The message header consists of the message type, which is the *FramebufferUpdate* type, and the number of following rectangles, which is one due to encoding single rectangles only. The rectangle header specifies the x,y-position as well as the width and the height of the encoded rectangle. Due to the predetermined tile numbering and ordering (see Figure 3.4), these values can be easily computed regarding the number of previously sent and the number of contained tiles. Splitting the rectangle with given coordinates and dimensions to four parts as shown Figure 4.6, will result in the four rectangles with given positions and

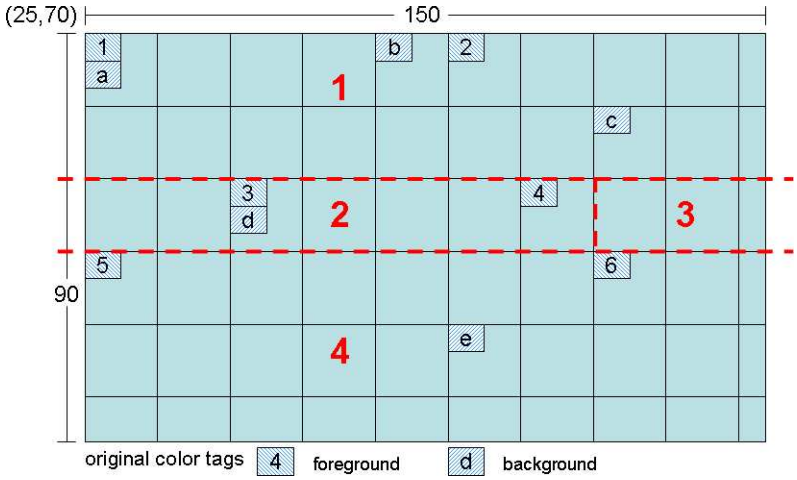


Fig. 4.6. Hextile encoded rectangle to be split

dimensions as displayed by Figure 4.7. The position and width of rectangle no. 1

are the same as those of the original rectangle, because it contains the first tile and complete rows of tiles. The third row of tiles is divided into two rectangles. Even if rectangle no. 3 could contain more tiles without exceeding the maximum size, it must be split as given, because adding the next tile in the sequence (the first tile of the next row) would break the mandatory rectangular shape.

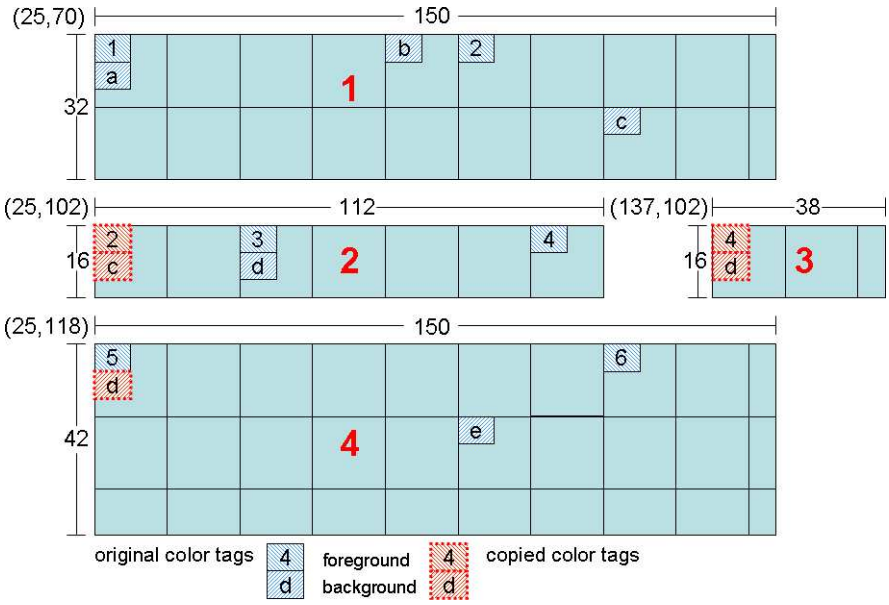


Fig. 4.7. Coordinates and dimensions after splitting

Furthermore, some adaption of the first tile of each new partition is needed. That is because color tags are not specified for each tile. In a sequence of tiles, the foreground and background color is specified once and valid for all following tiles until reassigned. If an existing update message is split, it can only be assured that the first of the newly created update messages, which contain the first part of the original message, has valid fore- and background colors assigned. Therefore the color values have to be acquired and added to the first tile of each new rectangle. Color information can be easily determined and buffered during message parsing. Figure 4.7 displays the color tags that have been added to the first tile of each rectangle. Rectangle no. 1 does not need new color tags, because it contains the first tile of the original rectangle. The first tile of rectangle no. 4 already specifies a foreground color and thus only the background color must be assigned additionally. Note that the background color that is specified by the third tile of the third row stays valid for the rest of the original rectangle and thus for the newly created rectangle no. 3 and no. 4.

4.2.6 Parsing and Buffering

In order to switch from connection-oriented to packet-oriented transmission, individual messages must be distinguished. Due to missing delimiters or information

regarding the length of each message, the incoming stream must be parsed and buffered until the end of a message is detected. Unfortunately, such buffering can result in delayed transmission. Testing with a Java based *VNC proxy* showed, that buffering of large framebuffer updates can take up to two seconds. For view-only clients such a delay is almost not noticeable, but for the teachers interacting with the *VNC server* the response time should be no longer than a few hundred milliseconds. However, large messages must be split anyway, as they exceed the maximum size for datagrams. But splitting can be done without reading a message in total. *Hextile* encoded messages can be split after each line of tiles, which are 16 rows of pixel data. Decoding framebuffer updates of other encoding schemas also produces pixel data in left-to-right top-to-bottom order. Hence, framebuffer updates can be split and forwarded partially without parsing and buffering them completely in advance.

Another solution is to serve the teacher's client by forwarding the stream received by the VNC server as displayed in Figure 4.8, which causes almost no additional delay. Obviously, such forwarding can only be achieved by use of stream-oriented data transmission. However, connection-oriented TCP communication offers the benefit of being reliable and serving the teacher's client by point-to-point communication is unproblematic, because scaling is not needed as the number of connected teachers should be rather low.

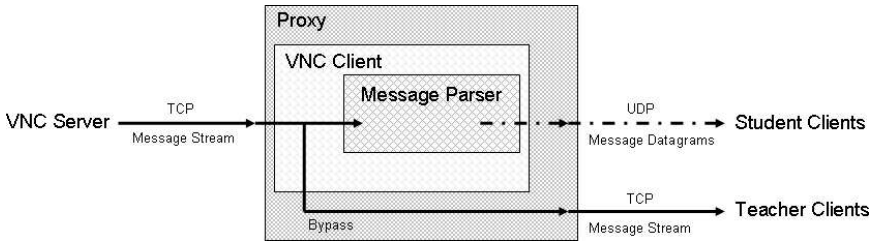


Fig. 4.8. Supplying teacher clients without parsing and buffering

4.2.7 Format of Datagram Content

Applying the zlib compression to the stream of messages and framebuffer updates, as is the case for some encodings (e.g. *ZRLE* and *tight*), is discouraged, because of the resulting message dependencies and the potential for failure due to packet loss.

Nevertheless, zlib compression is useful in order to reduce the network load, but must be applied to each single datagram. Hence, we must use an individual zlib stream for each datagram instead of using one zlib stream for the entire communication. As the compression ratio is rather low if applied to short sequences of data, only large datagrams should be compressed. Hence, each datagram must be labeled if compressed or not, which can be achieved by setting a flag. Instead of adding an additional byte, the compression flag can be integrated into the sequence number and thus will consume a single bit only. Regarding the fact that datagrams, which are

missing for some time, will rarely reappear and, even if they do, they are probably outdated anyway, it is sufficient to distinguish a few datagrams only. Therefore, we suggest limiting the range of sequence numbers to 128 and thus a header of one byte is sufficient as given in Figure 4.9a. Note that this is the header of the contained data and does not influence the datagram header at network level.

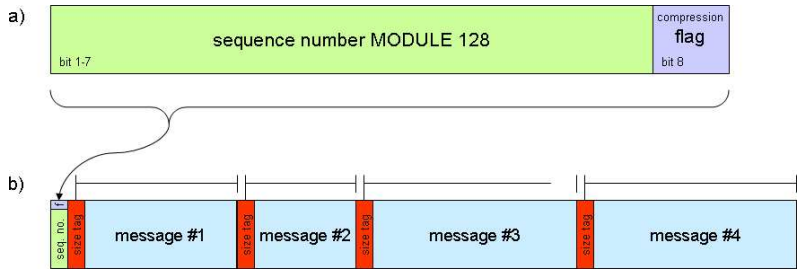


Fig. 4.9. Datagram with header and size tags

Considering the format of the datagram content, this one byte header is sufficient regardless of whether each datagram contains a single RFB message or a sequence of messages. However, we suggest adding a *size tag* preceding each message, which specifies the length of the following data as given in Figure 4.9b. This enables faster message handling at client side because it allows entire messages to be read without parsing them.

4.2.8 Unreliable Transmission: Packet Loss

The UDP protocol provides best effort transmission. UDP datagrams can be lost or delivered in the wrong order. Out of order delivery can be detected by adding a sequence number to each datagram. Whenever a datagram is received without having received the previous one, a packet loss may have occurred or the missing datagram may be delivered later. The loss of client to server messages of the type *KeyEvent* and *PointerEvent* are not crucial, but annoying because they affect a teacher's interaction with the desktop. The loss of such events result in an unresponsive desktop and actions must be applied again. The clipboard feature is rarely used and therefore a loss of *ClientCutText* messages will most probably stay unnoticed. Losing a *SetEncodings* message is also non-critical, because transmission stays unchanged. In contrast, the loss of a *SetPixelFormat* message affects the communication badly because the client expects all subsequent pixel data to be encoded by the specified pixel format but the server will still use the previous encoding. At best, the displaying of framebuffer updates is messed up due to applying a wrong color mapping. If the new format has changed the color depth or specified another number of bits per pixel, the communication will fail soon after in all probability. If *FramebufferUpdateRequest*s are lost, the communication will freeze, because the server will assume that the client is not ready to process messages and therefore will not send any more updates and the client keeps waiting for updates, which will never arrive. However, this problem can be solved by resending the request after a certain

period. Nevertheless, client-to-server communication suffers badly from unreliable datagram delivery. Hence, unreliable delivery of client-to-server messages is not very practicable. However, when designing a new client implementation regarding the previously suggested agreement on the pixel format and encodings, the *SetPixelFormat* and *SetEncodings* message types can be eliminated. If the proxy generates the *FramebufferUpdateRequest*s (in an appropriate manner) even the freezing problem is circumvented, but the lost event messages will still be annoying.

Considering the scenario of a single teacher (or maybe even a few teachers) that interacts with the desktop and the vast majority of clients (of students) connected in a view-only fashion, it will have almost no effect on the scalability of our environment, if the teacher's client will be connected to the server (or proxy) via the reliable connection-oriented TCP protocol as is the case in the original VNC design.

In the opposite direction the loss of messages of the types *Bell* and *ServerCutText* are unproblematic. Loosing a *SetColourMapEntries* message will result in a wrongly colored representation of the framebuffer updates, but unlike lost *SetPixelFormat* messages will not cause a failure while parsing subsequent messages. As the *SetColourMapEntries* type is rarely supported and may be disabled at server side, messages of this type can be eliminated. The last and most important server-to-client message type is *FramebufferUpdate*. Losing updates will result in outdated framebuffers on the client side. The outdated area could be requested again in a non-incremental fashion (as the server assumes that the client's data equals its own), but this demands the loss of an update to be noticed first. Due to the sequence numbers, the loss of *some* message is exposed, but not the type of the lost message. Hence, the client can only assume a framebuffer update to be missing, but without any clue regarding which area is outdated. As the previously sent request was the request to incrementally update the complete framebuffer (regarding the workflow of the VNC architecture as described in Section 3.2.1), the only way to ensure its framebuffer will be updated is to request a complete non-incremental update, which may be an overreaction for some lost data of maybe a few bytes only.

In order to compensate for lost datagrams, we rather suggest some kind of redundancy of transmitted pixel data in analogy to video transmissions. The architecture should ensure the entire framebuffer content to be transmitted redundant within a certain period. A full non-incremental update would fulfill this requirement, but as such updates are generally large, redundancy should rather be distributed in time to reduce bandwidth peaks. This can be achieved by transmitting smaller parts of the framebuffer, which can be initiated by a proxy that requests certain parts of the framebuffer to be updated after certain periods. The size of these parts and the frequency of their transmission determines the time within which all image data can be received as additional non-incremental updates. For example, partitioning the framebuffer into 24 disjoint horizontal stripes, each containing 1/24 of the framebuffer as displayed by Figure 4.10, and sending single stripes successively every 5 seconds ensures a redundant copy of the framebuffer content to be sent every 2 minutes. This does not ensure that after this delay clients have updated all image data, because non-incremental updates are also transmitted via UDP and therefore can be lost as well. But it increases the chance of viewing a completely up-to-date framebuffer. If the rate of lost packets is very high this method is obviously not

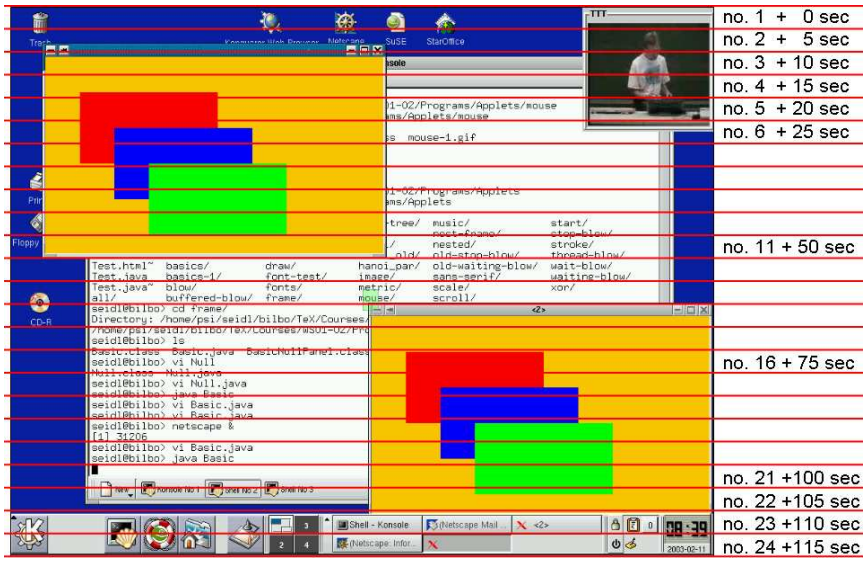


Fig. 4.10. Stripe partitioning for redundant transmission

practicable, but audio and video transmission will also suffer from packet loss and therefore will result in very low quality as well. In this case a useful synchronous lecture transmission is not possible anyway.

Note that a client or *proxy*, which fullfils the task of a *session recorder*, should not be served via unreliable communication, because this may produce *asynchronous electronic lectures* of lower quality.

4.2.9 Unreliable Transmission: Out of order Delivery

The UDP protocol is not only unreliable regarding data transfer as datagrams may be lost without notice. Datagrams may also arrive in the wrong order. The sequence numbers, suggested previously, enable the client to detect that datagrams are missing. However, missing datagrams may be received anytime later. Lost update messages cause outdated framebuffer content at the client side. Messages that are received too late contain the missing data and hence should be used to update the content. However, just decoding and copying the included pixel data will not necessarily result in up-to-date framebuffer content. Consider framebuffer updates with positions, dimensions and sequence numbers as given in Figure 4.11. The updates no. 1 and no. 5 can be applied later without causing any faults, because they are distinct from all other rectangles. Even update no. 6 can be applied later, but copying the pixel data of update no. 2 after applying all other updates, would result in overwriting update no. 6 and parts of no. 4. Due to the higher sequence numbers, the content of update no. 4 and no. 6 is more up-to-date and thus must not be erased. Update no. 4 can be applied after no. 5, but not after no. 6.

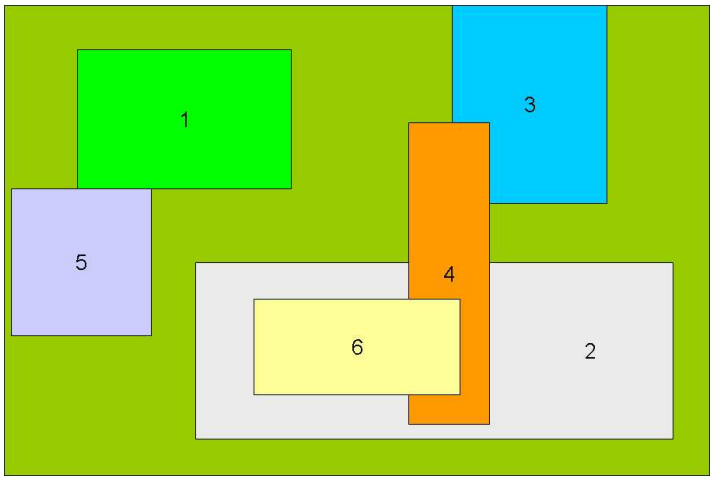


Fig. 4.11. Ordering of framebuffer updates

Therefore, we must ensure that framebuffer updates, which are received too late, are only applied in a way that they will not overwrite (more) up-to-date pixel data. This can be achieved by re-applying the newer updates after processing the late comer, but demands complete framebuffer updates or at least the contained rectangles of pixel values to be buffered by the client.

Another solutions is to copy only those pixels of the out of order delievered update message, which are not covered by updates with higher sequence numbers. This can be achieved by computing rectangle intersections to determine the affected areas. For this approach it is sufficient to buffer rectangle headers only instead of buffering entire framebuffer updates, because the headers contain the positions and dimensions of the reactangles. Hence, the second approach causes less memory usage for buffering and less message decoding, because each message is only applied (maybe partly) once.

Out of order delivery of datagrams will generally not cause (update) messages to be received much later than supposed, and if this does occur then the contained rectangles are probably no longer of interest because later updates may have already overwritten the area. Hence, we can limit the buffer size to contain a few updates only, so that the number of intersections to be computed or updates, which possibly must be re-applied, is manageable. Furthermore, the necessary computations can be delayed to idle phases in order not to influence the decoding of incoming up-to-date messages. Especially as the out of order delivered update may no longer be useful if overwritten by these incoming messages.

4.2.10 Additional Unicast Support

Currently not all networks support UDP multicasting (e.g. the Deutsche Telekom does not), but this hopefully will change with the spreading of the new version 6 of the internet protocol (IPv6) [Deering and Hinden, 1998]. Until then, it is advisable

to supply a limited amount of unicast clients in addition to multicast support. These clients receive the same data (packed in UDP datagrams) as the multicast clients, but instead of being part of a multicast group each datagram is sent individually to each client's IP address. In order to limit the network traffic produced by sending each packet several times, the number of clients should be restricted by the server according to the available bandwidth. Due to the connectionless communication, there must be a possibility to detect if unicast clients are still interested in receiving packets. In the case of UDP multicast communication this is done automatically by the network, but unicast delivery is done without knowledge of the presence of a receiver or not. Hence, a server may send datagrams via unicast even if the client is no longer running. Therefore, we demand disconnecting clients to send a short acknowledgement to the server before terminating. A short UDP datagram is sufficient. This is sufficient unless the acknowledgement is lost or not sent due to abnormal program termination, system shutdown or network trouble. Thus, it is advisable that each unicast client periodically sends an acknowledgement — an alive message — that it is still receiving data. The server administers a list of unicast clients, where clients are inserted when they connect and removed when a disconnect acknowledgement is received or *alive messages* are no longer received. In order not to disconnect clients due to packet loss, one or two missing alive messages should be tolerated by the server. Hence, the client must timestamp the received acknowledgements and disconnect clients after a timeout of three times the period between acknowledgements. This procedure guarantees that (most of the time) unicast traffic is produced only if consumed by a client.

4.2.11 Client Initialization

Regarding the conceptual design of a proxy placed between a VNC server and our newly designed clients, the proxy acts as VNC client and thus must connect and initialize according to the RFB protocol. The initialization of clients, which connect to the proxy, may differ.

The initialization specified by the RFB protocol specification [Richardson, 2005] consists of a client authentication, setting the connection to be shared or not, specifying the framebuffer resolution and an agreement concerning the pixel format and applied encodings. Even without requesting authentication and if all clients are shared by default and predetermined pixel format and encodings are applied, at least the resolution must be specified unless all teachers are forced to use the same resolution for all time, which is discouraged as it contradicts our transparent approach. In order to allow more flexibility it is advisable to keep the specification of pixel format and encodings during initialization.

During initialization a teacher's client must be distinguished from the students' ones in order to provide full access to the desktop or not. Furthermore, multicast and unicast clients must be distinguished as they are servered in different ways. Requesting authentication for students' view-only clients or not depends on whether the course is open to the public or not.

Thus the initialization process specified by the RFB protocol must be extended by requesting a certain kind of connection: *full-access*, *view-only multicast* or *view-only unicast*. The concept of a full-access and a view-only password may be applied. At least, the teacher's connection should require a correct authentication. If distinguishable passwords are provided for each of the three connection types the explicit request can be omitted. Furthermore, the initialization may omit the setting of the shared option (as all connections are forced to be shared anyway). The initialization of pixel data and encodings can be carried over from the RFB protocol, but later resetting of these options by a client is abolished. In order to provide meaningful synchronous electronic lectures, the integration of audio is mandatory. Hence, the initialization will probably require some information concerning the transmission of audio and video streams, such as data type, addresses and ports used for transmission.

In order to ensure a reliable initialization, a TCP connection will be used during handshaking. For the student clients the connection will be closed after all properties are arranged. Further data transmission, which is unidirectional server to client only, is handled by scalable UDP communication. Only teacher clients (generally there will be only one) stay connected via reliable and bidirectional TCP.

Unidirectional Initialization

The initialization between the VNC server and the VNC clients as well as the initialization between our proxy and the new clients is handled via bidirectional communication as both participants interact with each other. In order to serve clients via unidirectional transmission, which can be useful for satellite connections not offering an up-link for clients, a solution to support unidirectional initialization of clients must be achieved. As the complete handshaking phase consists of a few dozen bytes only and clients have to accept the server's properties anyway, initialization messages can be interspersed into the datagrams delivered to clients. Any client can survey the incoming messages of a given UDP multicast address until such an initialization message is received and attend the lecture transmission thenceforth. Obviously, no client authentication can be established, but messages could be encrypted using a key only known by authorized clients. The second restriction of unidirectional initialization is the missing possibility of additional unicast support, because this requires dedicated message forwarding for each client and thus knowledge of the clients existence. Furthermore, no input events can be sent, i.e. unidirectionally served clients are always view-only multicast clients.

4.2.12 Framebuffer Initialization and Late Join

In the original VNC set-up, each client places individual requests and, hence, is free to request a non-incremental update of the complete desktop to initialize its copy of the framebuffer. That is only practicable as long as only a few clients participate in a *synchronous electronic lecture*. As soon as the number of participants increases, they cannot be supplied with individual initial updates anymore, because this would possibly cause heavy server load and more network traffic and, thus, less scalability. This is particularly the case because commonly most students connect during a short

time slot at the beginning of the transmission. Therefore initial framebuffer requests should be gathered and answered with a little delay, which is no problem as audio and video streams have a little start-up delay as well. Moreover, a dedicated request to achieve an initial framebuffer update is not necessary as it always follows the initialization. By logging the time of connection, the proxy can periodically check if some clients have connected recently and sent a non-incremental update request on demand.

Without any initial framebuffer, newly connected clients will display only the incremental updates the server sends later. Due to the non-incremental update stripes used to diminish the effect of packet loss, each client should receive a valid framebuffer once within a certain period (of two minutes for the example given by Figure 4.10). Obviously, this is the only possible framebuffer initialization for unidirectionally initialized clients.

Instead of requesting updates for newly connected clients, the proxy can reuse the update stripes and in this will reduces the server load. Buffering the messages containing these stripes, allows new clients to be supplied with already encoded messages instead of requesting a complete framebuffer update. As all clients receive the same data, we have to take into account that the updates might be outdated. Outdated updates are better than no updates (for new connections), but must not have any impact on established clients. Therefore such updates must be marked, e.g. with a flag, and only be displayed by newly connected clients within a short period after their initialization. It is advisable to notify the user that the displayed content might be outdated, because otherwise a student may wonder why the teachers narration is not related to the displayed content. This can easily be achieved by displaying the marked updates in grayscales instead of colored.

Note that the period until a client can display an up-to-date framebuffer is not necessarily the period in which all non-incremental stripes are requested once. Any modifications of the server's framebuffer that occur after the client has connected, will be propagated in the form of framebuffer update messages. Especially, whenever the teacher switches to another slide, most of the framebuffer will be updated for all clients. Hence, after joining late, students will be able to view at least the next slide.

Commonly the term *late join* addresses the handling of clients that connect to an already running session. In our scenario, the solutions suggested above will serve any client in the same way, regardless of whether they connect in time before the lecture starts or, for instance, 20 minutes later. This is because a certain point in time, which describes when the lecture starts, is meaningless on the protocol layer. For the RFB protocol the session begins when the VNC client connects to the VNC server. In the case of individual TCP connections there cannot be a late joining client. In our environment, which supplies all clients with the same data, the beginning of a session is the establishment of the connection between the proxy and the VNC server. Hence, all clients (that connect to the proxy) join too late and no special late join handling is needed. However, for synchronous lecture transmission, the proxy can request a non-incremental framebuffer update at the time the lecture starts, e.g. by a special start button. This will ensure that all clients that are connected beforehand, but may

not have received an initial update yet, will be supplied with a valid framebuffer for the beginning of the lecture. Note that starting the proxy exactly at the time the lecture is supposed to start is discouraged, because students should have the possibility to connect slightly earlier to be ensured that their connection is properly working. Otherwise they may panic a little bit.

4.3 Summary

In its original design the VNC framework and the RFB protocol is not as scalable as required to provide a distance learning scenario with a large number of students, which is mainly caused by the lack of support for the one-to-many routing scheme. However, two steps are needed to handle many students simultaneously. At first we must ensure that all clients can be supplied with exactly the same data, which could be achieved by reducing the vast, but unneeded, diversity of pixel formats offered by the RFB protocol and enforcing the use of the same encodings for all clients. This is possible as long as clients can be assumed to be (almost) equal regarding processing power and network connections, which is the case in our e-learning environment. Even slightly outdated computers are more than fast enough to decode and display RFB framebuffer updates and, since we will combine VNC with audio (and video) streams, a faster connection than just a dial-up internet connection is required. Otherwise two or three categories of clients could be introduced and supplied with different one-to-many transmissions using different color depths (and also different settings for the audio and video streams if required, or even with video omitted).

The second step to support one-to-many message delivery is the switch from reliable, stream-oriented TCP to unreliable, packet based UDP data transfer. Although the RFB protocol lacks message delimiters or length-of-message tags, streams of RFB messages cannot only be distinguished by parsing and buffering them, but also can be split-up to fit into single UDP datagrams respecting UDP's size restriction. For stable network connections packet loss should rarely occur. However, some methods to compensate lost datagrams have been suggested. Moreover, client initialization was discussed including unidirectional initialization of receive-only clients to provide satellite transmission of synchronous electronic lectures.

The suggested modifications of the VNC architecture should improve VNC regarding our Criterion C11: Synchronous Electronic Lectures. By switching from point-to-point towards one-to-many communication, the *scalability* (Criterion C11b) is extended tremendously from a few to possibly hundreds of clients. Certainly this will work only as far as the network supports UDP multicasting. Otherwise clients must be served via multiple point-to-point connections, which limits scalability by the number of possible parallel connections not exceeding the bandwidth of the network. Even if UDP multicasting is supported by the network, it is advisable not to leave behind students with unsuitable network connections. Hence, the system should support at least some clients via unicast. Supplying unicast clients is only limited by the bandwidth. Unlike the original VNC architecture, multiple clients do not influence the processing load of the server, because no individual transformation and encoding of pixel data must be processed.

The benefit of *scalability* is achieved at the cost of small modifications to the *RFB protocol* and a variation regarding *data transmission*. Hence, Criterion C9d, which prefers standard formats instead of proprietary ones, is a little loosened. However, the benefits are much higher and the VNC architecture with its RFB protocol are not that widespread as, for instance, the use of RealMedia [RealMedia, 2006] or WindowsMedia [Windows Media, 2006] formats .

Further aspects of providing synchronous electronic lectures, namely *late join* (C11c) and *error tolerance* (C11d) have been discused and (partially) solved. Criterion C11e: Synchronization regarding real-time delivery is given by the VNC concept per se and the delay caused by placing a proxy within the communication line is negligible. Synchronization regarding multi user inputs is given by supporting parallel teacher clients, but could be extended further, for instance to support collaborative sessions for students. As teacher clients are supported via reliable, but unscalable, TCP connections, this may be a restriction of C11b: Scalability. However, it is hard to think of a scenerio of a meaningful and productive collaboration of several hundred clients actively interacting with a single desktop. Nevertheless, it may be useful to allow single students to interact with the desktop, which can be achieved be forwarding the client's *Key-* and *PointerEvent* messages to the VNC server. The proxy can filter the events of unauthorized clients. In order to avoid packet loss, a reliable TCP connection can be established between the authorized clients and the proxy. Some mechanism of hand-raising and dynamic authorization should be provided during the lecture.

Implicitly, we have already considered Criterion C11a: Addressed Participants, which are one (or maybe a few) teacher(s) and a large number of view-only students. This scenario fits to our lecturing concept. However, extending to environment in order to support other setups such as connecting lecture halls or collaborative sessions would be useful. In fact, we already have connected two lecture halls by use of a bidirectional communication channel as described in Section 2.1.2 (page 9). This was achieved by integrating additional audio and video streams but only on desktop stream. In order to support *Computer Supported Cooperative Work* (CSCW) [Li and Hopper, 1998a] and [Li et al., 1999a] suggest a proxy that provides floor control and user awareness.

To summarize, we have extended the *scalability* of the VNC architecture and thus strengthened VNC's suitability as a basis for an *e-learning environment*.

5

VNC Session Recording

One aim of this thesis is to provide a flexible and valuable environment to create *asynchronous electronic lectures* by recording computer-based live presentations. The *Virtual Network Computing* (VNC) architecture [Richardson et al., 1998] and the *Remote Framebuffer* (RFB) protocol [Richardson, 2005] provide access to a desktop, but are not specifically designed for the purpose of recording.

However, as the RFB protocol works on pixel basis, it can be used to serve a screen recording process. The screen recording technology consists of two stages: grabbing and encoding. Grabbing pixel data is solved by the VNC architecture by providing a framebuffer of pixel data, which can be used as the input for further encoding and storing (Figure 5.1). Due to using the framebuffer as the interface between the grabbing and the encoding process, both phases are independent of each other. Nevertheless, the incoming stream of RFB messages delivers beneficial meta data for the encoding process, because it specifies which areas of the framebuffer are updated and when. Such triggered processing probably achieves a better encoding and compression ratio than grabbing and encoding at a fixed frame rate (as commonly is the case for screen recording).

Fig. 5.1. Framebuffer as interface between grabbing and encoding processes

As the encoding phase is independent of the grabbing phase, any kind of encoding can be applied. However, common encodings designed to compress motion pictures are lossy (contradicting our Criterion C9a: Lossless Reproduction) and not suitable for screen content [Lauer and Ottmann, 2002]. The *TechSmith Screen Capture Codec* [TSCC, 2006] is especially designed for lossless compression of screen captured data, but is available for a single platform only and thus not platform independent as required by Criterion C10: Platform Independency. As the encodings specified by the RFB Protocol Specification [Richardson, 2005] are especially designed to compress screen content, it is natural to apply them not only for transmission but also for

recording purposes. Moreover, it is not necessary to encode the framebuffer of the client in order to record the pixel content. In fact, the framebuffer update messages received by the VNC server already contain encoded pixel data and therefore the decoding and re-encoding can be avoided.

5.1 File Formats

The concept of *VNC Session Recording* was introduced by [Li and Hopper, 1998a] and oftentimes reapplied throughout their later work [Li and Hopper, 1998b, Li et al., 1999b, Li et al., 1999a, Li et al., 2000a, Li et al., 2000b]. Session recording is achieved by seamlessly placing a proxy between the VNC server and the client component, analogous to the synchronous scenarios described in the previous chapter. In fact, the proxy, which merges clients' events and forwards framebuffer updates, can also handle the session recording. The recording proxy writes all framebuffer update messages with an additional timestamp to a log file (Figure 5.2). The timestamp specifies the delay since the beginning of the session.

Fig. 5.2. Logging messages with timestamps

Logging the stream received by the VNC server can be achieved by storing byte after byte without any knowledge of the contained data. In contrast, adding a timestamp to each message demands that messages are distinguished from each other. As the RFB protocol does not specify message delimiters or provide the lengths of messages, the proxy must parse and partly[1] decode messages.

Replaying or reviewing[2] is achieved by reading, decoding and displaying the logged sequence of messages in the same way as is the case during synchronous playback, except that the input is read from a file instead of being received via network. The messages are sequentially replayed at the same rate as when they were recorded.

The architecture design described in [Li and Hopper, 1998b] consists of a *Review Server*, which supplies a distant *Reviewer* with messages (analogous to the client/server design of VNC). The task of the Review Server is to read messages from a log file and send them according to their timestamps. Li and Hopper measured the duration of replaying recorded sessions in fast forward mode, which means processing and displaying as fast as possible, ignoring timestamps. This demonstrates the fast forward mode to be no better than four times faster compared to ordinary playback, which would be over 20 minutes for a 90 minute lecture. One reason for the bad performance of their architecture probably is the format of their log file, which neither provides message delimiters nor sizes, as is the case for the original RFB protocol. The *Review Server* does not need to access any message content, but

[1] sequences of pixel data can be skipped without decoding if the length is known
[2] following the naming of VNC clients, which are often called *VNC Viewer*

must parse each message in order to access the subsequent timestamp. The distant reviewer must parse the message once more in order to display the recorded session.

Hence, we suggest extending the format by adding an additional tag before each message that specifies the size of the following data in bytes (Figure 5.3). The size can easily be determined during recording by buffering each message before writing it to the log file, which we already stated as mandatory to archive scalability via UDP multicast (Section 4.2.3). The size tags enable better message handling during replay. Messages can be read in total without parsing and thus processed faster. Size tags allow messages to be skipped on demand, which can be beneficial for fast forwarding or random access. Additionally, this provides a better extensibility in future by enabling skipping of unknown message types.

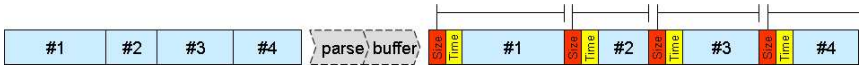


Fig. 5.3. Logging messages with timestamps and size tags

Instead of logging the size of each message, [Li and Hopper, 1998b] suggest extending the format of the log files by adding file pointers referencing the previous message (see Figure 5.4). However, as this format still demands files to be read sequentially and the file pointers can be easily achieved by caching the positions while reading (and parsing) the log file, the benefit of storing the file pointers within the log file is rather limited. In order to improve performance, some or all messages may be kept in memory anyway.

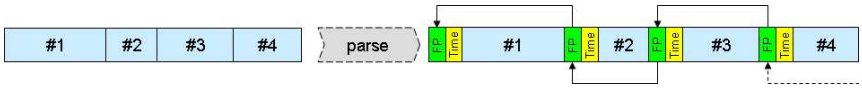


Fig. 5.4. Logging messages with timestamps and back references

5.1.1 Delta Pixel Values

VNC's framebuffer updates contain *absolute pixel values*, which are sufficient for sequential replay but insufficient for rewinding. Old pixel values of the framebuffer are replaced by new ones and cannot be recovered unless tracing back to the nearest past update that contains the relevant pixels. Therefore, [Li and Hopper, 1998b] suggest storing (relative) *delta pixel values* instead of absolute ones. The conversion is done during recording by applying the *XOR* (*Exclusive OR*) operator to the pixels of the framebuffer updates and the corresponding pixels of the previous framebuffer (Figure 5.5).

Storing delta values allows the *Reviewer* application to process earlier recorded sessions in both directions. This is very useful as the user can rewind the session a little bit whenever she/he did not grasp something immediately and thus wants to replay

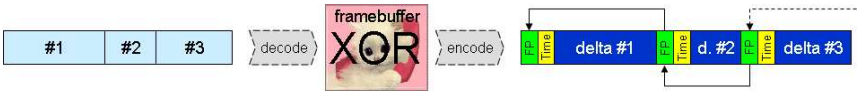


Fig. 5.5. Logging delta framebuffer updates

that part of the session once more. Skimming through a recorded session by *visible scrolling* (Criterion C7c: Visible Scrolling) also demands fast processing of recorded content in both directions. Storing delta pixel values enables sequential playback in either direction, but still does not support *random access* as is requested by Criterion C7b: Random Access.

Unfortunately, this operation may require decoding and re-encoding of pixel data as shown in Figure 5.5 and moreover, the usage of two framebuffers, one for the current pixel values and one for those of the newly received framebuffer update. As [Li and Hopper, 1998b] make use of *Raw* encoded and thus uncompressed framebuffer updates, the *XOR* operator can be applied byte after byte. However, we discourage the usage of *Raw* encoded values due to resulting in large file sizes and high bandwidth consumption.

Rewinding absolute pixels demands to process at least all framebuffer updates that have any contribution to the framebuffer state at the demanded playback point. In the worst case, this requires parsing of all previous messages if the file format does not provide access to message headers, as is the case for the formats in Figure 5.2 and 5.4, and to rectangle headers, as is the case for all formats discussed so far.

5.1.2 Recording Distinct Rectangles

Handling messages becomes easier due to storing the length of each message within the log file, which prevents many parsing processes. However often update messages do not need to be accessed as a whole, but rather the rectangles they contain must be accessed. Sometimes it is even sufficient to access rectangle headers. Recall the idea of rewinding or the computing of rectangle intersections to re-apply messages if delivered in the wrong order (Section 4.2.9). Unfortunately, accessing rectangles requires message parsing, because each framebuffer update may contain a sequence of rectangles without delimiters or knowledge of a rectangle's length. Therefore, each rectangle may be preceded by a size tag as given in Figure 5.6 analogous to the extended message logging (Figure 5.3).

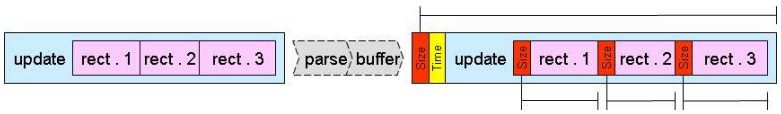


Fig. 5.6. Logging messages with size tags for rectangles

Instead of adding a tag before each rectangle, each framebuffer update that contains a sequence of rectangles can be transformed to a sequence of framebuffer update messages, all of which contain a single rectangle only (Figure 5.7).

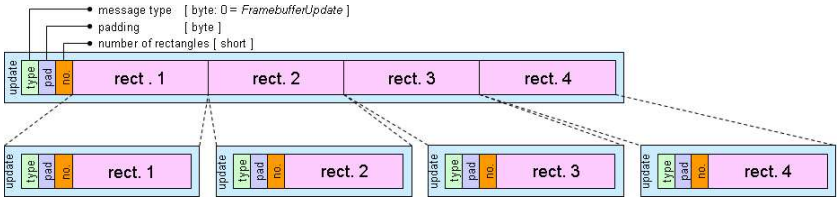


Fig. 5.7. Transforming sequences of rectangles to a sequence of messages

After the transformation, the previously suggested format of logging lengths of messages is sufficient to access rectangles individually without message or rectangle parsing[3] (Figure 5.8).

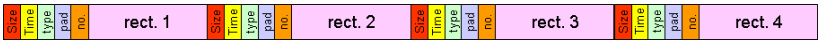


Fig. 5.8. Storing updates with single rectangles

If logging no other messages than framebuffer update messages and ensuring that update messages contain single rectangles only, the message headers can be omitted, because they will always consist of the same four bytes, which is the type byte (always 0 = *FramebufferUpdate*), a padding byte (of irrelevant content) and two bytes for the number of rectangles (in this case always one). Hence, the recorder stores timestamped rectangles rather than messages as given in Figure 5.9.

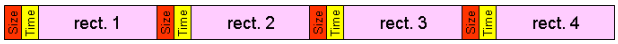


Fig. 5.9. Storing rectangles (omitting message headers)

Preserving only rectangles is sufficient in order to replay a VNC session. As the aim of this thesis is to provide more features than sequential replay, we should regard some kind of extensibility of the format to enable integration of additional data, for instance annotations. This could be done by withdrawing the idea of storing rectangles and keep storing framebuffer update messages that contain single rectangles. However, as framebuffer updates will constitute the most part of the session, we suggest another solution. The rectangle headers contain a field for the *encoding type* of the following pixel data. By ensuring that additionally introduced message types will not collide with the potentially occurring encoding types, we can reuse the encoding type field for the purpose of marking other data as well. Thus, the rectangle's *encoding type field* becomes our *message type field*.

[3] except for the fixed sized message header

Recall the format of the rectangle headers as given in Figure 3.3. The RFB protocol specifies the encoding type to be after the values, which set the rectangle position and dimension. However, these fields are possibly meaningless for most other message types. Even some of the RFB protocol's own pseudo-encodings (e.g. *DesktopSize*) do not require (all of) the dimension and size fields, but must contain them due to the defined ordering. In order to circumvent this problem, we suggest adapting the header by moving the type field to the beginning (in combination with removing the original message header) as given in Figure 5.10. The other parameters must only be included if required by the given *encoding type* (which is also our *message type* now).

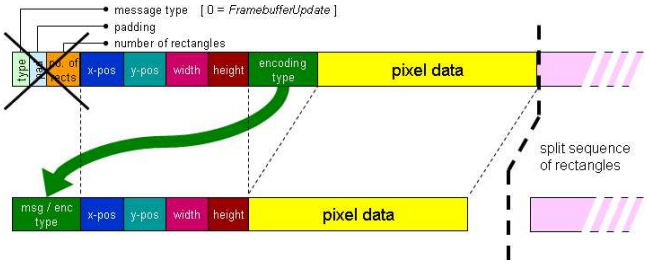


Fig. 5.10. Storing rectangles (omitting message headers)

Note that in the first public release of the RFB protocol, which was *Version 3.3*, all defined encoding types *required* all parameters of the rectangle header and therefore any arbitrary ordering was equally meaningful. The above mentioned pseudo-encodings were introduced later.

5.1.3 Event Logging

Besides logging framebuffer update messages, [Li et al., 1999a] also log user events, such as *KeyEvent*, *PointerEvent* and *ClientCutText* messages. These events are not needed for replaying a session, but may deliver meta data for indexing and retrieval purposes. Li et al. suggest the use of two log files, one for the updates and one for the events. Framebuffer updates are logged with timestamps and back references in the form of file pointers as given in Figure 5.4. The second log file records user events, but also stores the timestamps of the update messages, which allows user events to by synchronized with framebuffer updates later (Figure 5.11).

Logging client-to-server and server-to-client in the same file will fail because the values that represent certain message types are not distinct (see Section 3.2.2). The value 0 corresponds either to the server's *FramebufferUpdate* type or to the client's *SetPixelFormat* type. However, logging only the types *KeyEvent*, *PointerEvent* and *ClientCutText* as user events, will not collide with any server message type.

Note that logging *KeyEvent* messages will preserve any keystroke of the teacher and hence the log will contain any password entered during a session although it was not visible on screen. For security reasons, we discourage logging of key events.

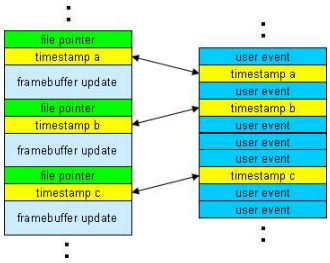


Fig. 5.11. Synchronization of logged user events and framebuffer updates

5.1.4 Log File Header

Framebuffer update messages can only by parsed if the number of bytes used per pixel value is known, because reading, for example, 10 pixel values from a RFB stream or log file means reading 10, 20 or 40 bytes for data sizes of 8, 16 or 32 bit, respectively. Decoding of framebuffer updates requires a framebuffer to be initialized to the same size as the server's framebuffer, and translating pixel values to color values demands knowledge of the applied pixel format.

The original (synchronous) VNC client is provided with this data during initialization. This is also the case for the environment of [Li and Hopper, 1998b], where a *Review Client* connects to a *Review Server*, which supplies the client with messages read from a log file. As Li and Hopper's log file format does not enable the reading of complete messages due to missing size tags, the *Review Server* must also parse message after message and therefore must, at least, know the number of bytes per pixel. Even if the log file provides the suggested size tags, the *Review Server* must know the parameters in order to initialize the client. If distributing recorded VNC sessions via download or storage media (e.g. CD and DVD), the (asynchronous) *review application* must also be provided with those parameters. Hence, for each log file we must store some *initialization parameters*, either as a separate file or, as we prefer in order to reduce the number of files, as the *header* of the log file. Note that the size tags are still useful, because they enable messages to be read and forwarded without being parsed.

Recall the *initial handshaking* as specified by the RFB protocol [Richardson, 2005]. The communication starts with agreeing on a *protocol version*. Such an agreement is not needed to replay a log file, but the protocol version or a *file format version* is meaningful to distinguish different file formats (considering future extensibility). The *authentication* of the RFB protocol can be omitted unless the file should be encrypted to be readable for authorized students only. The *server initialization* provides the parameters of the framebuffer (resultion, pixel format and color depth), which are required to decode framebuffer updates. Furthermore, the *server initialization* delivers a name for the session, which commonly is the name of the server, but in our e-learning scenario, storing the name of the teacher and the title course is more suitable (if available).

In close relation to the *initial handshaking phase* of the RFB protocol, we suggest a *log file header* (Figure 5.12) that contains the *protocol/file version* as well as the

server initialization, including the framebuffer's *resolution*, the applied *pixel format* and the *name string*, as specified by the RFB protocol. As the server's *pixel format* may be overruled by *SetPixelFormat* messages of the (synchronous) client, either these messages must also be logged so that the (asynchronous) replay client can adjust message decoding correspondingly whenever required, or we must ensure that the specified format stays valid for the entire log file. Regarding the common VNC workflow (Section 3.2.1) and our intention to supply several synchronous clients with identical messages (Chapter 4), we can assume that the *session recorder* (e.g. a VNC proxy) will set an appropriate pixel format immediately after the initial handshaking, which will not be changed afterwards. In consequence, the *session recorder* will not write a *log file header* that contains the pixel format it *initially* received from the VNC server, but a header with the *currently* valid pixel format.

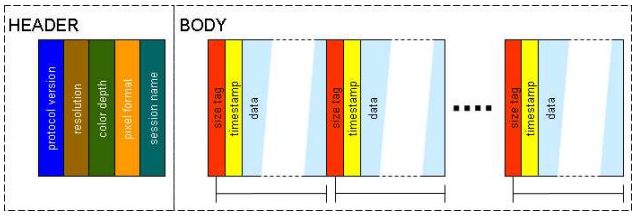


Fig. 5.12. Log file with header

Note that the other approach of logging *SetPixelFormat* messages will fail unless the *client to server* messages are distinguished from the *server to client* messages, because the original RFB protocol defines "0" to be the *message type value* for both the *FramebufferUpdate* ***and*** the *SetPixelFormat* message type. Furthermore, note that for the purpose of replaying log files, it is meaningless to specify a set of encodings, because the logged framebuffer updates are already encoded.

For a detailed *TTT file format specification* refer to Section 9.7 (page 210).

5.2 File Sizes

In order to distribute asynchronous electronic lectures via download or storage media such as CD or DVD, the file size is rather important. [Li and Hopper, 1998b] have compared the sizes of their log files with the sizes of equivalent MPEG video recordings of the desktop and state that even uncompressed (*Raw* encoded) VNC sessions are 8% to 85% smaller than the corresponding MPEG video. However, due to the demand-driven design of the RFB protocol, the rate in which framebuffer updates occur varies depending on the network connections between the server, proxy and client components. A higher bandwidth provides a smoother display of the remote desktop but also causes larger file sizes. The size of the MPEG video will probably stay the same (or just increase slightly) due to its fixed frame rate. As [Li and Hopper, 1998b] results were acquired in 1998, repeating the experiments today will propably expose larger VNC session recordings due to improved network bandwidth and computing power.

On the other hand, the *Raw* encoding they applied during their experiments is very inefficient as it provides no compression of pixel data. But if the size of uncompressed recordings already falls below that of MPEG videos, compressed VNC sessions will perform much better. [Li et al., 1999a] also compare the sizes of uncompressed (*Raw* encoded) and compressed session recordings. Their results reveal that other encodings reduce the file sizes by 40% to 92%. Note that [Li et al., 1999a] do not mention which of the RFB encodings were applied but typically *Hextile* is preferred as it was the best performing encoding that was specified by the RFB protocol in 1998.

The number and the sizes of framebuffer updates (and thus the file sizes) highly depend on the tasks performed during a session. Recording a static desktop without any user interaction will produce only one or a few framebuffer update messages, which supply the client with an initial copy of the framebuffer's content. Showing a movie scaled to fullscreen mode will rather produce a tremendous amount of messages. Commonly, "most of the time [...] only a small area of the screen is affected" [Li et al., 1999a] and hence, the encodings will achieve suitable compression ratios. The experiments described in [Li and Hopper, 1998b] reveal consumptions of 0.59 to 2.78 Mbytes per minute for a framebuffer with a resolution of 796×576 pixel and *Raw* encoded updates (color depth is not stated). Regarding the better compression ratios of other encodings this would result in approximately 0.04 to 1.67 Mbytes[4] per minute. However, we experienced that rarely more than 0.3 Mbyte/min will be consumed if recording real live lectures at a resolution of 1024×768 pixel, 8 bit color depth and applying *Hextile* encoding. Commonly 120–200 kbytes/min are achieved, which results in approximately 10–15 Mbyte for a typical lecture of 80–90 min. These values were acquired by analyzing several dozen recorded live lectures from the courses "*Informatik I*" (Seidl, 2001/02), "*Technische Grundlagen des Elektronischen Publizierens im WWW*" (Meinel, 2001/02; only about 80 kbyte/min), "*Abstract Machines*" (Seidl, 2002) and "*Medienwissenschaft I*" (Bucher, 2002). All these courses were recorded during 2001 and 2002 at the Universität Trier with the first prototype of the *TeleTeachingTool*, which stored timestamped RFB messages with size tags. Table 5.1 lists the file sizes and the average per minute consumption for one course (not regarding the sizes of additional audio and video streams). The other results are available in Appendix A.1.

5.2.1 File Compression

The sizes of the recordings can be further reduced by approximately half by applying *file compression* before distributing the recorded sessions. The 10–20 Mbyte file of a single lecture of about 80–90 minutes typically can be compressed to 3 to 8 Mbyte (only regarding the desktop recording and omitting audio and video files). Hence, it is advisable to apply some kind of compression to the file in order to reduce file sizes (regarding Criterion C9e: File Size and Bandwith). Instead of compressing the log file for distribution only, we rather suggest a *compressed file format*, i.e. applying *zlib deflate* compression to the body of the file (i.e. the messages as shown Figure 5.12). A compressed file format will permanently reduce file sizes, but downloading compressed files requires that the recordings are extracted (to their original size)

[4] improvements of 40–92%: min. $0.59 * 0.08 = 0.0472$ to max. $2.78 * 0.6 = 1.668$

Course: "*Abstract Machines*" (Seidl/Wilhelm, 2002):

name	duration	size	density
abstrakt_2002_04_16_tr.vnc	92 min	9.3 Mbytes	103 kbytes/min
abstrakt_2002_04_23_tr.vnc	85 min	10.0 Mbytes	121 kbytes/min
abstrakt_2002_04_30_tr.vnc	91 min	11.5 Mbytes	130 kbytes/min
abstrakt_2002_05_07_tr.vnc	97 min	13.9 Mbytes	147 kbytes/min
abstrakt_2002_05_14_sb.vnc	65 min	12.4 Mbytes	195 kbytes/min
abstrakt_2002_05_28_sb.vnc	71 min	33.5 Mbytes	483 kbytes/min
abstrakt_2002_06_04_sb.vnc	47 min	21.3 Mbytes	465 kbytes/min
abstrakt_2002_06_11_tr.vnc	86 min	11.3 Mbytes	134 kbytes/min
abstrakt_2002_06_18_tr.vnc	86 min	12.1 Mbytes	144 kbytes/min
abstrakt_2002_06_25_tr.vnc	84 min	10.3 Mbytes	125 kbytes/min
abstrakt_2002_07_02_tr.vnc	90 min	10.3 Mbytes	118 kbytes/min
abstrakt_2002_07_09_tr.vnc	17 min	3.4 Mbytes	207 kbytes/min
		average:	197 kbytes/min

Table 5.1. Files sizes of recorded VNC sessions (8 bit)

before replaying them. If file compression should be optional, the header must be extended with a flag that indicates whether the body of the file is compressed or not. Note that such compression can also be used to encrypt a file so that it is readable for authorized students only.

Since winter 2002/03 the *TeleTeachingTool* applies *zlib deflate* compression to the body of the recordings. Furthermore, VNC sessions are recorded at 16 bit per pixel (instead of 8 bit/pixel) to provide better quality due to more detailed coloring. The resulting files typically achieve rates of 20–60 kbytes per minute for *slide presentations* that are sometimes enriched with *dynamic content* such as animations or programming examples (as shown in Figure 5.13). This is about a quarter of the uncompressed 8 bit recordings. Table 5.2 shows the files sizes of the course "*Abstrakte Maschinen*" of Prof. Dr. Helmut Seidl, recorded during summer 2003. The values of further courses are listed in Appendix A.2.

Fig. 5.13. Presented slides (left) and simulations (right)[5]

Besides the file sizes in bytes, we have also measured the *numbers of updated pixels*, which are listed as average per minute *pixel density* and potentially reveal the degree of the *dynamics* of a recording. However, this is a rather vague measure as a high *pixel density* possibly refers to many slides and/or much *dynamic content* such as animations but also may be caused by movements of the mouse pointer or annota-

[5] Presented in lecture 2004/05/19 of "*Abstrakte Maschinen*" (Seidl, 2004)

Course: "*Abstrakte Maschinen*" (Seidl, 2003):

name	duration	size	density	pixel density
abstrakt_2003_04_29.ttt	91 min	2.3 Mbytes	26 kbytes/min	1255 kpixel/min
abstrakt_2003_05_06.ttt	105 min	2.8 Mbytes	28 kbytes/min	2069 kpixel/min
abstrakt_2003_05_13.ttt	77 min	1.7 Mbytes	23 kbytes/min	1173 kpixel/min
abstrakt_2003_05_20.ttt	95 min	2.3 Mbytes	25 kbytes/min	1407 kpixel/min
abstrakt_2003_05_27.ttt	88 min	2.3 Mbytes	27 kbytes/min	1826 kpixel/min
abstrakt_2003_06_03.ttt	91 min	2.3 Mbytes	27 kbytes/min	2087 kpixel/min
abstrakt_2003_06_17.ttt	92 min	2.9 Mbytes	32 kbytes/min	2743 kpixel/min
abstrakt_2003_06_24.ttt	65 min	2.8 Mbytes	44 kbytes/min	2846 kpixel/min
abstrakt_2003_06_25.ttt	92 min	5.9 Mbytes	65 kbytes/min	7028 kpixel/min
abstrakt_2003_07_08.ttt	91 min	2.5 Mbytes	28 kbytes/min	1785 kpixel/min
abstrakt_2003_07_15.ttt	89 min	2.6 Mbytes	30 kbytes/min	1096 kpixel/min
abstrakt_2003_07_16.ttt	61 min	1.9 Mbytes	31 kbytes/min	1061 kpixel/min
abstrakt_2003_07_29.ttt	81 min	2.8 Mbytes	35 kbytes/min	2211 kpixel/min
		average:	32 kbytes/min	2199 kpixel/min

Table 5.2. Files sizes of recorded VNC sessions (16 bit) with additional file compression

tions made within the presentation software, which are recorded *pixel-based* (unless handled separately by the RFB protocol's *cursor encodings* or are symbolically represented). However, the amount of updated pixels in relation to the file sizes reveal the achieved (inverse) *compression ratios*, i.e. the reduction in data quantity, defined as:

$$compression\_ratio = \frac{size\_original - size\_compressed}{size\_original}$$

Note that the *pixel density* was not measured for the uncompressed (older) recordings because of the different file format, which does not enable direct access to rectangle headers. The *pixel density* would probably be similar to the listed values of the same course given and recorded in one of the following years.

The average (inverse) *compression ratio* of the course "*Abstrakte Maschinen*" (2003) is 99.3%[6]. Other courses by Prof. Seidl achieve compression ratios of about 98–99%. The same is the case for the course "*Programmiersprachen*" by Dr. Alexandru Berlea (recorded in winter 2005/06). Such high compression ratios can be achieved for lectures that present mainly text, sketches, tables or graphs but (almost) no high colored images. Lower *compression ratios* must be assumed for presentations with a high *pixel variety*, i.e. many different pixel values instead of solid coloring.

During the *media science* courses of Prof. Dr. Hans-Jürgen Bucher (Universität Trier) many slides with high colored images and scanned newspaper articles (e.g. Figure 5.14) were presented. The resulting files are about three to five times larger (about 150–250 kbyte/min) than those of Prof. Seidl although the *pixel density* is lower; about 800 kpixel/min compared to 1000–2000 kpixel/min (Table 5.3 and Appendix A.2). For such presentation content the *pixel variety* is much higher which

[6] 2199 kpixel/min at 16 bit and 32 kbyte/min: $\frac{2199k * 2byte - 32kbyte}{2199k * 2byte} = 0.9927$

results in less efficient compression. The analyzed sessions of Prof. Bucher achieve *compression ratios* of about 85–90% (compared to 98–99%). Considering the 16 bit color depth (instead of 8 bit), this is still approximately half the size of the uncompressed recordings.

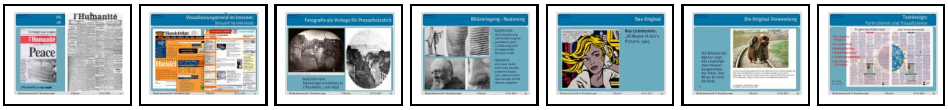


Fig. 5.14. Slides with high pixel variety[7]

Course: "*Medienwissenschaft II*" (Bucher, 2002/03):

name	duration	size	density	pixel density
medien2_2002_11_05.ttt	65 min	10.8 Mbytes	170 kbytes/min	713 kpixel/min
medien2_2002_11_26.ttt	70 min	12.5 Mbytes	183 kbytes/min	1036 kpixel/min
medien2_2002_12_03.ttt	93 min	22.2 Mbytes	244 kbytes/min	854 kpixel/min
medien2_2002_12_17.ttt	90 min	19.5 Mbytes	222 kbytes/min	936 kpixel/min
medien2_2003_01_07.ttt	92 min	30.8 Mbytes	343 kbytes/min	1006 kpixel/min
medien2_2003_01_14.ttt	94 min	25.2 Mbytes	274 kbytes/min	815 kpixel/min
medien2_2003_01_21.ttt	98 min	21.8 Mbytes	227 kbytes/min	560 kpixel/min
medien2_2003_01_28.ttt	93 min	17.8 Mbytes	197 kbytes/min	862 kpixel/min
medien2_2003_02_11.ttt	92 min	3.5 Mbytes	39 kbytes/min	638 kpixel/min
medien2_2003_02_18.ttt	82 min	11.3 Mbytes	141 kbytes/min	764 kpixel/min
		average:	204 kbytes/min	818 kpixel/min

Table 5.3. Files sizes of recorded VNC sessions (16 bit) with high pixel variety

5.2.2 Sizes of Recordings Without Keyframes

Besides the framebuffer updates of the VNC session, all of the analyzed recordings include additional updates as *parted keyframes* (at a period of two minutes as described in Section 5.3.3). In order to test a new TTT implementation, which was created to improve performance, we recorded two courses during summer 2006 without these additional updates. Hence, the resulting recording comprises exactly of the messages caused by the original VNC workflow (Section 3.2.1) (plus a small portion filled by the TTT related header and annotation messages). Omitting the keyframes reduced the *pixel density* to 300–800 kpixel/min, which is about a third of the size of comparable lectures with keyframes (Table 5.4 and Appendix A.3).

Regarding the byte consumption and thus the file sizes, an *average density* of 10–50 kbytes per minute is reached. This results in file sizes of 1–4 Mbyte for typical lectures, which is a very good result, especially considering that such sizes are not

[7] Presented in lecture 2003/01/07 of "*Medienwissenschaft II*" (Bucher, 2002/03)

uncommon as the size of the presented source documents (e.g. *pdf* or *PowerPoint* slide presentations). As these sessions were recorded at a color depth of 24 bit using 32 bit per pixel, the file sizes could even be lowered if storing 16 bit (or 8 bit) VNC sessions instead.

Course: "*Compilerbau*" (Seidl, 2006):

name	duration	size	density	pixel density
compiler_2006_04_26.ttt	89 min	0.6 Mbytes	7 kbytes/min	224 kpixel/min
compiler_2006_05_03.ttt	87 min	2.5 Mbytes	29 kbytes/min	1021 kpixel/min
compiler_2006_05_08.ttt	88 min	2.3 Mbytes	27 kbytes/min	492 kpixel/min
compiler_2006_05_15.ttt	88 min	1.3 Mbytes	15 kbytes/min	451 kpixel/min
compiler_2006_05_17.ttt	86 min	2.4 Mbytes	29 kbytes/min	547 kpixel/min
compiler_2006_05_22.ttt	89 min	4.0 Mbytes	46 kbytes/min	685 kpixel/min
compiler_2006_05_24.ttt	90 min	4.1 Mbytes	47 kbytes/min	786 kpixel/min
compiler_2006_05_29.ttt	85 min	14.9 Mbytes	180 kbytes/min	2392 kpixel/min
compiler_2006_05_31.ttt	88 min	2.2 Mbytes	26 kbytes/min	506 kpixel/min
compiler_2006_06_07.ttt	69 min	1.9 Mbytes	28 kbytes/min	499 kpixel/min
compiler_2006_06_12.ttt	85 min	2.4 Mbytes	29 kbytes/min	427 kpixel/min
compiler_2006_06_14.ttt	80 min	1.2 Mbytes	15 kbytes/min	393 kpixel/min
compiler_2006_06_19.ttt	89 min	1.8 Mbytes	21 kbytes/min	493 kpixel/min
compiler_2006_06_21.ttt	87 min	1.6 Mbytes	19 kbytes/min	445 kpixel/min
compiler_2006_06_26.ttt	74 min	3.6 Mbytes	50 kbytes/min	751 kpixel/min
compiler_2006_06_28.ttt	89 min	4.9 Mbytes	56 kbytes/min	917 kpixel/min
compiler_2006_07_03.ttt	87 min	4.1 Mbytes	48 kbytes/min	837 kpixel/min
compiler_2006_07_05.ttt	78 min	1.1 Mbytes	15 kbytes/min	370 kpixel/min
compiler_2006_07_10.ttt	88 min	3.4 Mbytes	39 kbytes/min	567 kpixel/min
compiler_2006_07_12.ttt	89 min	1.0 Mbytes	12 kbytes/min	306 kpixel/min
compiler_2006_07_17.ttt	86 min	1.0 Mbytes	12 kbytes/min	282 kpixel/min
compiler_2006_07_19.ttt	89 min	1.6 Mbytes	18 kbytes/min	404 kpixel/min
compiler_2006_07_24.ttt	82 min	2.5 Mbytes	31 kbytes/min	546 kpixel/min
compiler_2006_07_26.ttt	55 min	1.7 Mbytes	33 kbytes/min	667 kpixel/min
		average:	34 kbytes/min	625 kpixel/min

Table 5.4. Files sizes of recorded VNC sessions (32 bit) without keyframes

5.2.3 Summary

In summary, we can state that by means of *VNC session recording*, approximately 80–90 minutes of real live lectures, which typically consist of slide presentations and some additional dynamic content, can be preserved by file sizes of no more than 5 Mbyte. If storing content with a high *pixel variety*, larger file sizes of about 10–15 and rarely up to 30 Mbytes are produced. This is achieved by VNC's *on demand updating* approach and using the *Hextile* encoding (or other encodings) as specified by the RFB protocol in combination with applying file compression (*zlib deflate*) to the body of the recorded session. The achieved *compression ratios* for the recorded

pixel data exceed 80% if presenting content with a high *pixel variety* and reach 95–99% for other lectures that typically consist of text, sketches and graphs.

Note that these values regard the *desktop recording* only. At least an *audio stream* (for the teacher's verbal narration) and maybe a *video stream* (for a live video of the teacher) must be added. However, this is also the case for the symbolic recorders and, in fact, for any other recording environment. The integration of audio and video streams into the *TeleTeachingTool* is addressed in Section 9.6.

5.3 Random Access and Keyframes

Random access is the basis for most *navigational features*. In order to achieve *random access*, the state of the framebuffer must be computable for any point in time within the duration of the recorded session, preferably in *real time fashion*. If *random access* is not supported, a session must be replayed in total (which is very unpleasant for longer sessions) or only certain chunks are accessible. Consider an asynchronous electronic lecture, which enables access to the beginning of each slide. Accessing a slide triggers the playback of a corresponding audio stream, but this stream is always played from the start and students cannot skip parts of the verbal narration. Note that *Random access* is meant in terms of *time*, not *file*. However, if replaying a log file without copying all or most of the data into memory, random access within the log file must also be supported.

Neither storing *absolute* nor *delta pixel values* provide random access per se. In fact, the RFB protocol is hardly designed for that purpose. It rather provides *sequential* and *in order* computation of a framebuffer's content. The idea of storing delta pixel values, as suggested by [Li and Hopper, 1998b], enables processing in both directions, but still demands sequential computation.

Potentially any update message since the beginning of the recording may influence the framebuffer's content for a certain point in time. The *brute force approach* of providing the corresponding set of pixel values by computing all framebuffer updates, starting with the first logged message, proceeding up to the specified timestamp, is easy to implement, but highly inefficient. The approach will work for short recordings or if accessing points at the beginning of the session, but presumably will tend to perform poor towards the end with the increasing number of framebuffer updates that must be processed.

Consider a log file with 18 Mbytes of framebuffer updates and a duration of 90 minutes. Computing the content of the framebuffer at minute 5 must approximately respect one Mbyte of data. Accessing minute 81 would rather demand 16.2 Mbytes of framebuffer updates to be decoded. Assuming a resolution of 1024×768 pixels and the highest possible color depth of 32 bits per pixel value, a framebuffer contains no more than $786432 * 4$ byte $= 3$ Mbyte of uncompressed pixel data. Hence, in order to reconstruct the framebuffer of minute 81, at least 80% ($\approx \frac{16.2-3}{16.2}$) of the computations are performed unnecessarily. Now consider backwards visible scrolling. Skimming from minute 81 back to 77 at a rate of $\frac{1}{10}$ Hz (one frame every 10 seconds)

results in 379.2 Mbyte[8] of data to be processed. Obviously, decoding that much data in a real time fashion must fail.

5.3.1 Checking the Past

In order to improve efficiency, message decoding should be reduced to a minimum, i.e. decoding of framebuffer updates must be limited to rectangles, which provide a contribution to the required state of the framebuffer. Hence, we must compute which update rectangles previous to a given point in time are *not* covered (completely) by other succeeding rectangles (up to the given time). As the probability that an update contains relevant pixels decreases with the distance, computation should start at the access point and progress backwards until any pixel is covered by at least one rectangle (Figure 5.15). In the worst case, all rectangles up to the beginning of the log file must be scanned for relevant pixels. However, a recorded slide presentation probably will cause most of the framebuffer to be updated whenever switching slides.

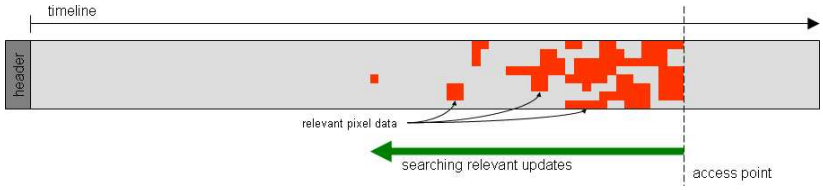


Fig. 5.15. Backward processing to acquire relevant framebuffer updates

In order to determine whether an update contributes any pixels to the desired framebuffer state demands knowledge of the rectangle positions and dimensions but unfortunately the original RFB protocol encapsulates the rectangle headers (which contain the required data) within the stream of update rectangles, which are encapsulated within the stream of (update) messages. Hence, accessing any piece of information demands parsing any previous data and would result in the same inefficiency as stated above. This is the case for the VNC client/server communication as well as for the log file formats suggested by [Li and Hopper, 1998a, Li and Hopper, 1998b].

In fact, the efficient calculation of certain framebuffer states (which is needed to provide *fast random access*) demands not only fast access to messages but also to rectangle headers (as suggested in Section 5.1.2). Accessing previous messages via file pointers [Li and Hopper, 1998a, Li and Hopper, 1998b] is not sufficient due to missing rectangle delimiters. Furthermore, messages should be kept in memory (if possible) in order to circumvent slow file I/O.

[8] 6 frames/min: $\sum_{k=77*6}^{81*6} \left(k * \frac{18Mbyte}{90*6}\right) = 6 * \sum_{k=77}^{81} k * 0.2Mbyte = 379.2Mbyte$

5.3.2 Keyframes as Check Points

Another approach to supporting efficient *random access* is the idea of *keyframes* analogous to most video formats. The recorder of [Li et al., 1999a] stores so-called *check points*, which are framebuffer updates that contain the pixel data for the entire framebuffer as is the case for the initial update message. In the case of random access, the review applications scan the log file for the closest check point previous to the intended access point. Hence, it is sufficient to progress all subsequent messages, which potentially can override some of the check point's pixels, to acquire the required state of the framebuffer.

Keyframes result in large framebuffer updates because they contain the pixel values of an entire framebuffer state. Hence, many keyframes increase the file size. Therefore, selecting a keyframe rate becomes a trade-off between access time and file size. Computing keyframes at playback time instead of storing them within the log files, does not cause larger file sizes but increases the startup time of the reviewing application.

Random access by making use of keyframes does not necessarily lead to better performance. The contribution of a keyframe to the final state of the framebuffer may be rather low. Consider a recorded slide presentation with slides and keyframes as given in Figure 5.16. Due to the large distance to the closest previous keyframe, framebuffer updates containing three slides (no. 5, 6 and 7) are decoded although each slide probably will cover most of the area. The approach of determining relevant updates may detect that 95% of the required pixels are already covered if progressing up to the closest slide, which is slide no. 7. Hence, only very few of the updates between the keyframe and slide no. 7 must be decoded (including the large keyframe update and the updates of slide no. 5 and 6). Instead of placing keyframes at a fixed rate, the updates that represent slides could be extended to keyframes, but would collide with the approach of forwarding messages instead of the less efficient solution of decoding and re-encoding all pixel data.

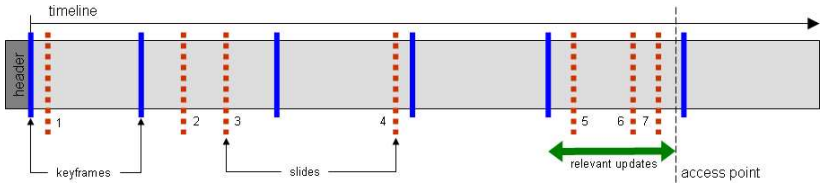


Fig. 5.16. Log file with keyframes

5.3.3 Keyframe Stripes for Invariant Frame Computation

A third approach to computing the framebuffer state for a given point in time is based on the idea of *non-incremental update stripes*, which we have introduced to counteract packet loss during transmission (Section 4.2.8). Within a certain period

these stripes cover the entire framebuffer. The period depends on the rate and size of the updates. A rate of 12 stripes per minute and 24 stripes in total results in a period of two minutes (used by the *TeleTeachingTool*). Hence, at most all updates within two minutes previous to the intended access point may have a contribution to the final state. Regarding this *invariant*, it is sufficient to decode the updates between the access point minus two minutes and the access point itself (Figure 5.17[9]). However, the effect on the file size is (almost) the same as storing a keyframe every two minutes. There may be fewer updates, because the stripes are requested from the VNC server and thus the server may have combined it with other updates. Nevertheless, this approach is useful if recording the same message stream that is used to supply online students via synchronous electronic lectures. The log file will contain the stripes anyway and therefore should be used instead of placing additional keyframes.

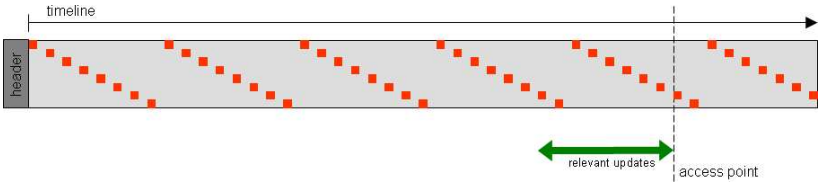


Fig. 5.17. Log file with non-incremental update stripes

5.3.4 Optimization: Current State as Keyframe

Any of the three described framebuffer state computations can use the *current* framebuffer state as a *keyframe*. Hence, accessing a point in time that lies briefly beyond the *current replay time* can be optimized. Consider skipping forward 30 seconds. As the current state covers the entire framebuffer, only those messages after the *current replay time* and up to the set *access time* can make any contribution to the required framebuffer state. Therefore, repeatedly forward skipping (and if storing *XOR pixel values* also backward skipping) at small increments (as is done to perform *visible scrolling*) is generally performed faster than accessing a distant timestamp.

5.4 Random Access Performance

Our intention is to provide *asynchronous electronic lectures* that support *random access* for any given point in time (within the duration of the lecture). *Access points* are specified in the form of *timestamps*, which have a step size of one millisecond, which commonly is the smallest increment between two timestamps. Each logged message corresponds to a timestamp, but obviously not every possible timestamp is necessarily connected to a message (otherwise there must be at least 1000 messages

[9] the figure shows only 8 stripes to be more readable

per second). *Immediate random access*, i.e. accessing without noticeable delay, demands an efficient implementation. Recall the inefficiency of the example mentioned above, which caused hundreds of Mbytes to be processed.

A limiting factor for a fast *response time* is the *access time*, i.e. the time that is necessary to compute the appropriate state of the framebuffer after specifying a certain timestamp. In order to give a prediction of the *average access time*, we have measured the *access times* while seeking certain positions within several recordings. For each recording, uniformly distributed points within its duration were accessed. The distance between two consecutive access points was set to one minute and the average of all access times per lecture was acquired. In advance to performing the general tests, we applied also distances other than one minute. Comparing the results showed that any values achieved by applying distances between one second and five minutes were almost the same, but applying higher distances caused larger variations.

During the tests the points were accessed in a *back-to-front order*, because otherwise the current state of the framebuffer (which is the final state of the previous test) would have enabled the optimization suggested in Section 5.3.4 since the following access point is placed shortly afterwards. In this case it would have been sufficient to regard only those messages between the last time set and the new access point and therefore would have resulted in faster access times. Seeking is not affected by this optimization if the requested access point corresponds to an earlier timestamp than that of the current state of the framebuffer.

The state of the framebuffer is computed once by use of the *keyframe stripes* as described in Section 5.3.3 and once by a *back to front search* of the *effective rectangles* (i.e. the contributing updates) as described in Section 5.3.1. For the *keyframe stripe* approach, any messages between the new *access timestamp minus two minutes (and five seconds)* and the new *access timestamp* itself are processed plus any necessary annotations or mouse cursor messages (which are not integrated in the keyframe stripes and thus need not necessarily be covered within the two minute period). For the other approach, the measured time includes the determination of the significant messages (framebuffer updates as well as annotation and cursor messages), the decoding of those messages and the updating of the framebuffer state.

We analyzed several courses that was recorded with the *TeleTeachingTool* throughout the last years at the Universität Trier and the Technische Universität München. All courses were recorded with a resolution of 1024×768[10], a color depth of 16 bit and include *non-incremental keyframe stripes* (24 stripes at a rate of $\frac{1}{5}$Hz and thus a 2 min period) unless stated otherwise. The durations of the listed recordings are approximately 80–90 minutes each and are listed in detail in Appendix A. All lectures are publicly available at our lecture archive http://ttt.uni-trier.de. Note that some erroneous recordings (e.g. split due to network failure) as well as the few lectures with other resolutions were ignored and are not listed in the graphs given below. All *average access times* listed here were confirmed by at least a second measurement, which proved a deviation of less than 5% (otherwise the test was repeated).

[10] Or a slightly smaller resolution (e.g. 1024×738) giving some space for control elements

The tests were performed on an AMD Athlon XP 3000+ with 1.5 Gbyte memory[11] running SuSE Linux 9.2 and SUN's Java (version 1.5.0_06). (Adapted) routines of the *TeleTeachingTool* in version 21.06.2006 were used. Note that the framebuffer was not displayed during these tests, because the graphical output routines commonly consume a lot of time (especially as the tests were performed in Java) and only the final state of the framebuffer must be displayed but no intermediate framebuffer states. Furthermore, each recorded session was load into memory beforehand.

5.4.1 Test Results

Figures 5.18 and 5.19 show the *average access time* per recorded lecture of the courses "*Informatik I*" [Winter 2004/05] and "*Informatik II*" [Summer 2005] by Prof. Dr. Helmut Seidl. Each graph presents the results for both approaches.

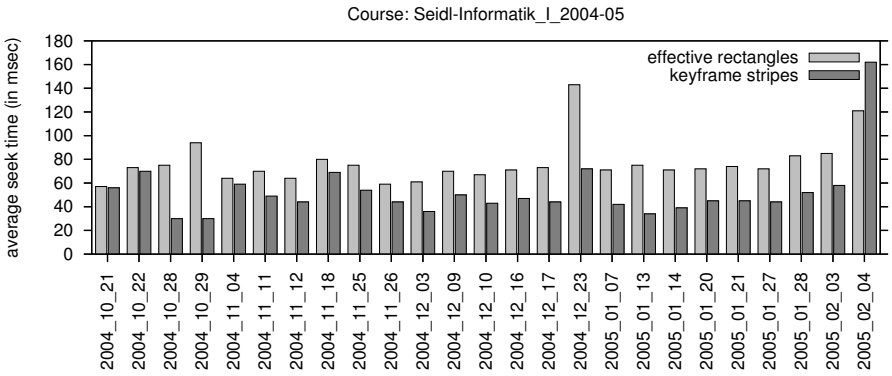


Fig. 5.18. Average access time per lecture

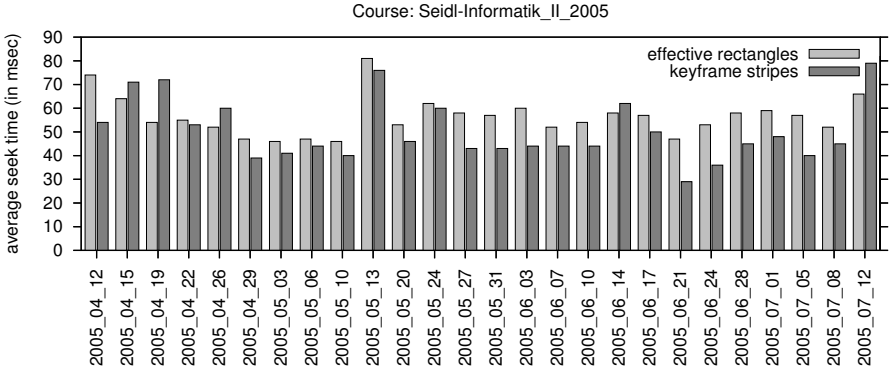


Fig. 5.19. Average access time per lecture

[11] As these are not memory intensive tests, similar results can be achieved with less memory

During the lectures mainly slide presentations were shown, partly enriched with some live programming as well as running some examples. More animated content were presented in the lectures 2004_12_23 and 2005_02_04 (both "*Informatik I*"). For most recordings, the *average access times* for the approach regarding the *keyframe stripes* are approximately 40–60 msec and for the other approach slightly higher (approx. 50–80 msec). The approach of determining the effective rectangles generally must decode fewer framebuffer updates but must check whether an update must be decoded or not (if it is covered by other updates). As the recording contains update stripes, the *effective rectangles* approach also will generally test no more than two minutes.

Analyzing the course "*Programmiersprachen*" by Dr. Alexandru Berlea (recorded in winter 2005/06, Figure 5.20) revealed an *average access time* of about 50 msec for the *effective rectangles* approach and around 10–50 msec longer for the other approach. Hence, the values are similar to those of the other two courses but this time the *effective rectangles* approach is better performing than the *keyframe stripes* approach.

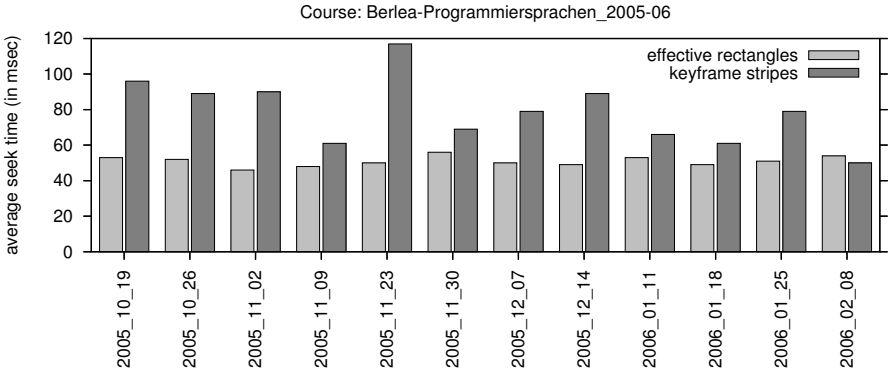


Fig. 5.20. Average access time per lecture

The results of the courses "*Abstrakte Maschinen*" by Prof. Dr. Helmut Seidl recorded in summer 2003 (Figure 5.21) and in summer 2004 (Figure 5.22) confirm the results by revealing *average access times* between 30 and 120 msec for most of the recorded lectures and a benefit of the *keyframe stripe* approach can be stated for the recordings of the summer 2003, but almost identical or slightly better *effective rectangles* values are achieved for the recordings of the other year. Hence, we cannot clearly favor either approach here.

The recordings 2003_06_25, 2004_05_19 and 2004_06_23 expose noticeably higher *average access times* of 294 msec, 397 msec and 182 msec, respectively, for the *effective rectangles* approach and 277 msec, 366 msec and 220 msec, respectively, for the *keyframe stripes* approach. Surveying the recordings revealed that during these lectures a simulator was used over a period of about 20 minutes per lecture. The VAM simulator (*visualization of Abstract Machines* (VAM) [Ziewer, 2001, VAM, 2006]) was used to visualize the memory management (stack, heap and registers) during

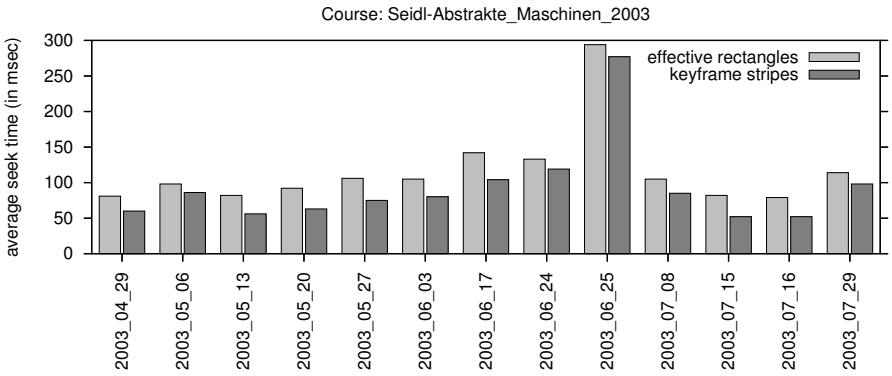


Fig. 5.21. Average access time per lecture

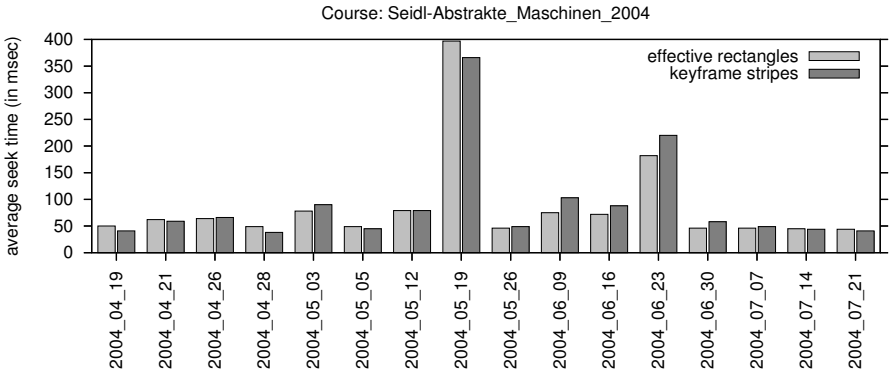


Fig. 5.22. Average access time per lecture

program executions (see Figure 5.13, page 90 to the right). Setting, copying and deleting values results in animated movements and fading effects of the objects, which represent the corresponding memory cells. Hence, 20–25% of the recordings correspond to *dynamic content* that generally causes more updates, which must be scanned and/or decoded during the tests and often result in larger files. This higher dynamics cause a higher *pixel density* of about 5-10 Mpixel/min compared to 1–2 Mpixel/min for the other lectures of those courses (the *pixel densities* are listed in Appendix A.2).

However, high *average access times* does not necessarily correspond with large file sizes. Most of the lectures of these two courses achieve file sizes of 2–3 Mbyte (about 30 kbytes/min; see Appendix A.2 for details) but, for example, the lecture 2004_05_03 reaches 6.2 Mbyte while achieving an *average access times* of 78 msec, which is also reached by other lectures with only half the file size. During this specific lecture, the simulator was also used for a period of about ten minutes, but furthermore some high colored pictures (the desktop background) were shown, which probably increased the file size due to achieving a lower compression ratio. On the other hand, the lecture with the worst average access times, lecture 2004_05_19, resulted only in a file size of 3.5 Mbyte for 77 minutes and thus a density of 47 kbytes per minute,

which is not particularly high compared to other recordings from the same series. Moreover, this particular recording has a *pixel density* of 10.3 Mpixel/min, which is about eight times the average density of all other lecture from that course. The lecture 2004_05_03 that has a file size of 6.2 Mbyte but achieves a good average access time of 78 msec has a *pixel density* of only 1.7 Mpixel/min.

In fact, surveying the *pixel density* of other courses (Appendix A) expose a correlation between the *pixel density* and the *average access time*, which is not surprising as more pixel updates cause more processing. Generally the correlation to the *keyframe stripes* approach is stronger than to the *effective rectangles* approach.

5.4.2 Different Presentation Styles

Dynamic content typically causes more framebuffer updates. However, in the case of the simulator usage the animated areas mainly consist of solid coloring and hence are easy to compress. Presentation content that shows a higher *pixel variety*, such as the lectures of the *media science* courses of Prof. Dr. Hans-Jürgen Bucher, who presents slides with many high colored images and scanned newspaper articles (e.g. Figure 5.14), can be less efficiently encoded and decoded, which results in larger file sizes (see Table 5.3 and Appendix A.2) and higher *average access times* as presented in Figures 5.23 and 5.24 although the *pixel density* is lower compared to the previously analyzed courses (0.8 Mpixel/min vs. 1–2 Mpixel/min). The *average access times* for the *effective rectangles* approach are around 100 msec, which is also reached by some of the previously analyzed recordings. The average access times of the *keyframe stripes* approach start at almost the same level but reach peaks of up to 216 msec. Generally the results of the *effective rectangles* approach are better for the recorded lectures of these two courses.

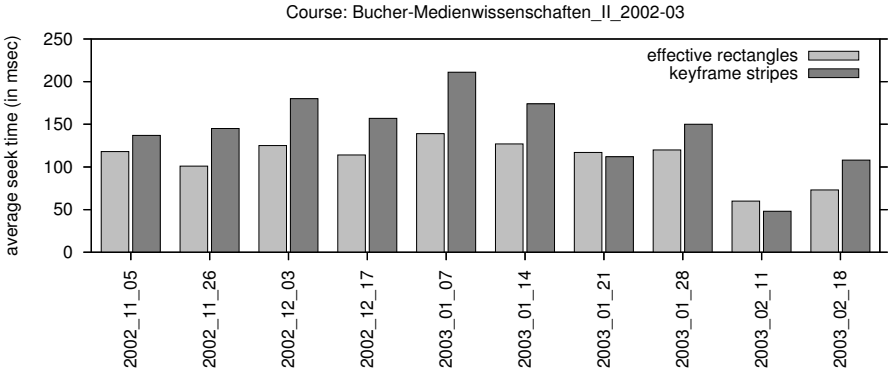


Fig. 5.23. Average access time per lecture

This is also the case for the recordings of the course "*Informatik III*" (winter 2005/06) by Prof. Dr. Johann Schlichter. Figure 5.25 reveals average access times for the *effective rectangles* approach that are almost constantly between 100 and

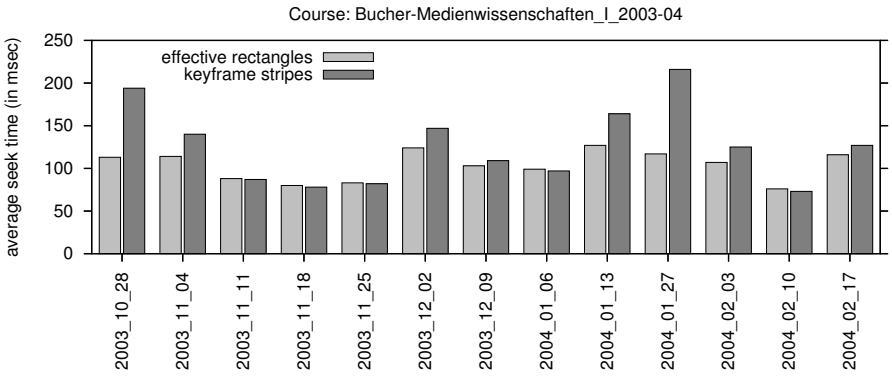


Fig. 5.24. Average access time per lecture

110 msec, but exposes approximately 50–100 % higher access rates for the other approach. Unlike the slides of Prof. Bucher with their high pixel variety, the presented content of Prof. Schlichter is more similar to that of Prof. Seidl as it consists of text, graphs and sketches as well as some dynamic simulations, but almost no high colored images. Nevertheless, the achieved *average access times* are very similar to those of Prof. Bucher's lectures. The reason is caused by the applied presentation software. Prof. Schlichter presents html based content, which is generated by *Targeteam*, a system for supporting the preparation, use, and reuse of teaching materials [Teege and Breitling, 2002, Targeteam, 2006]. The html slides are accessed and displayed with a standard web browser and are dynamically annotated by use of a Java applet. This pixel-based annotations and scrolling pages within the web browser generate more updates. However, access times of around 100 msec are still acceptable as they enable the user to access ten different points within the duration of the recording.

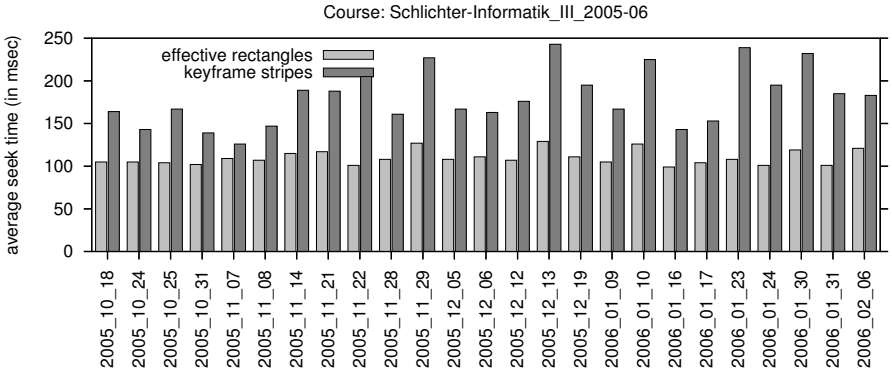


Fig. 5.25. Average access time per lecture

5.4.3 VNC Sessions without Keyframes

All of the recordings we have analyzed so far contain *non-incremental keyframe stripes*, which constitute one full keyframe at a period of two minutes. Hence, each recording contains approximately 40–45 non-incremental keyframes, which are not necessarily required for replaying the session. During summer 2006 we have recorded two courses by Prof. Dr. Helmut Seidl without these additional update stripes. Therefore the computation of the framebuffer state for a given access position can only be achieved by the *effective rectangles* approach. Unlike the previously analyzed recordings, which were recorded at a color depth of 16 bit, these two courses were recorded at 32 bit per pixel (24 bit color values stored in 4 bytes each, since 3 byte values are not supported by the used VNC server implementation) and thus contain typically twice as much pixel data (which probably does not cause twice the amount of compressed data as the unused byte is commonly always set to zero).

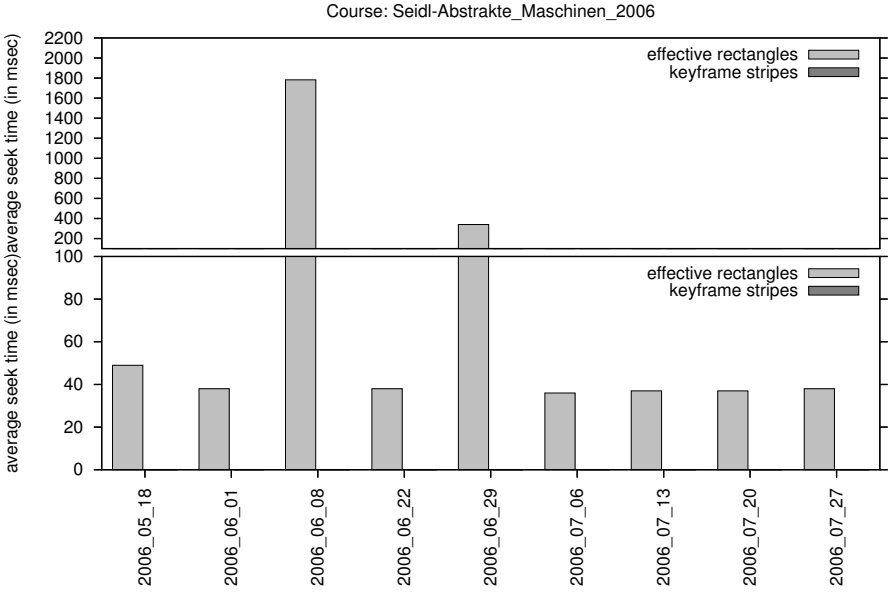


Fig. 5.26. Average access time per lecture (no keyframe stripes)

The *average access times* for the lectures of the course "*Abstrakte Maschinen*" are given in Figure 5.26 and for the "*Compilerbau*" course in Figure 5.27. The graphs reveal very good access times of about 40 msec for most of the lectures, especially if regarding the 32 bit values instead of 16 bits only. However, the graphs evidently expose some tremendously high peak values of several hundred milliseconds and even up to two seconds, which might be acceptable for slide based navigation and is still better compared to most streaming media but nevertheless is anything but a perfect access time for user interaction.

During the "*Abstrakte Maschinen*" lecture 2006_06_08 a lot of *dynamic content* (produced by the simulator) was presented and a different VNC server implementation

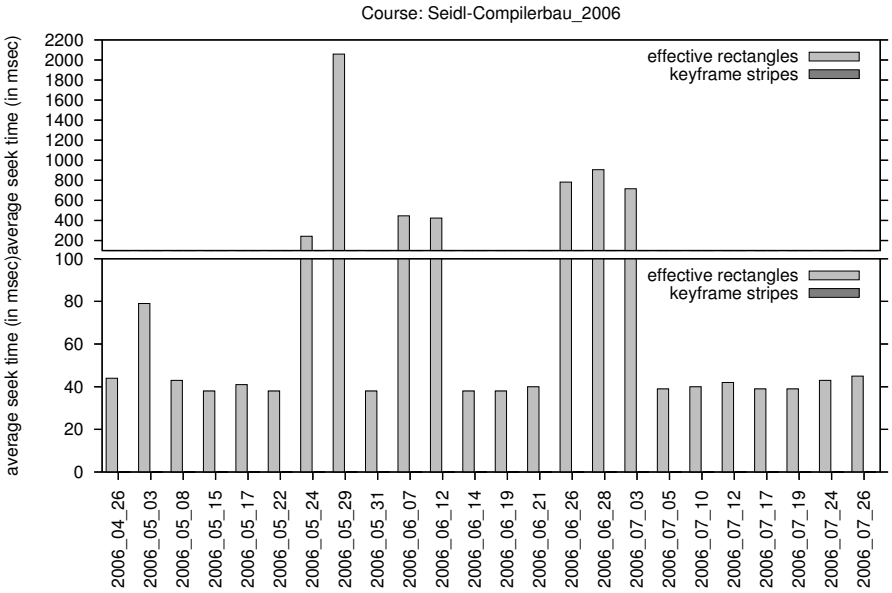


Fig. 5.27. Average access time per lecture (no keyframe stripes)

was used. That VNC server generated significantly more large updates. Other recordings of that course typically have an *average pixel density* of several hundred kpixel per minute (listed in Appendix A.3). Values between 1–2 Mpixel/min are not uncommon for recordings that contain keyframes (A.2). However, this particular lecture of the "*Abstrakte Maschinen*" course reveals an *average pixel density* of 30071 kpixel/min, which is about 70 times more than the average of all other recordings of that course and therefore we will ignore this recording as massive outlier (probably caused by a faulty VNC server).

Surveying the other recordings of the two analyzed courses revealed that during the lectures, whose recordings suffer from bad average access times, the presentation software was not switched to *fullscreen mode*. Hence, the presented slides were always surrounded by a border. The border (or at least parts of it) stayed unmodified during (almost) the complete session, which is the *worst case* for the *effective rectangles* approach as messages that occurred very early during the lecture are relevant to (almost) any later framebuffer states including those towards the end of the session. Compare the example given in Figure 5.28 against Figure 5.15 (page 95).

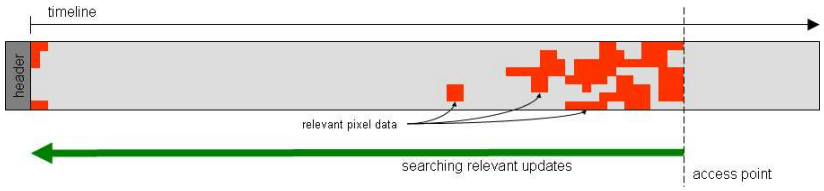


Fig. 5.28. Backward processing to acquire relevant framebuffer updates (worst case)

Note that the lecture "*Compilerbau*" 2006_05_29 achieves even worse access rates than the others, because the presented slides were not only presented within a fixed border but also not scaled to fit into that border. Therefore the lecturer often scrolled the slides and thus causing a higher *pixel density* of 2392 kpixel/min, which is about three to four times the typical amount achieved for that course. Scrolling causes large-area modifications similar to a full screen movie or animation unless the VNC's *CopyRect* encoding is applied, which we have discouraged in Section 4.2.4 in order to avoid message dependencies. The *CopyRect* encoding could be enabled for VNC session recording but then we cannot use the same VNC message stream for transmission ***and*** recording and furthermore message dependencies would have to be respected while computing the framebuffer state for random access.

The *effective rectangles* approach tests every rectangle between the access point and the first effective rectangle whether it has any effective pixels or not. In this worst case scenario, many tested rectangles offer no contribution to the final state. The test for effective pixels is currently implemented (within the TTT) as a comparison against a bit mask, which marks the already covered pixels. Hence, its performance strongly depends on the *number of tested pixels*. If presenting slides within a static (and thus unmodified) border, each slide covers large areas of the interior but possibly offers no contribution to the final state as the slide is probably covered by a later shown slide. Nevertheless the large number of pixels caused by each slide must be tested. Consider the example given in Figure 5.29. If the eight slides would have been presented in fullscreen mode without a border, it would have been sufficient to test only slide no. 8 and slide no. 7 in order to achieve a full update of the entire framebuffer.

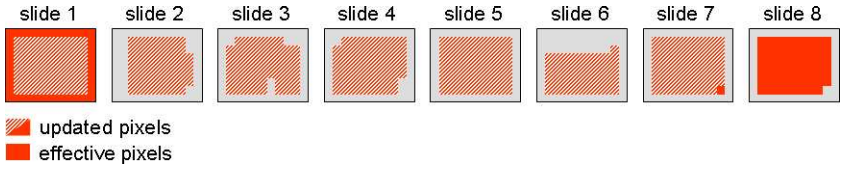


Fig. 5.29. Effective pixels of a series of slides

Although it is commonly seen as a better style to present slides in fullscreen mode, we do not intend to force teachers to do so, because this would contravene our *transparent recording approach*. Furthermore, VNC session recording is not limited to slide based presentations. The border of any other application or the task bar of the recorded desktop will probably cause similar effects. Hence, the recording and replay environment must be able to handle such recordings in an appropriate way to achieve better access times. The usage of keyframes (either as stripes or full keyframes) ensures that no messages previous to the keyframe has any influence on framebuffer states later than the keyframe. However, the *average access times* of recordings that do not contain keyframes (Figure 5.26 and 5.27) are typically better (due to them containing less updates), except for the worst cases.

5.4.4 Completing Updates instead of Keyframes

Another approach than using (full) *keyframes* is to analyse each recording and add *completing updates* whenever needed, i.e. updates that cover any pixels which have not been updated for a while. This guarantees that only a certain number of updates or a certain period in time must be tested (as is the case for recordings with keyframes) but reduces the number of redundant pixel updates. This can be achieved analogous to the *effective pixel test* but this time in a *front to back* ordering. The pixels that are modified by an update within a certain time span are marked and at the end of the period it is tested whether all pixels are updated within the period or not (Figure 5.30a). Probably some pixels are not marked and therefore addi-

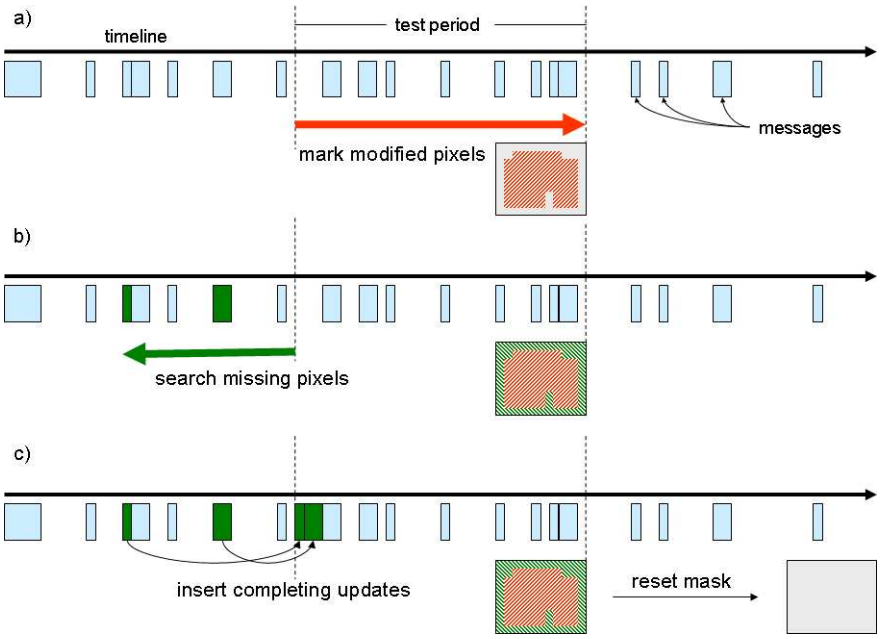


Fig. 5.30. Determining and inserting *Completing Updates*

tional messages that contain the missing pixels must be inserted. This can be done by copying the messages that contain the missing pixels from the previous period. Searching for the appropriate updates can be achieved by a *back to front* search starting at the end of the previous period and using the current pixel mask (which reveals any necessary pixels) as shown in Figure 5.30b. The copied updates must be inserted at the ***beginning*** of the current period (Figure 5.30c), because otherwise they may overwrite other messages of the period. Afterwards the pixel mask is reset and the next period will be tested. As the algorithm guarantees that any previous period already updates any pixel of the entire framebuffer, the *back to front* search will end at least at the beginning of the previous period. Due to the *initial framebuffer update* all pixels are included at the end (in fact already at the beginning) of the first period and therefore this period can be skipped during the test. Note that the copied messages may contain more pixels than necessary and therefore new

messages of smaller sizes could be composed instead. However, this would require messages to be decoded, partitioned and re-encoded rather than just copied. The analysis and modification of recordings is best suited for an *automated post processing phase*, because if performed during playback time it would increase the startup time of the replay application.

5.4.5 Optimized Effective Pixels Test

Regardless of whether adding keyframes or not, we assume that more suitable *average access times* can be achieved by improving the *test for effective pixels*. The currently implemented approach of using a *bit mask* and testing for each pixel whether the corresponding flag is set or not, highly depends on the *pixel density* of the recorded session. As updates are only applied in the form of rectangles and as the presented content also has typically rectangular shapes (e.g. slides as well as application windows) a rectangle inclusion (or intersection) test could be applied instead. Hence, the performance would no longer depend on the number of *updated pixels* but on the number of *updates* (regardless of their sizes).

Furthermore, the *bit mask* should be replaced by a more suitable data structure. For instance, images can be manipulated and accessed rather quickly if represented as *quad trees* [Hunter and Steiglitz, 1979]. As our *bit mask* is a two colored image only, the efficiency of such a data structure would improve further. *Quad trees* can be efficiently implemented from binary arrays [Samet, 1980] and work best if used to compress axially parallel rectangles, which is the case for our rectangular updates. In order to approve the given suggestions and modifications for arbitrary courses, other lectures with a higher *pixel variety* must be recorded first.

5.5 Summary

VNC Session Recording by logging timestamped RFB protocol messages was introduced by [Li and Hopper, 1998a] and extended in other works [Li and Hopper, 1998b, Li et al., 1999b, Li et al., 1999a, Li et al., 2000a, Li et al., 2000b], but mainly focuses on *sequential replay*. The concept of storing *delta pixel values* enables sequential replay in either direction [Li and Hopper, 1998b]. *Random access* is addressed by placing *check points* (keyframes) [Li et al., 1999a], but not in a way sufficient to provide fast random access. However, *fast random access* is a key feature not only to provide *visible scrolling* (Criterion C7c) but any kind of navigational (C7), retrieval (C8) or post-processing (C6) features.

The drawback of the mentioned works is the suggested file format, whose inefficiency is mainly caused by the RFB protocol specification due to missing delimiters. Accessing certain messages, rectangles or headers within the logged stream of messages always requires all previous content to be parsed, which obviously is highly inefficient. Adding file pointers as *backreferences* to previous messages is only a slight improvement. In fact, fast access to the *successive* messages is required. Therefore,

we suggest *size tags* that specify the length of the following data, which can be easily achieved by buffering single messages before writing them to the log file. The size tags also enable skipping of unknown messages and thus improve extensibility. Furthermore, we suggest storing *distinct rectangles*, which provide fast access to rectangle headers and thus improves the computation of rectangle intersections and the areas affected by framebuffer updates.

We have analyzed recorded VNC sessions for several courses during which different presentation styles were used and have shown that the suggested file format modifications lead to suitable *average access rates* less than 150 msec (typically around 40–100 msec) for most lectures. We have also made suggestions regarding how to achieve further improvements.

Furthermore, generating asynchronous electronic lectures by recording VNC sessions results in handable file sizes fulfilling Criterion C9e: File Size and Bandwidth. In fact, the log files are smaller than corresponding video recordings, but provide better quality due to lossless compression schemas. With the concept of a *compressed file format*, the files sizes are further reduced.

6

Annotations and Digital Whiteboard

Multimedia-based electronic documents offer various possibilities to annotate the presented content, for instance, by adding *textual notes*, *cross links* (within a document or to other documents), adding audio or movie clips, developing sketches by freehand drawing tools, highlighting and underlining parts to emphasize their importance, et cetera (see [Schütz, 2005] for a detailed list).

While discussing the transition from traditional to digital lectures (in Section 2.1.3) we have suggested replacing traditional blackboards and overhead projectors with digital analogons in order to preserve handwritten notes in a digital form of high quality (which cannot be achieved by videotaping [Lauer and Ottmann, 2002] or document cameras [Effelsberg and Geyer, 1998]).

With appropriate input devices, such as *electronic pens*, annotating documents is similar to writing on a blackboard or overhead slides and, furthermore, many presentation systems support at least simple annotating features. During the first lectures we have recorded with the *TeleTeachingTool*, we have used such built-in annotation features. In particular, our presentation software supported freehand drawing, which we used to underline words for emphasizing them and thus focusing the attention of the audience and also to add additional notes or sketches. While adding notes and drawing sketches worked in an almost satisfying way, the emphasizing was rather annoying because it often happened that words were crossed out instead of underlined. Deleting such a line was only possible by deleting all other annotations as well, including sketches or handwritten comments. Additionally, the annotations were limited to the presented slides since the freehand tool was integrated within the presentation software. Therefore, we decided to integrate simple but effective annotation tools within our lecture recording environment.

6.1 Annotations

The *TeleTeachingTool* presents a (remote) desktop, which offers the flexibility to present arbitrary applications and document formats. By virtually placing a separate transparent layer above the desktop presentation and adding annotations to this

annotation layer the *TeleTeachingTool* enables the user to annotate any underlying applications. Such annotations are *vertically layered* [Schütz, 2005] and thus are totally independent of the application (since the desktop is presented as pixel image). *Horizontally linked* annotations would rather be integrated into the presented document [Schütz, 2005], which requires access to the presented documents and thus is typically limited to certain document formats only.

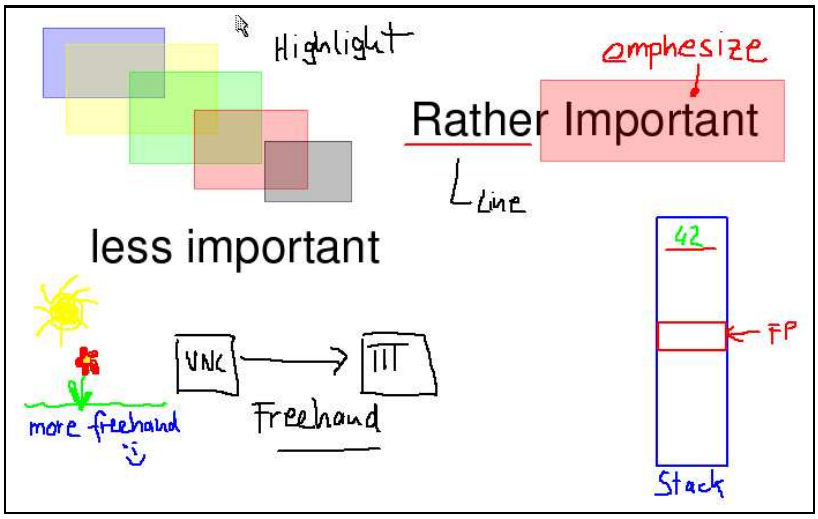


Fig. 6.1. Annotations of the *TeleTeachingTool*

Our *annotation tool* should at least support *freehand annotations*, because this is the natural replacement of handwritten blackboard notes or annotating overhead slides. Furthermore, a support for focusing the attention of the audience must be given. Our experiences showed that a (maybe enlarged) *mouse pointer*, which seems to be a natural replacement for a traditional pointing device, such as a stick or a laser pointer, is not suitable. Unlike the mouse pointer, a traditional pointing device is typically used to point to a certain element and than will be removed. Typically the mouse pointer will always be visible. This might be preferable because the focused element will stay emphasized. Consider, for instance, a traditional presentation during which the presenter points rather briefly to a topic by use of a laser pointer, which might not be noticed by some people in the audience. However, a mouse pointer is inappropriate for permanently focusing of elements, because in general the presenter is used to the mouse pointer in such a way that she/he will often not be aware of the pointing functionality due to the everyday usage of the pointer in a common desktop environment. Hence, the pointer very often will not point to elements by intention, but rather point to arbitrary positions. Because of this and our unsatisfying experiences with the freehand annotations as an emphasizing tool, we rather suggest the use of explicit *emphasizing annotations*. Additionally it is useful to be able to remove certain of the previously applied annotations in order to make corrections, for example while drawing sketches, or to remove all annotations to clear the screen. In summary, we want to provide the functionality of freehand drawing, emphasizing and removing of annotations.

The *TeleTeachingTool* (TTT) currently supports the following annotation types, all of which can be applied in several colors (except deletion):

annotation	description
freehand	a line of connected dots
line	a straight line
rectangle	a rectangle drawn parallel to the axes
highlight	a translucently filled rectangle
delete	erases certain annotations
delete all	removes all annotations

The graphical visualization of *TTT annotations* is shown in Figure 6.1.

The annotations are not handled *pixel-based* as part of the framebuffer but are specified via coordinates and colors and are *represented symbolically*. *Pixel-based* annotations could be integrated into the existing protocol either by placing rectangles that exactly cover the affected pixels (which is easy for horizontal or vertical lines but rather complex for freehand drawings) or must respect and thus re-encode the current framebuffer content. Besides the rather complex computations needed and the higher bandwidth and larger file sizes caused by pixel-based storage, it is impossible to remove annotations (unless we provide a backup for overwritten pixels or request the area in a non-incremental fashion). In contrast, the *symbolic representation* of annotations produces less data (typically a few bytes per annotation only, see Section 9.7 for the specification) and applying annotations on a separate layer without affecting the framebuffer that represents the desktop, enables more flexibility in editing annotations and especially supports the removal of annotations. The deletion of *annotation objects* is performed by choosing the *delete mode* and selecting the appropriate annotation that should be removed, i.e. clicking on the visual representation of the annotation object.

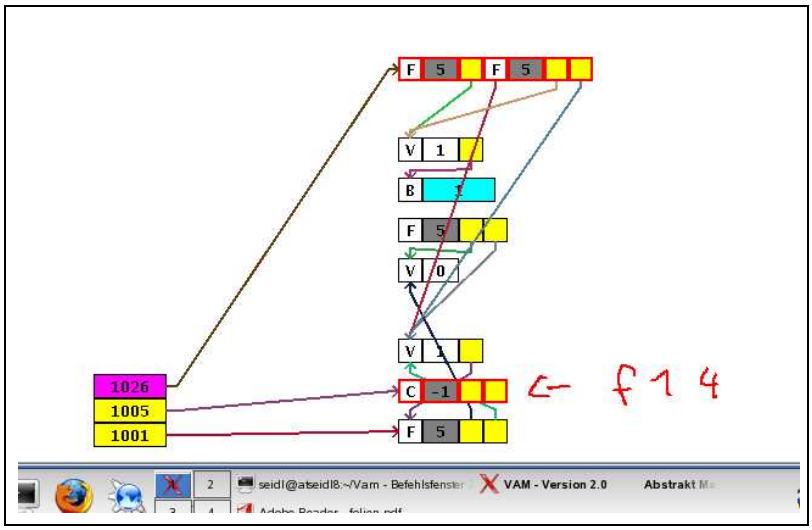


Fig. 6.2. Annotating a dynamic desktop

Unlike the annotating feature of the screen recorder *Camtasia* [Camtasia, 2006], our annotations are not applied to a *static screenshot* of the desktop but to the *dynamically* updated desktop. Although it is obviously not meaningful to annotate a moving object, for example the moving cells of the *VAM Simulator* we have already mentioned in Section 5.4.1, it can be useful to annotate *static* elements, for instance the program code or certain (currently) stationary cells of the simulator as shown in Figure 6.2.

6.1.1 Annotation Controls

[Schütz, 2005] divides *parallelly divided* and *integrated* annotation controls, where *integrated* means that the *annotation controls* are part of (integrated into) other control elements of the presentation software. As our screen grabbing approach is independent of any presented applications, the *TeleTeachingTool* must use *parallelly divided* controls, i.e using an own *tool bar* as shown in Figure 6.3.

Fig. 6.3. Annotation Controls

The buttons of the *annotation controls* correspond to the following functionalities (left to right): enabling/disabling the annotating mode, choosing one of several colors, choosing one of the annotation tools (freehand, highlight, etc.) and removing all annotations. By choosing the appropriate annotation tool (and color) a teacher can annotate slides and any other presented material *dynamically* during the lecture.

6.2 Digital Whiteboard

Beside annotations, our system offers a *digital whiteboard* which provides a *blank page* that can be displayed on demand in order to give additional space for annotations. Consider a lecturer who presents slides, but sometimes uses a traditional blackboard in order to describe certain issues, maybe because students have asked. In order to preserve such excursions in an *electronic lecture*, they must be digitizable in a suitable quality. Offering annotations alone might not be sufficient. Although a teacher may leave some free space upon her/his slides in order to place annotations during the presentation, more space may be needed to explain unforeseen students' questions. Adding *blank pages* offers additional space.

Another possibility would be to switch back to the *edit mode* of the presentation software and insert an additional blank slide. However, this would disrupt the presentation. In contrast, the *whiteboard* of the TTT can be enabled whenever needed and, after the excursion has finished, the teacher can switch back to the originally presented slide with a single button press (the outmost right button in Figure 6.3).

Since it is often useful to recall the annotation made during the excursion, the *whitboard* and the *desktop* have different *annotation layers* and thus independent *sets of annotations*. Hence, the teacher can switch between annotated slides and annotated whiteboard excursions. Moreover, the TTT supports not only a single *whiteboard* but rather *whiteboard pages* with individual annotations for each page.

6.3 Protocol Integration

In order to transmit and record the *annotations* they must be integrated into the *adapted RFB protocol*. *Rectangles* (including *highlighting* rectangles) can be specified by the x,y coordinates of one corner and the width and height, *lines* by specifying two points and *freehand annotations* by a list of points. Additionally, the color must be noted. The removal of annotations could be achieved by assigning a *unique ID* to each *annotation object* and explicitly removing objects by their ID. However, we have chosen a very simple but pragmatical solution: Since removing is performed by virtually wiping out the objects with the electronic pen, the corresponding annotations can be removed by specifying the *removal points* (which correspond to the pen movements while wiping out) and deleting any annotations that intersect with this points.

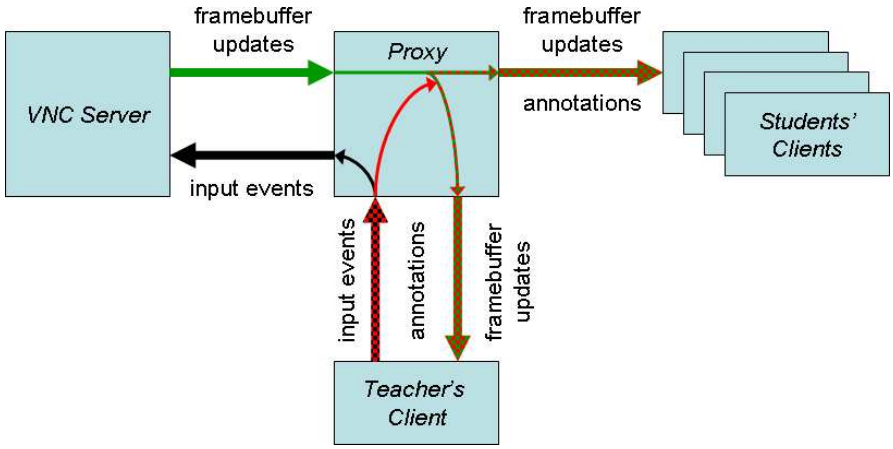


Fig. 6.4. Data flow between TTT components

The integration into the protocol is achieved by defining new *message types* with appropriate values for each of the annotation types as specified by the *TTT protocol description* given in Section 9.7. Furthermore, these new *annotation messages* must be delivered to the students' replaying applications. Recall that our *proxy* receives messages from the *VNC Server* and processes them (i.e. transforming, recording and transmitting) and additionally handles the *input events* sent by the teacher's application in order to remotely access the presented desktop. Annotations are produced by the *teacher's application* as well and should be transferred to the *students' client* and to the recording component. Therefore we suggest delivering *annotation*

messages from the *teacher's application* to the *proxy* analogous to the *input event messages* as given in Figure 6.4. Unlike the *input events*, the *annotations* are not sent to the *VNC Server* but are merged with the *framebuffer update messages* received by the *VNC Server* and then transmitted to the *students' clients* or delivered to the *recording component*. As stated in the figure, the *teacher's application* is also supplied with the *merged framebuffer update and annotation stream*. As the *teacher's application* is the producer of the annotations, this would not be necessary. However, such a design enables all clients to be supplied with the same data.

Note that our design still uses an original *VNC Server* implementation, which respects the original *RFB protocol* and thus is not aware of the annotation concept. However, this is not necessary since the *proxy* filters any annotation messages.

Recording of TTT annotations is achieved in the same way as logging *framebuffer updates*. In fact, merging the additional *annotation messages* with the stream of *framebuffer updates* is sufficient for recording, because the *recording component* will automatically timestamp and log each message of the incoming message stream, including the annotations.

6.4 Evaluation of Annotation Usage

A complex evaluation regarding pedagogical issues and learning results of students could not be performed throughout this thesis. Nevertheless, we can give a short summary of how the annotations were used during our lectures and how this affected the dynamics of electronic lectures.

We have analyzed 85 recorded lectures by Prof. Dr. Helmut Seidl, which were recorded during four semesters, regarding the usage of our annotation tools [Ziewer and Seidl, 2004]. We have analyzed the *number of applied annotations* per lecture, distinguishing between the four *annotation types*: *freehand*, *line*, *rectangle* and *highlighting*. Since the annotations are recorded as individual message types, counting the occurrences is easily established by automatically analyzing the message headers for each recorded lecture. Note that a *freehand annotation* is counted as a single occurrence from setting the electronic pen on the display, i.e. producing the start point, until lifting the pen again, but the writing of a word or drawing a sketch typically consists of several individual *freehand annotations*. Hence, the number of occurrences of *freehand annotations* cannot be directly compared with the number of occurrences of the other annotation types, which exactly reflect single uses, for instance to highlight a word.

The *annotation tool* was used for the first time during winter 2002/03. The lectures of that semester reveal approximately 180–280 annotations per lecture and the usage increased to 200–400 annotations per lecture in the following semesters. The number of annotations that were used per lecture are given in Figure 6.5, where the left values relate to the oldest and the right to the newer lecture dates.

The most frequently used *annotation type* was the *highlighting annotation*, which was used about 130–230 times per lecture (Figure 6.6). *Highlighting annotations*

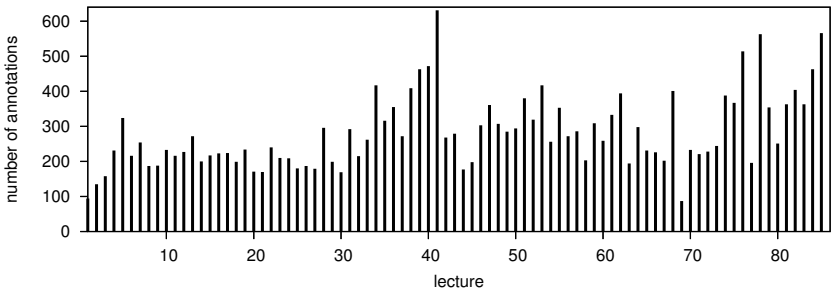


Fig. 6.5. Annotations per lecture (rect,line,high,free)

have been accepted by the teacher almost since the very first lecture and the usage stayed relatively constant throughout the years.

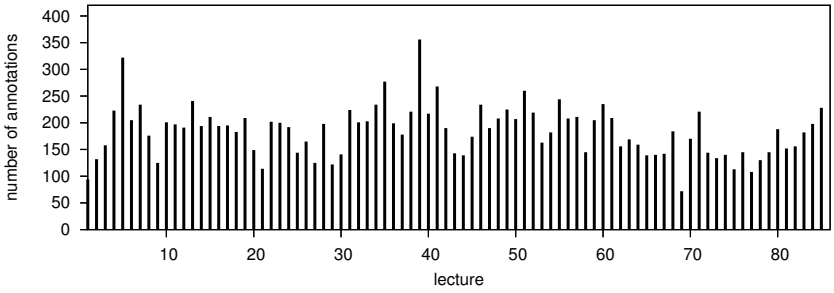


Fig. 6.6. Highlighting annotations per lecture

Analyzing the *freehand annotation usage* reveals different results as shown in Figure 6.7. During the first semester (the first 27 analyzed lectures in the figures), Prof. Seidl rarely used *freehand annotations* (less than 20 occurrences per lecture). Since the second semester he started using *freehand annotations* more frequently and there is a slight increase in the last semester.

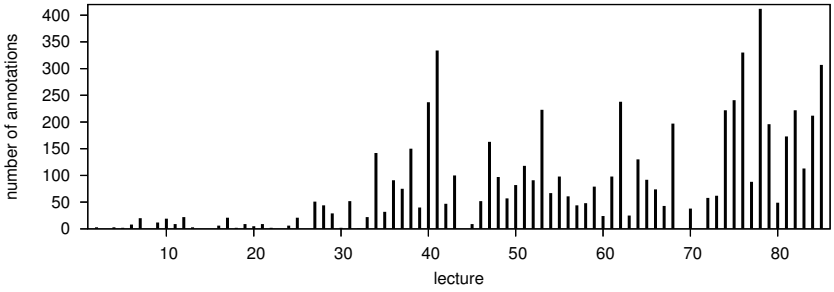


Fig. 6.7. Freehand annotations per lecture

The possibility of *line annotations* was hardly ever used and the *rectangles* were drawn about 10–50 times per lecture. However *rectangle annotations* were often applied to draw a rectangle around some text in order to emphasize it. Hence, *highlighting annotations* could have been applied instead.

Furthermore, we have analyzed how often the color was changed between two consecutive annotations. This feature was barely used during the first lectures, the teacher started using it more frequently after about two months. Overall he switched to another color about 30–50 times per lecture in order to distinguish different annotations by using different colors.

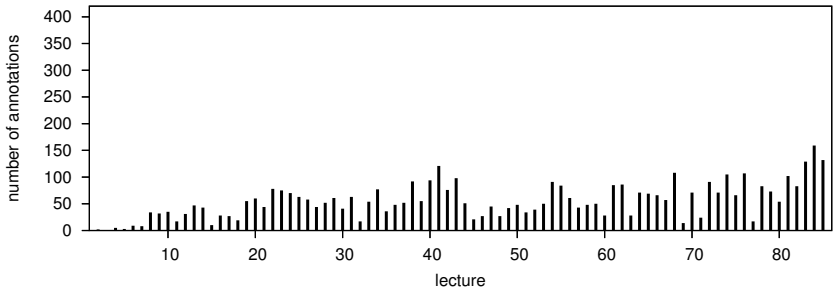


Fig. 6.8. Color changes per lecture

Although more recordings of different teachers must be analyzed in order to give a general statement, these results give an impression of how *TTT annotations* potentially can be used. Regarding these results we can state that for Prof. Seidl the *highlighting* and *freehand annotations* are of most importance. The *rectangle annotations* can often be replaced by *highlighting annotations* since they were both used with the intention of emphasizing. Although Prof. Seidl used the freehand feature of the presentation software to underline certain words for emphasizing purposes *before* we introduced the TTT annotations, he rarely used the line feature later. Thus, we conjecture that applying *highlighting annotations* is much more effective.

Students' comments concerning the *TeleTeachingTool* revealed that some of the students complained about too many *highlighting annotations* in the lectures of Prof. Seidl. An extreme case of a slide with many highlighting annotations is shown in Figure 6.9. Almost every element was emphasized during the talk but typically only the last one is still valid for focusing purposes. Therefore, the previously made *highlightings* should rather be deleted. Since an *explicit deletion* by the teacher would be an additional task to be performed, which should be avoided. Instead we suggest automatic removal of previously made *highlightings*. This can either be realized by limiting the number of simultaneously visible *highlighting annotations* to one or maybe up to three annotations at a time, or by setting a timeout for such annotations so that they will automatically disappear after several seconds. Considering the implementation of the *TeleTeachingTool* the first approach is preferable, because it can be realized by modifying the *TTT teaching component* (by sending appropriate *delete annotations*) while keeping full compatibility for the student component. This modification will affect only newly produced *electronic lectures*. If the *highlighting*

annotations of previously recorded lectures should be limited as well, a modification of the replaying engine is inevitable. Since the *highlighting* annotation type might have been used for drawing purposes during previously recorded lectures, for instance to draw boxes, the feature of automated removal should stay optional, i.e. students should have the option to disable that feature.

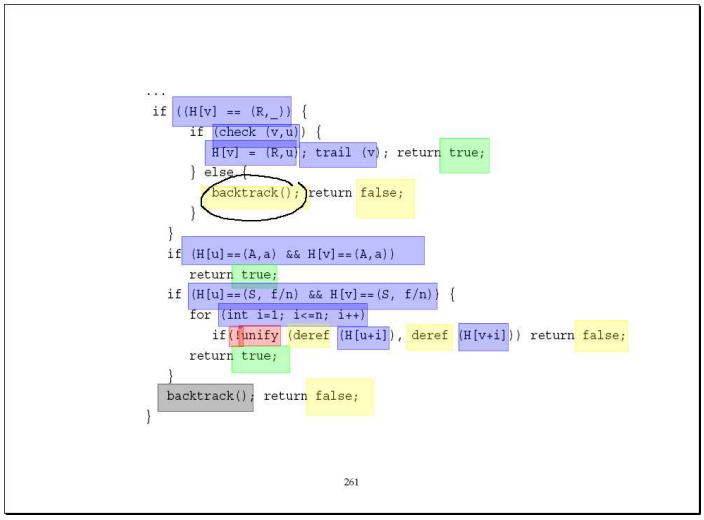


Fig. 6.9. "Over-highlighted" slide

In summary, the most frequently used *annotation feature* is *highlighting* in order to focus the attention of the audience. In order to strengthen the effect of this feature, the number of simultaneous *highlighting annotations* should be limited in order not to lose the focusing effect. For any other commenting of the presented content the *freehand drawing feature* is of highest importance. Considering the usage statistics, the other annotation types can be neglected, but the use of *different colors* is helpful in order to distinguish the applied annotations. Regarding the intention to provide an *easy to use* and *concise* control bar that offers access to simple but effective annotation tools, *emphasizing* (with automated removal), *freehand drawing* and *deleting* in combination with a few colors might be sufficient.

6.4.1 Dynamics of Lectures

Presenting a static slide over a longer period is rather problematic from a pedagogical point of view [Edelmann, 1995], especially if the teacher is only visible in a small video (if at all) as is the case for *electronic lectures* that are presented on screen. Therefore, we have additionally analyzed the recordings regarding their *dynamics*, i.e. any visible modifications of the presented lecture including slide switches, animations, annotating or just moving the mouse pointer. We have classified periods as *dynamic* or not according to the *time spans* between consecutive modifications. *Time spans* up to 3 seconds were classified as *dynamic*. The lectures of Prof. Seidl show

a proportion of *dynamic periods* of about 15–35%. If regarding only the *dynamics* caused by the desktop (i.e. the framebuffer updates) without considering annotations the proportion decreases to approximately 10%, if using other applications besides the slide presentation (e.g. the VAM simulator) about 20% are achieved.

We have compared those values with the *dynamics* of 28 lectures of Prof. Dr. Hans-Jürgen Bucher, whose lectures were also recorded with the *TeleTeachingTool*. Instead of the *TTT annotation system*, Prof. Bucher has used the built-in freehand drawing feature of the applied presentation software. The *dynamic proportion* of this lectures reaches only 5–15%

Hence, the *dynamic proportion* of (electronic) lectures can be significantly increased by use of the simple annotating features of the *TeleTeachingTool*. Even higher *dynamics* are achieved whenever additional applications and especially animations or visualizations are used during the presentation. However, adequate evaluations must be performed in order to state the effect of *TTT annotations* and *dynamic proportions* regarding the learning results of the students.

6.5 Summary

The *TeleTeachingTool* offers an easy to use *simple* but *effective annotating functionality*, which is suitable to focus the attention of the audience as well as to add additional comments and sketches and furthermore increase the *dynamic proportion* of *electronic lectures*. By offering an *electronic whiteboard*, the teacher can add blank pages *on demand* in order to give additional explanations whenever questions arise during a lecture. After such an excursion the presenter can switch back to the presentation, which stayed unchanged during the excursion, including previously made annotations.

TTT annotations are not limited to the presentation software but can be applied to any presented applications and documents, because they are applied on a *separate layer* on top of the pixel-based desktop representations. This design can be extended to *multiple layers* in order to support several *sets of annotations* as suggested by [Schütz, 2002, Lienhard and Lauer, 2002, Lienhard and Zupancic, 2003]. They suggest *annotation layering* in order to enable student annotations. By providing several virtual layers for the teacher's and the student's annotations, and maybe for additional annotations of other students, each set of annotations can be enabled or disabled on demand, i.e. certain annotations can be shown or hidden.

The annotation and whiteboard features are integrated in our adapted VNC environment by *defining additional message types*. The *recording component* automatically *timestamps* each logged *annotation message* and thus enables a *dynamic replay* of annotations as requested by [Lauer and Ottmann, 2002] (our Criterion: C4d). Another criterion (C4c; also requested by [Lauer and Ottmann, 2002]) is that annotations should be associated with slides and disappear when switching to another slide and appear again when switching back. Typically such an association requires an integration of annotations into the presented document (*horizontally linked* annotations

[Schütz, 2005]), but our annotations are applied *vertically layered* and independent of the underlying content in order to support the annotating of any applications. We diminish this drawback by automatically removing all annotations whenever the teacher switches to the next or previous slide by pressing one of the keys commonly used to switch slides, i.e. the arrow keys and the page-up/-down keys. Hence, new slides will always be presented without the annotations of the previous slide. In most cases, this approach is sufficient. An association with slides to ensure that annotations can appear again corresponding to the presented slide is more challenging. Nevertheless, we will address this topic and suggest solutions in Section 7.7 (after discussing some necessary prerequisites).

7

Navigation and Automated Indexing

Lectures and presentations are *sequential* and so are their video style recordings, the *asynchronous electronic lectures*. Students who have not attended the corresponding *live lectures*, e.g. distance students or local students that missed a lecture, may watch the electronic version in full length. But in general pure *sequential playback* is not sufficient. Students rather want to access certain topics and thus like to locate and study parts of special interest, which could be the beginning of a chapter or a specific definition. They may want to replay interesting or complicated sequences but skip other parts.

However, as standard screen recording does not conserve the document structures (of presented slides), only *sequential playback* or at most *navigation by time* is possible. Although such *timeline navigation* is a useful feature, for instance to skim through a lecture or to skip back a little bit to replay the just seen part once more, it is often not the best way of navigation. 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 caused by buffering techniques (e.g. 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. Recall the tedious search of a certain song on a music tape or a passage on a video tape, always winding the tape 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 the identification of the target sequence, but also the identification of 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 given predetermined *table of contents*, which exactly refers to meaningful positions within the timeline of the media. In order to improve *navigational features* for lecture recordings, *indices* are required as *navigation marks* addressing special positions within a recording. Any *timestamp* and thus any recorded data associated with a timestamp is a *potential index*, but a recorded VNC session commonly consists of thousands of timestamps. However, *meaningful indices* are only those addressing points of interest, like the beginning of a chapter or an animation as well as switching to another slide, which is called *navigation by slide*.

In this chapter we will first give an overview of *navigation by time*, including requirements and suitable *graphical user interfaces* (GUI). Then different *classes of navigational indices* are described and discussed in order to elaborate an approach to acquire meaningful *navigation marks*. Afterwards, we introduce and discuss different approaches of *automated slide and animation detection* to be performed during *post-production* as well as *on the fly* during a live lecture. Additionally, we discuss the *visual representation* of indices as well as the automated generation of *annotated scripts*. Furthermore, suggestions are made as to how *TTT annotations*, which are independent of the presented content by default, can be *interlinked* and *associated* with slide indices and presentation content.

7.1 Navigation by Time

Navigation by time or *timeline navigation* is navigating during replay by specifying a certain (access) point in time within the duration of the recorded session. The most common user interface to specify the *access point* is a *slider* as shown in Figure 7.1. The slider represents the *timeline* of the lecture. The left point (starting point) relates to the beginning of the lecture and the right point to the end and thus the duration of the lecture. As the representation is straight proportional, any slider position in between relates to a position within the duration of the lecture. Selecting a time is done by clicking a certain point on the slider or dragging the slider knob to the desired point and releasing it (depending on the slide implementation). This provides *random access* to the specified access point on the timeline. If the screen is updated in relation to the knob position while still dragging the knob, which is called *visible scrolling*, the user can skim through the lecture and thereby get a better overview. If *visible scrolling* is not supported, the displayed media will be updated according to the end point (i.e. the release point) only. Of course *visible scrolling* is preferred but is not/cannot be supported by all players and media formats. *Streaming media* generally cannot support visible scrolling "... because only a small part of the document is buffered locally at any given time. In order to make random access work in real time, the document as a whole must be stored locally" [Lauer and Ottmann, 2002].

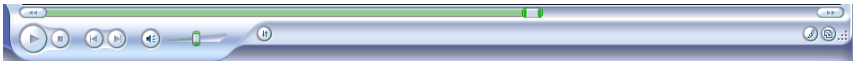


Fig. 7.1. Controls of the WindowsMediaPlayer including a timeline slider

A slider is a very comfortable user interface but not necessarily a very exact one. The number of *selectable positions* and thus the *possible access points* is limited by the number of pixels corresponding to the length of the slider. For the common screen resolution of 1024×768 pixel the length is approximately 900 pixels and thus the smallest increment is theoretically six seconds which relates to one pixel. However, if the slider is shorter or if the slider implementation provide only larger step sizes, the smallest increment might be bigger. Morever, setting the slider to an exact position is not that easy. If *exact positioning* is demanded, e.g. for editing purpose, the user

interface should provide an additional input field, which enables the user to specify the point in time by entering the corresponding time value (i.e. by entering digits).

Another useful feature for *navigation by time* are buttons or keyboard shortcuts, that can be used to *skip back or forth* a certain amount of time (e.g. -5 min, -1 min, +1 min, +5 min). This enables *relative* navigation instead of *absolute* positioning. Often students just want to skip back a little bit in order to replay the just seen but not totally understood part of the lecture or they want to skip parts they already have understood. For instance some introduction to a topic might be clear to the student but certain details that are discussed afterwards might not be and therefore the student wants to skip the introduction and progress directly to the detailed discussion.

A crucial factor for the usability of *navigation by time* is the *response time*. In fact, a *fast response time* is also beneficial for other *navigational features*, however the user might be willing to wait a few seconds if accessing a certain slide or performing a search, but not if *navigating by time*, especially not during *visible scrolling*. This is because navigating by time is commonly composed of several *navigational steps*, for instance, clicking several positions on the slider until the searched position is found. Or consider a student pressing the +1 min button several times and then recognizing that she/he skipped slightly beyond the desired time and hence pressing the -1 min button once. If *random access* demands time consuming computing or buffering, as is commonly the case for streaming media, *navigation by time* is less useful and *visible scrolling* without *fast random access* is meaningless. Unfortunately, the *RFB Protocol* [Richardson, 2005] is not designed to support *random access* by default but can be appropriately adapted (Chapter 5). In particular, we have shown that (average) *access rates* for *random access* of less than 200 msec can be achieved and for typical slide based presentations even access rates of less than 50 msec are possible as shown in Section 5.4. As the *response time* of the *graphical user interface* correlates to the *access rate* we can presume *fast random access* and low *response times* for VNC based recordings.

7.2 Navigational Indices

[Minneman et al., 1995] divide four broad classes of *navigational indices*. [Li et al., 2000a] applied these classes to the context of VNC session recording. They use a VNC proxy recorder that not only stores timestamped *framebuffer update* messages but also records *user event* messages (e.g. key events) (see Section 5.1.3), which also can be used as *indexing marks*.

7.2.1 Intentional Annotations

Firstly, there are ***intentional annotations***, which are indices the teacher creates during the presentation especially for the purpose of indexing, i.e. to mark particular points of interest. Highlighting a text string on the desktop leads VNC to automatically create a *ServerCutText* message, which can be used as index. However, the

teacher has to remember to create such indices during presentation, but she/he may forget to do so in the heat of the moment. Consider intentionally indexing headlines. The teacher must remember to highlight the headline of each slide in order to generate a *ServerCutText* message and thus copy the textual string to be included in an overview or table of contents. This approach influences the live presentation without any positive effect for the local audience, in opposition to the *transparency* recording approach, and therefore is not very commendable.

Additionally, the VNC client of [Li et al., 2000a] offers the possibility of inserting brief notes which are stored as *ClientCutText* messages to fit into the RFB protocol. If used during a lecture, this feature may interrupt and distract the natural flow of the presentation. The explicit insertion of brief notes is better suited for post-processing and, in this case, should be filed under *post-hoc indices* (see below).

On the other hand, *intentional indices* can be placed automatically. *teleTASK* [teleTASK, 2006], a pixel-based recording system, uses a plug-in for *PowerPoint*, during the presentation to generate an index each time the teacher switches to another slide. This allows slide-based navigation. Unfortunately, this feature demands the use of *PowerPoint* and if teachers forget to start the plug-in no such indices are recorded and there is no way of applying the plug-in afterwards.

7.2.2 Side-effect Indices

The second class are ***side-effect indices***, which are activities whose primary purpose is not indexing but provide indices because these activities are automatically timestamped and logged. Hence, *side-effect indices* are recorded without any impact on the presentation process and therefore are well suited for our *transparent recording approach*.

Examples are *input events* such as keystrokes, pointer movements and button presses, which occur while the teacher interacts with the VNC desktop. [Li et al., 2000a] suggest logging *KeyEvent*, *PointerEvent* and *Bell* messages. Their idea is to generate indices of such events because users may remember them and want to access the corresponding part of the recorded session. Obviously, this idea is only meaningful to users who have attended the live session or, at least, have watched the recorded session before. Especially the *Bell* message type, which signifies that something happened and commonly results in a short sound (the signal bell), is only meaningful if the user remembers that such an event occurred and knows to which topic it refers. However, Li et al. have a different scenario in mind. Instead of recording live lectures (where the signal bell is typically inaudible to the audience), they rather intend to record desktop sessions in order to demonstrate software usage.

Nevertheless, their studies revealed that "pointer events are hardly referred" [Li et al., 2000a] and thus provide not very interesting indices. Especially the events caused by moving the pointing device are commonly irrelevant, because each movement generates dozens of such events and therefore an entire session may contain several thousand in total, for instance, if the pointer device is used to annotate slides.

On the other hand, there might be teachers who control their slide presentations by use of key presses only and thus generate sessions without any pointer events.

[Li et al., 2000a] suggest the logging of *key events* in order to enable *retrieval* of inserted text. Furthermore, the inserted text may be used the generate a *textual index*. A *sequence of key events* results in (or can be combined with) *textual content* that is searchable. Since at most only small texts are entered during a presentation, the benefit for later text search is rather limited. In the scenario of *lecture recording*, special keys are often used for special purposes, e.g. the *page down* key to switch slides. Logging such keys would deliver good indices. However, as most presentation software applications offer several possibilities to switch slides (e.g. mouse buttons, the page-up/down keys, the arrow keys or the b and n key) it is almost impossible to generate a good index table on the basis of these events alone. Moreover, the used keys may be application dependent and therefore must be defined prior to recording and must be used in the expected way, which leads to similar problems to those of *intentional annotations*. Hence, we cannot rely on such events and therefore they are not sufficient to provide navigation within (asynchronous) electronic lectures on their own, but at least they might be useful as *additional indicators*.

In fact, we discourage the logging of key events due to *security issues*. As we log desktop sessions and allow the usage of arbitrary applications, the teacher potentially may enter a password in order to connect to a remote computer or a protected web page. Unfortunately the entered password, although not visible on screen, would be logged in the form of *key events*. If the *key event log file* (either a separate file or combined with the framebuffer updates) is handed to students, they potentially may access the teacher's password(s). In the case of generating a textual index, which consists of any text phrases entered during a lecture, even the unmasked password is displayed. Admittedly, the way of storing the key events in a database utilized by [Li et al., 2000a] is more secure, but nevertheless the benefits for retrievability are rather limited as typically only small text passages are inserted, e.g. shell commands. Note that *full text search* is by all means a very reasonable feature but must access the slide content instead of the *key events* (see Section 8.1).

Besides the already mentioned *intentional* placement, *ClientCutText* and *ServerCutText* messages also provide *side-effect indices* when used for the original purpose of copying text (i.e. clipboard content) between session partners. Note, that due to the double usage of these two message types, it is not distinguishable, if they were created by *intention* or as *side-effect* only. Hence, any marked text may occur as a headline in a table of contents (suggested by [Li et al., 2000a]) and thus possibly irritates students.

7.2.3 Derived Indices

Another class of indices are ***derived indices***, which are produced by *automated analysis* of recorded data. [Li et al., 2000a] suggest analyzing framebuffer updates as "a big screen change may suggest that an application window has been opened or closed" [Li et al., 2000a]. Considering our intention of recording slide presentations, such big screen changes also exposes slide changes.

Post-processing enables the tweaking of parameters to achieve optimal results. New or improved index analysis can be applied to all recordings and so even very old lectures benefit from research advances. Additionally, as the *derived indices* are calculated after the presentation is finished, the creation cannot be forgotten during the recording process.

7.2.4 Post-hoc Indices

Finally, there are ***post-hoc indices***, which are *manually* created during *post production* to mark points of interest. If placed with consideration, for example to mark definitions or the beginning of chapters, they are very useful. Insertion of keywords to describe chunks of the electronic lecture, as suggested by [Brusilovsky and Miller, 2000], are also filed under *post-hoc indices*. Unfortunately, manual placing of indices is very time consuming and thus undesirable for a *lightweight* lecture recording process.

However, students may annotate and comment an electronic lecture (for instance as described by [Lienhard and Lauer, 2002, Lienhard and Zupancic, 2003]) which is also a post editing process but not a mandatory one. Such student notes provide *post-hoc indices* with a special meaning to the student who placed it and, if the comments can be exchanged among students, may also be useful for fellow students.

7.2.5 Index Querying

The VNC session recorder of [Li et al., 2000a] stores all events in an *event store* [Spiteri and Bates, 1998], which allows retrieval by use of SQL[1] queries. The query

```
select * from frame-buffer-update where update >= 40%;
```

leads to all framebuffer update events that change 40% or more of the screen. Entering queries in SQL syntax is obviously not very user friendly. Therefore, [Li et al., 2000a] provided some browser interfaces for common queries. For the given example they suggest a field labeled "update" and inserting the value 40 results in the SQL query given above. However, the meaning of the "update" field and the possible values must be explained to the user. This approach may be useful for *advanced users*, but is not very intuitive, especially not for *novice students* with less computer practice. Hence, most students will probably not use such features. A major problem of the system of [Li et al., 2000a] is that the student must decide, which are meaningful values for the queries. A structured overview, as claimed to be necessary by [Lauer and Ottmann, 2002], would rather give (a list of) *predetermined* meaningful indices with some kind of classification (e.g. indices that refer to slides) and the student must only select *which* of the indices she/he wants to access. Hence, we need to *determine indices* and *classify* them and finally present them in the form of an

[1] SQL (commonly expanded to Structured Query Language) is a computer language to interact with relational database management systems

intuitive user interface in order to provide *meaningful* and *easy to use* asynchronous electronic lectures.

7.2.6 Automated Analysis

Regarding the *lightweight* content creation approach, the generation and classification of indices should/must be *automated* as far as possible. Hence, the *manual* placement of ***post-hoc indices*** may be offered for *additional* index generation or to place *optional* student comments, but is unsuitable for *automated* processing. The use of ***intentional annotations*** is also discouraged, because it opposes the *transparency* of the session recording as it demands teachers to think about useful indices and keywords *during* the live presentation. In terms of *transparency* only *side-effect* and *derived indices* provide an appropriate way of structuring electronic lectures. As a (VNC session) recording consists of thousands of *potential indices*, we need *automated analysis* to *classify* them and acquire a *meaningful* selection.

Regarding the classification, it is better to find fewer but consistently meaningful indices instead of presenting all the meaningful indices along with many useless ones ("*false positives*"). Consider a recorded slide presentation with 30 slides. If the review environment presents, for instance, 27 or 28 indices, each referring to a real slide switch, the indices are well-suited for navigation and will satisfy most users. However, if their are 50 slide indices, 30 of which refer to the beginning of a slide, but 20 referring to arbitrary and meaningless points in time, the user might be confused about the meaning of the indices and wonder why an index provides access to the middle of nowhere. Hence, the classification algorithms should deliver only meaningful indices at high probability.

Note that the analysis and the suggested algorithms and thresholds of the following sections are closely related to the *TeleTeachingTool* and TTT recordings (because it is the basis implementation for this research). However, many aspects can be transformed to other *VNC session recorders* or even to *screen recorders* in general. The main difference between VNC recorders and other screen recorders is the *message based* and *demand-driven* data recording. Pixel values are encapsulated within *framebuffer updates* and thus we have sets of rectangular screen modifications. Furthermore, the *demand-driven approach* in combination with the *timestamp message logging* provides some useful *metadata*, e.g. *what* was updated and *when*, and therefore enables a better classification than recording (full) frames at a fixed frame rate. Furthermore, the VNC *input event message types* as well as the TTT *annotation message types* offer the possibility of identifying user events, which is hardly possible just on a pixel basis. Note that several presentation software applications provide their own *annotation systems*. The resulting annotations are recorded by the TTT (and all other screen recorders), but are encoded *pixel-based* as part of the framebuffer in this case, overwriting other pixels and hence cannot be extracted in a straightforward manner. On the other hand, TTT annotations, which are stored symbolically, can be filtered and thus analysis can be applied just to the framebuffer. This is also the case if the VNC server supports the special *cursor encodings*, which handle the mouse cursor on a separate layer and thus independent of the framebuffer content. Additionally, all TTT recordings (unless recorded with a TTT since version

21.06.2006) contain *non-incremental update stripes*, which are used as partitioned keyframe parts as suggested in Section 5.3.3. Other VNC session recorders may use full keyframes (like [Li et al., 1999a]) or no keyframes at all. (Standard) video formats that are used by other screen recorders commonly make use of keyframes as well.

7.3 Slide Detection

Navigation by slide is one main feature a playback engine should offer [Lauer and Ottmann, 2002]. 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 data*. Switching slides during a presentation commonly results in a modification of large areas of the screen. In our 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. "Large" can be seen in the meaning of *by byte*, i.e. the length of messages, or *by-area*, i.e. the area of the framebuffer which is affected by an update.

7.3.1 Slide Detection By-Byte

The computation of *slide indices* in terms of *large by-byte* is straight forward if logging VNC messages with corresponding timestamps and the length of each message (Section 5.1, Figure 5.3). Otherwise the entire log file must be parsed once in order to determine message lengths. A classification of message sizes can be achieved by applying an *empirically determined threshold*. Any framebuffer update whose size exceeds this ***slide detection threshold*** is classified as a slide.

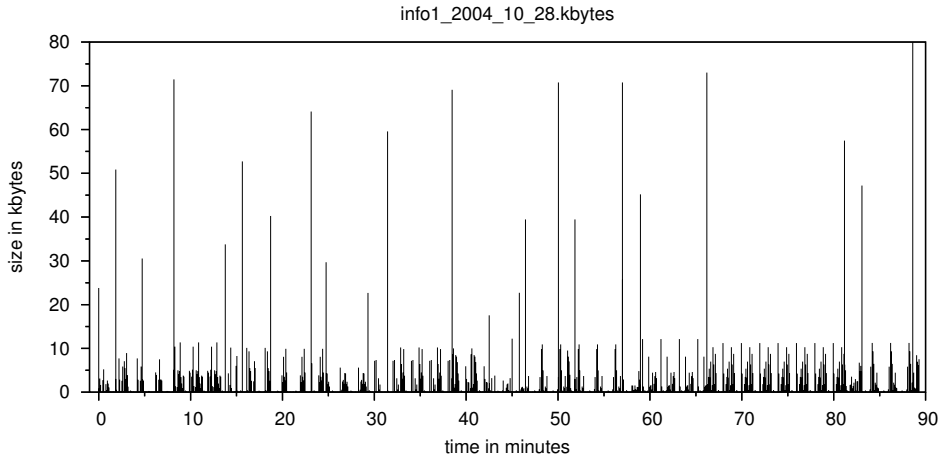


Fig. 7.2. Message sizes

Figure 7.2 shows the message sizes in kbytes of each lecture of the course "*Einführung in die Informatik I*" [WS2004/05] of Prof. Dr. Helmut Seidl, recorded at the Technische Universität München at a resolution of 1024×768 (or slightly less to keep space for control elements during recording) pixels and 16 bit pixel values[2]. The large messages denote slide switches. The smaller ones (see Figure 7.3 for a zoomed graph) are mainly caused by the periodically appearing *non-incremental update stripes* which are used as (partitioned) keyframes (Section 5.3.3)[3]. If the framebuffer content stays

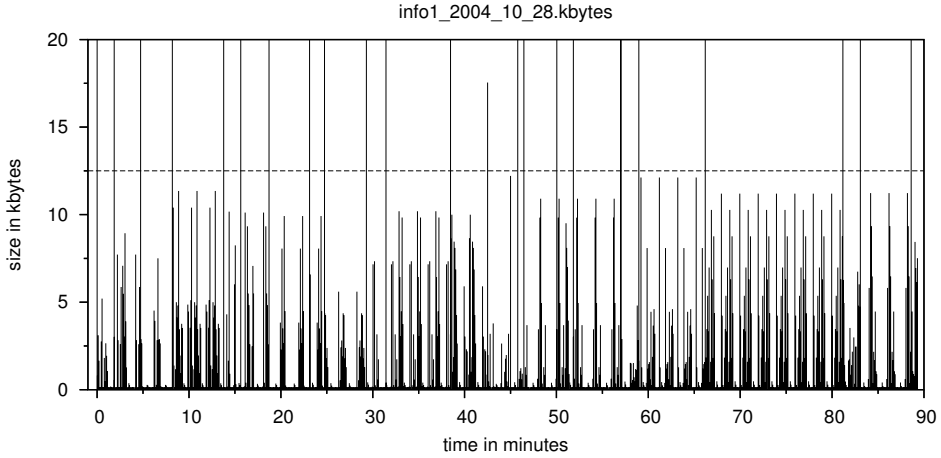


Fig. 7.3. Message sizes (zoomed)

unmodified for a while (e.g. minute 70–80), the graph evidently exposes the period of two minutes after which (almost) identical stripes are transmitted once again. The update messages, which are caused by these stripes, must not be taken into account when computing indices. However, the RFB protocol does not offer a possibility to distinguish these *non-incrementally* requested stripes from other framebuffer updates, which were requested in an *incremental* fashion, because only the requests have an appropriate flag but not the update messages. One could try to regard the rectangle coordinates to test if an update covers the requested stripe or not. However, the VNC server is free to combine or split updates (depending on the server implementation) and therefore this approach may fail. Hence, the stripes must be eliminated by setting the *slide detection threshold* to a larger value than the largest stripe message. For the given example recording, a threshold of 12,500 bytes eleminates all messages caused by stripes. A closer look at the recording reveals that 23 of 24 slides are detected applying that threshold which corresponds to a hit rate of at least 95%. The one elided slide appears at minute 45 and is only 12,212 bytes in size due to containing a few words (and thus a few pixels) only (as shown in Figure 7.4) and therefore achieves a good compression ratio. As the largest messages caused by the update stripes are 12,122 bytes large (min 60–65), reducing the threshold to

[2] this and all other mentioned recordings are freely available at our lecture archive http://ttt.uni-trier.de

[3] Recordings stored by the TeleTeachingTool up to Version 20.06.2006 contain non-incremental update stripes (12 or 24 stripes at a period of 2 min.)

include the missing slide is discouraged, because this may lead to false detections if different content causes slightly larger stripe messages. Recall that we favor dropping a meaningful index against generating many *false positives*, i.e. meaningless indices.

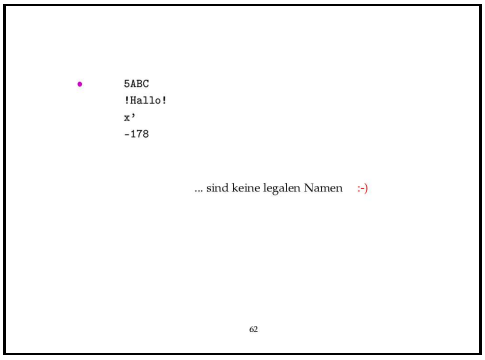


Fig. 7.4. Undetected slide

Applying the same *slide detection threshold* of 12.500 bytes to other recordings from the same course revealed similar results and hit rates of above 90% (see Appendix B.1). The undetected slides contained only a few words (as the example given above), small sketches or (series of) graphs as shown in Figure 7.5.

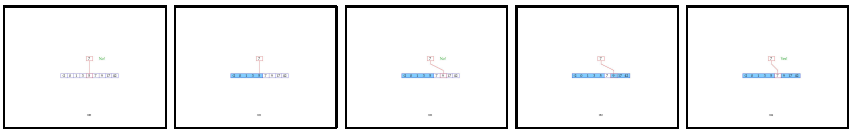


Fig. 7.5. Series of undetected slides

Our previous research [Ziewer, 2004] revealed a *slide detection threshold* of 10,000 bytes to be suitable for recordings with a lower color depth of only 8 bit and a resolution of 1024 × 768, which is the case for the recorded courses "*Informatik I (Programmierung in Java)*" [WS2001/02] by Prof. Dr. Helmut Seidl recorded at the Universität Trier and "*Abstract Machines for Compilers*" [SS2002] presented by Prof. Dr. Helmut Seidl (Universität Trier) and Prof. Dr. Reinhard Wilhelm (Universität des Saarlandes).

We noticed that the slide detection algorithm sometimes generates multiple indices for the same slide. However, these are no faulty detections, but a result of the teacher's presentation style (during that particular course). Slides were annotated using the built-in drawing features of the presentation software. Deleting annotations caused a redrawing of the screen and thus resulted in additional framebuffer update messages, which the algorithm detects as a slide change. In fact the presentation software showed the same slide two (or more) times in a row, exceeding the threshold each time and therefore such slides are detected twice (or more often). At least the indices determined here give meaningful partitions of such slides as removing previous annotations certainly lead to breakpoints within the presentation.

Limitations

Slide detection by message sizes works well for many lectures by Prof. Seidl. However, we encountered problems applying the same *slide detection threshold* to other lectures (with the same resolution and color depth). This was especially the case with the course "*Medienwissenschaft I: Theorien und Methoden*" [WS2003/04] of Prof. Dr. Hans-Jürgen Bucher, recorded at the Universität Trier. This media science course was complex in its graphical representation. The extensive usage of large high colored images (mainly scans of newspapers and magazines) and figures leads to a higher *pixel variety* (i.e. many different pixel values instead of solid coloring) and thus to larger message sizes (due to less efficient compression) as shown in Figure 7.6 for one example lecture (others are listed in Appendix B.2).

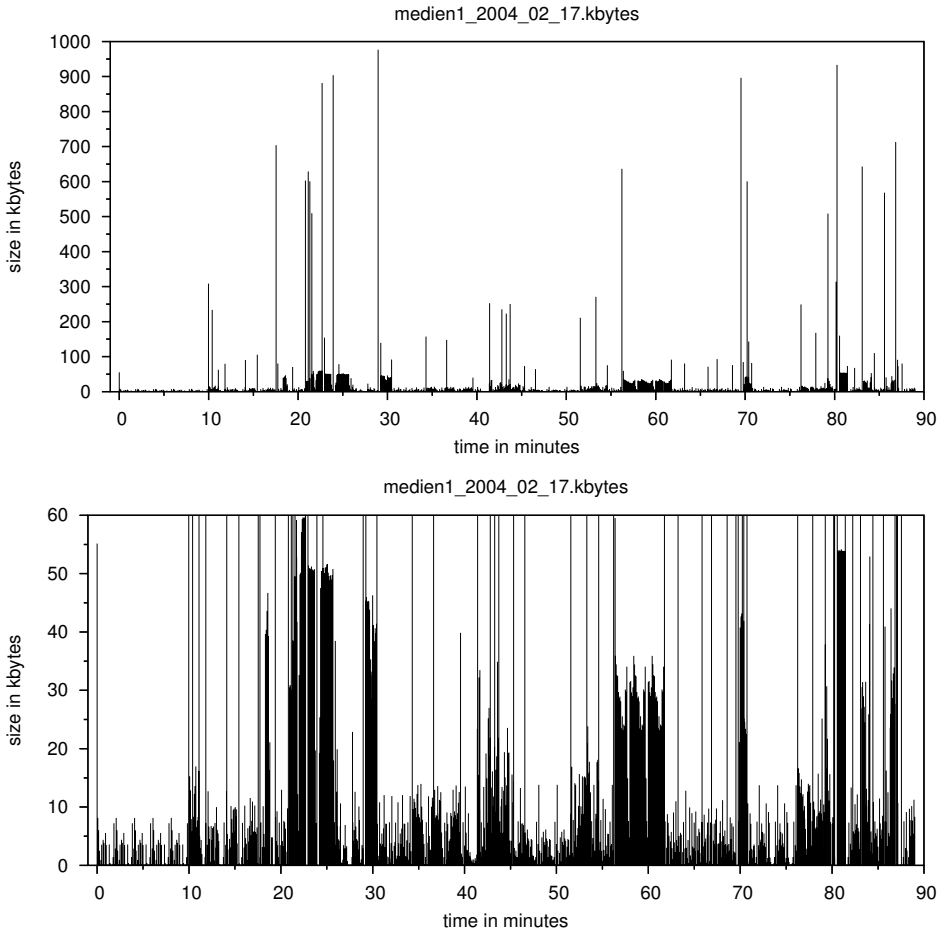


Fig. 7.6. Message sizes (full scaled and zoomed)

For that particular lecture, the values of the first ten minutes look very similar to those seen previously. Again, the two minute period of the non-incremental update

stripes is evident. Surveying the range between minute 56 and 62 also reveals the two minute period but consists of much larger messages which partly exceed 30 kbytes. This would suggest increasing the *slide detection threshold* for the given lecture (or course) in order to exclude the update stripes again. However, the recording contains even larger update stripes. The peak of 903,990 bytes at 23:55 min. was caused by the slide shown in Figure 7.7 to the left. The slide included two newspaper scans and therefore does not achieve very good compression ratios as we apply lossless compression to obtain better quality. Some of the following update stripes (containing parts of that slide) exceed even 50 kbytes in size, but the next slide (right side of the figure), appearing at 25:55 min., is only 38,478 bytes in size. This slide, like many others, consists mainly of text, which can be compressed rather efficiently. Hence, increasing the threshold to filter large stripes would also mean dropping all indices for text-only slides. This is not only the case for the given example recording but also for other recordings of that course (see Appendix B.2).

Fig. 7.7. Slides with scanned newpaper (904 kbytes) and simple text (38 kbytes)

In fact, the approach of detecting slides by the sizes in bytes of the framebuffer updates is sufficient for *homogeneous* recordings, i.e. when all slides show similar content (in terms of structure not semantics) and thus are of similar size in bytes. For *homogeneous* recordings the *slide detection threshold* could even be computed dynamically based upon the *peak values*. However, this approach is almost certain to fail for *inhomogeneous* presentation content. Such is the case for that particular media science course. Its presentations mixed slides containing only some text, followed by others showing huge colorful illustrations. Slide detection on a *size by-byte* basis for such presentations is not sufficient, because a fragment of a colorful illustration can be larger than a complete slide containing only text. This is even more severe if mixing *incremental* and *non-incremental* updates as is the case for most of our recordings. However, only partial non-incremental framebuffer updates (stripes) are requested and the affected area of such an update is less than five percent of the desktop resolution (if the framebuffer is partitioned to 24 stripes[4]). Switching fullscreen slides, in contrast, results in framebuffer updates that affect (almost) the complete desktop. Hence, the approach of determining slide switches regarding the area that is covered by framebuffer updates is more promising.

[4] some of the older recordings contain a partitioning of 12 stripes

7.3.2 Slide Detection By-Area

Computation of indices regarding the area that is covered/affected by the framebuffer updates for each given timestamp mainly consists of computing rectangle unions and intersections. This can only be achieved in an efficient way if the *rectangle headers* are directly accessable within the recorded sessions (as suggested in Section 5.1.2) or if the rectangle positions and dimensions are cached preliminarily (e.g. by parsing all update messages once). As most VNC server implementations deliver framebuffer updates with distinct rectangles, it is possibly sufficient to sum up only the rectangles' dimensions not regarding overlapping. However, the protocol specification does not ensure distinct rectangles. The number of pixels are accumulated for each given timestamp and then translated to *relative values* in relation to the maximum number of pixel values, which is specified by the resolution of the framebuffer. This is not possible for the other approach since *by-byte* values cannot be transformed to relative values due to a missing maximum. However, *relative values* are more suitable to achieve a meaningful *content prediction*, because they are independent of the number of bytes used per pixel (color depth), the screen resolution, the applied encodings or the encoded data, i.e. the presented content.

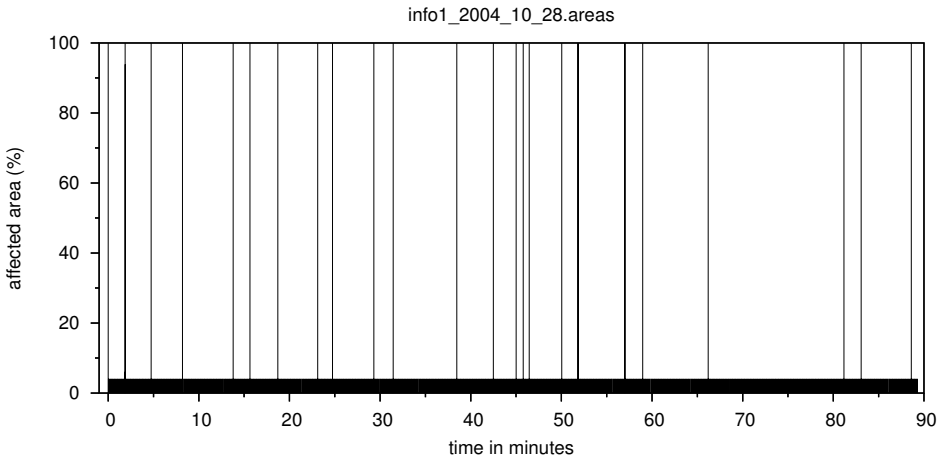


Fig. 7.8. Affected areas (i.e. message sizes *by-area*)

Figure 7.8 displays the affected areas for the same lecture as examined for the *by-byte* approach (compare with Figure 7.2) and reveals perfect matches. Each of the 24 slides are detected and each slide covers 100% of the framebuffer, even the one which was ignored by the *by-byte* approach due to its small size. The bottom line[5] is caused by the *non-incremental stripes*, which cover $\frac{1}{24} = 4.1\overline{6}\%$ each. Unfortunately, we cannot assume we will receive such perfect matches for all recordings. The main reason for the perfect matches in this case was the VNC server implementation used, which evidently generated entire updates instead of applying a fine-grained partitioning

[5] It is actually not a line but distinct values at a frequency of $\frac{1}{10}Hz$, which only appears as line due to the scaling of the graph

to the framebuffer. However, other VNC implementations probably will. Although not analyzed in detail, we observed that the *Linux* implementations generally send fewer but larger rectangles and the *Windows* implementations tend to generate very many small update rectangles instead.

For the example lecture of the media science course the affected areas in comparison to the message sizes are given in Figure 7.9. Evidently, the problematic large update stripes are all reduced to $4.1\overline{6}\%$ and thus can easily be filtered. However, setting the

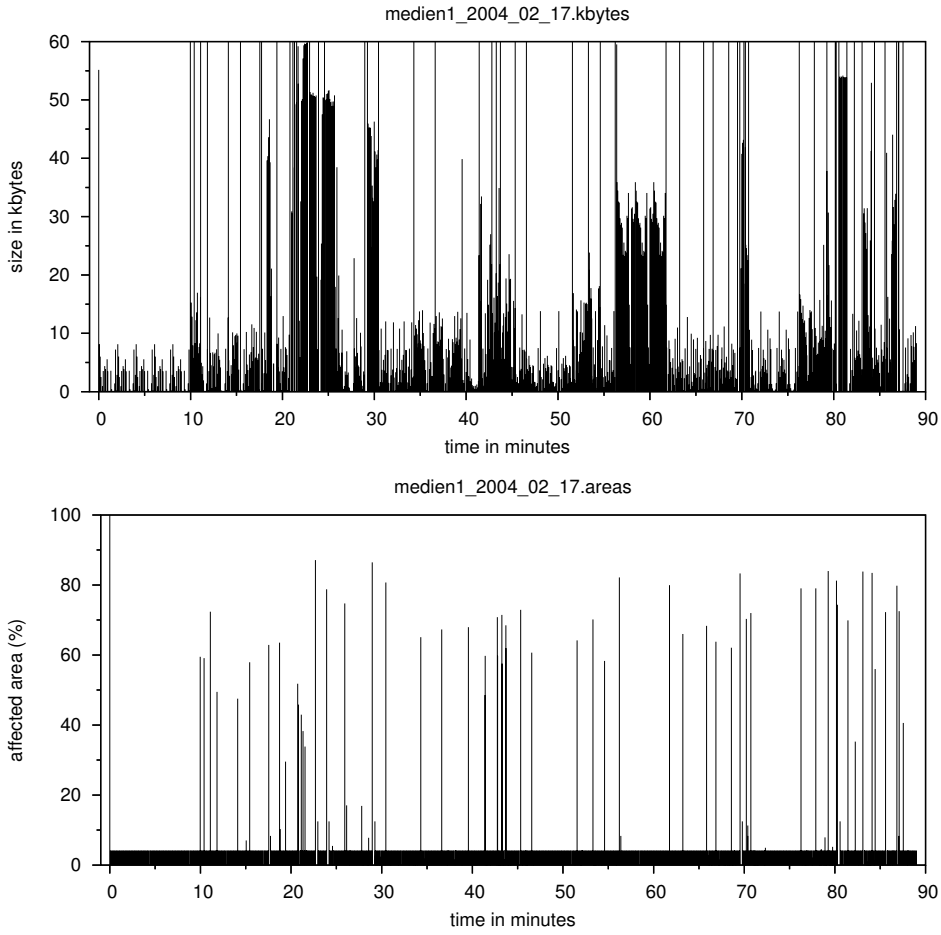


Fig. 7.9. Message sizes (zoomed) vs. affected areas

by-area slide detection threshold to a value slightly exceeding the *stripe filter* of $4,1\overline{6}\%$ ($\frac{1}{12} = 8.\overline{3}\%$ for older TTT recordings), generates too many indices. In the zoomed view of the graph (Figure 7.10 to the left) some values between 5% and 20% are visible. The corresponding framebuffer updates were either caused by slide overlays, for instance at minute 15:02 (7%) and 17:44 (8.3%), or by placing freehand annotations, which were recorded pixel-based using the freehand drawing feature of

the presentation software (min. 24–30; 12–18%). This is also the case for the small updates between minutes 70 and 80. Detecting slide overlays may be meaningful unless a slide is partitioned to a large number of overlays, but freehand annotations should not cause *slide indices* (although the detection of annotations can also be a meaningful feature). Hence, we must increase the *by-area slide detection threshold* to 20% in order to filter annotations, but at cost of some slide overlays. Nevertheless, the first slide (to which the overlays are applied) will still be recognized, because it is commonly larger as it must overwrite (most of) the previous framebuffer content.

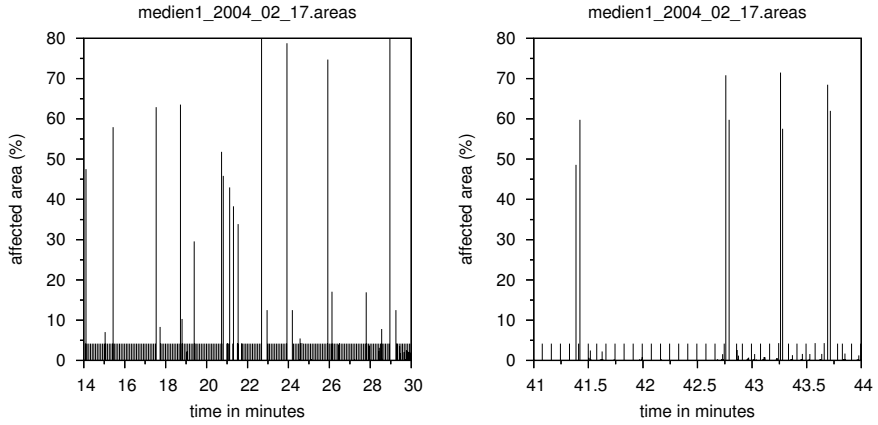


Fig. 7.10. Affected areas (zoomed)

Another phenomenon is visible in the detailed view at the right hand side of Figure 7.10. There are four occurrences of framebuffer updates that affect at least 45% of the framebuffer and which are followed by another large update within a very short interval of less than 2 seconds. Surveying the recorded session reveal 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. Even if the previous index was caused by another slide, dropping that index is unproblematic, because the corresponding slide only appeared for a few seconds and therefore that particular slide is obviously not that important. Consider the teacher skipping a slide or hitting the "next slide" button (e.g. a mouse button) by mistake and switching back immediately. Showing a slide for less than five seconds will hardly be meaningful. Hence, applying a *slide detection threshold* to derive *potential slide indices* and afterwards applying a ***slide drop threshold*** of five seconds (as delay between consecutive indices) to drop overruled *potential slide indices* is reasonable to classify *slide indices*. Commonly, slides are presented much longer than five seconds unless they contain overlays or sequences of slides, which may be used to explain certain issues. For instance the slides shown in Figure

7.5 are used to explain the "binary search" algorithm in a step by step fashion. It is a matter of opinion whether each overlay should be counted as single slide or not. Slide overlays could be accumulated to one slide index by increasing the *slide drop threshold* to 10 or 15 seconds. Unfortunately, increasing the *slide drop threshold* would result in the last of the overlays to be accessable rather than the first one. Therefore, it is probably better suited to treat such slide sequences as *animations* (Section 7.4). This is also the case when recording other applications unless they are used to present static pages (e.g. a web browser presenting slides in the form of html pages).

7.3.3 Whiteboard Pages

Besides presenting slides by use of presentation software, the *TeleTeachingTool* offers the possibility of inserting *blank pages* on demand (*electronic whiteboard*; see Section 6.2, page 114). The *whiteboard* is enabled and disabled by a special message, which is rather short (a few bytes only) and thus would not be classified as an index by the *by-byte* approach. However, *whiteboard notes* are of special interest as they are often used to explain a certain topic in detail or on demand if asked for by students. Moreover, *whiteboard notes* are commonly not part of a published set of slides or a script. Fortunately, *whiteboard indices* can easily be integrated whenever a corresponding message enables the whiteboard. Likewise, an additional *slide index* is inserted whenever the whiteboard is turned off again. The same result is achieved if a *whiteboard message* is treated as a *framebuffer update* that covers 100% of the screen.

7.3.4 Conclusion

With the *by-area* approach and the empirically determined threshold, *slide indices* for pixel-based recordings can be generated in a fully automated manner. The use of *relative* values instead of *absolute* ones as the basis for a content prediction algorithm leads to much better results, because relative values provide some kind of abstraction from the presented content. Applying a *slide detection threshold* of 20% in combination with an error correction, which accumulates slides that appear within a time span of the *slide drop threshold* of five seconds, results in a *slide detection rate* that (almost) matches the *real* slide switches. Hence, if recording slide presentations, the *automated slide detection* for *pixel-based recordings* reveal slide indices that are (almost) as good as those of *symbolic recordings*. Typically more than 95% of the slide switches are detected. Thus, we have eliminated one main drawback of the screen recording approach.

7.4 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 various "*motion images*". Such *dynamic content* consists of, for instance, animations, simulations and executed programming examples, which we will all call *animation* to simplify matters. Slide overlays or sequences of slides that appear within a certain delay may also be classified as *animation*. Unlike (static) slides, *dynamic content* does not generate *distinct peaks* but rather causes many updates 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 section (the start of the animation) and maybe the last one, which is less meaningful as an *access index*, but reveals the *duration* of the animation.

During the course "*Abstrakte Maschinen im Übersetzerbau*" [SS2004] Prof. Seidl used the VAM simulator (*visualization of Abstract Machines* (VAM) [Ziewer, 2001, VAM, 2006]), which visualizes the memory management (stack, heap and registers) during program execution. Screenshots are shown in Figure 7.11. Setting, copying and deleting values results in animated movements and fading effects for the objects, which represent the corresponding memory cells.

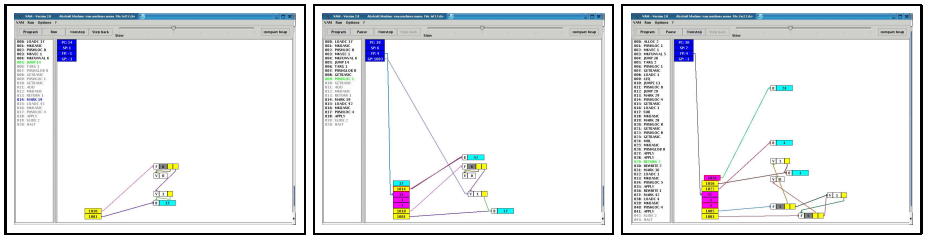


Fig. 7.11. Simulator Visualization[6]

Figure 7.12 (see also Figure 7.13 for a closer look) shows the resulting graphs of the update sizes *by-byte* and *by-area* for a recorded lecture during which the simulator was presented. The message sizes in bytes hardly reveal the usage of the simulator, but the graph of the affected areas evidently exposes the dynamic presentation content between minute three and seven. Even the short pause, which the teacher made to explain the presented simulation, is recognizable by the gap around minute five. The simulator usage results in framebuffer updates, whose sizes in bytes hardly exceeds the *by-byte slide detection threshold* of 12,500 bytes (Figure 7.13 to the left) due to the (almost) solid coloring of the moving objects, which results in good compression ratios. Showing a short movie instead would cause much larger sizes. Again, the relative values of the affected areas reveal more explicit results as shown on the right hand side of the figure.

Framebuffer updates, which follow each other at a high rate, are potentially caused by *dynamic presentation content*. Examining the gaps between the updates also

[6] Presented in lecture 2004/05/19 of "*Abstrakte Maschinen*" (Seidl, 2004)

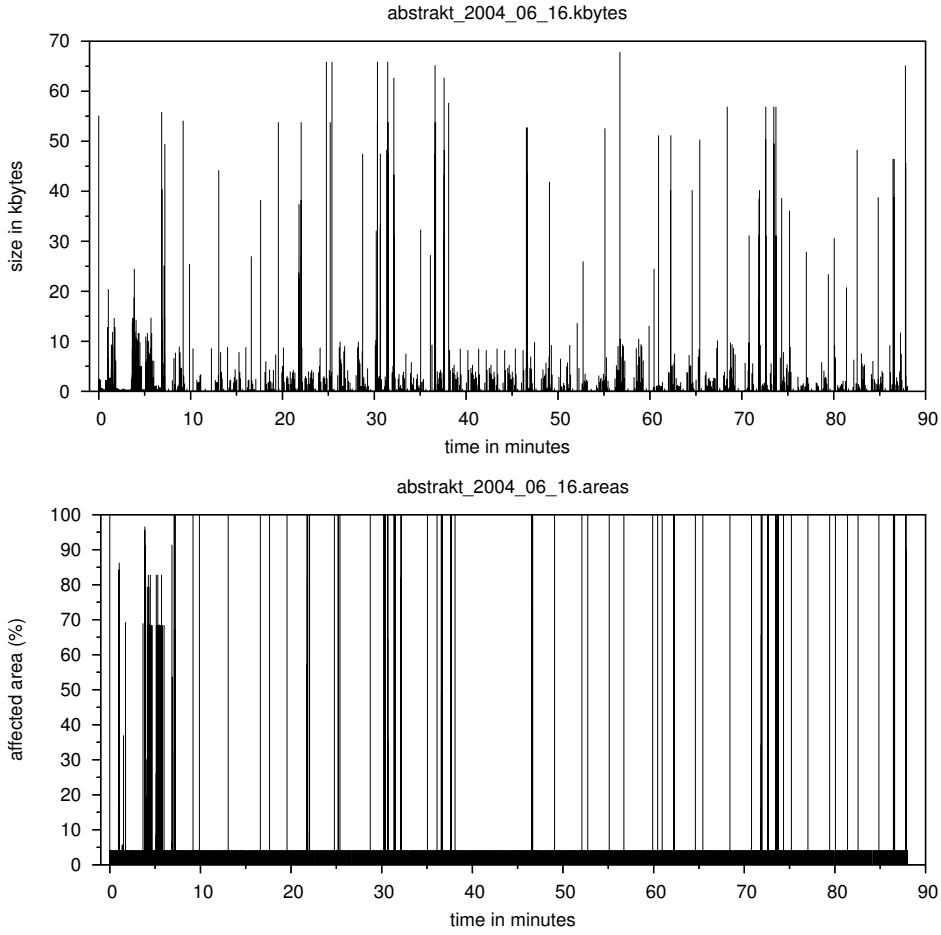


Fig. 7.12. Message sizes vs. affected areas

reveals the end of such a sequence of updates and thus enables multiple animations to be distinguished from each other. Hence, an animation starts whenever the delay between two consecutive framebuffer updates falls below a certain threshold and it lasts as long as the threshold is not exceeded (by the following delays between the following updates). Whenever the delay exceeds the threshold, the animation has completed. This ***animation detection threshold*** must either be lower than five seconds or the updates caused by the *non-incremental update stripes* must be filtered, because otherwise the delay between two consecutive updates will never exceed the mark of five seconds and thus will cause the algorithm to assume and classify one endless animation for each recording. For the preferable approach of regarding the affected areas instead of sizes in bytes, filtering is easily achieved by applying a ***stripe filter threshold*** of $4.1\overline{6}\%$. Only rectangles that update more than $4.1\overline{6}\%$ of the framebuffer are considered. A side effect of the *stripe filtering* is that movements of the mouse cursor are also filtered, which is not only acceptable but also rather appreciated, because moving the pointing device is commonly not the

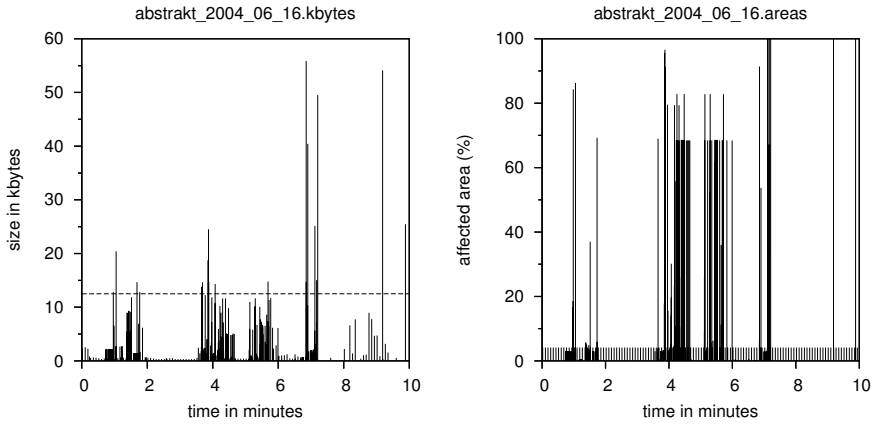


Fig. 7.13. By-byte and by-area results for recorded animations (zoomed)

kind of dynamic content that is of much interest. Note that the mouse cursor is not filtered if using the special *cursor encodings* but these encodings do not affect the framebuffer anyway and therefore filtering is not necessary. [Li et al., 2000a] enabled searching and accessing pointer events but stated that this feature is hardly used. Other sources of irritation are also filtered, for instance a (permanently changing) performance monitor or a clock, which are often placed in the task bar and hence may be visible during desktop recordings.

Surveying the results caused by recording the simulation, which are shown in a detailed view in Figure 7.14, exposes delays between consecutive 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 but a high value may assume two consecutive slides to be an animation instead. Hence, the delay 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. If the pause between the two parts should be overruled, the *animation detection threshold* must be increased to 30 seconds. But 30 seconds may already be a period in which a slide can be presented in a meaningful way. Applying an *animation detection threshold* of 15 seconds to the given lecture gathered some of the less interesting shell interactions, which were done to start applications. On the other hand, also some meaningful slide indices were combined to a single one. Therefore, the lower threshold of ten seconds is preferable. Reducing the threshold further to only 5 seconds, resulted in 67 instead of 59 indices. The eight additional indices were inserted during the period of the animation/simulation. Surveying the recorded lecture revealed that not that many indices are needed. Analyzing other lectures with dynamic content showed similar results.

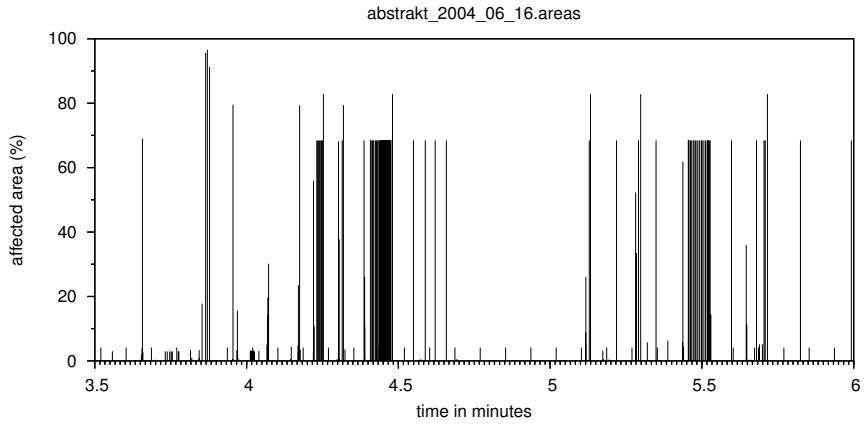


Fig. 7.14. By-area results for recorded animations (zoomed)

7.4.1 Slides vs. Animations

We have presented ways to detect slides and animations within recorded VNC sessions. Unfortunately, *animation detection* cannot be treated independently of the *slide detection*. This is not only the case for choosing a suitable *animation detection threshold* so that two consecutive slides are not classified as animation. Recall that we have suggested dropping potential slide indices which are succeeded by another one shortly after, in order to eliminate irritations caused by *skipped slides* (or delayed slide transmission). Applying the *animation detection* after the *slide detection* would cause the slide error correction to drop messages possibly useful for the animation detection and thus may cause the animation detection to fail. It is even not unproblematic if applying both algorithms directly to the recorded data and thus independently of each other. Consider a sequence which is assumed to be caused by *skipping slides*. The *animation detection* rather would classify such occurrences of subsequent updates as *animation*. However, the *slide detection* and the *animation detection* differ in their way of classifying the potential indices of a sequence to be important or not. The *slide detection algorithm* drops all indices except the *last* one, but, as animations should be visible in total, the *animation detection* rather keeps the *first* index and drops all following potential indices until a decreasing rate of updates reveals the end of the sequence. Classifying a sequence once as a skipped and thus unimportant slide, and once as meaningful animation is obviously inconsistent and therefore must be avoided. Therefore, animations must be distinguished from slides.

Approach I: Duration of sequence

Skipping slides (willfully or by mistake) as well as slightly delayed transmission of slide parts will rarely last for more than a couple of seconds. On the other hand, it is hard to imagine a meaningful animation of a short duration, for instance, of four or five seconds only. Hence, a sequence of framebuffer updates

with short delays is classified as an animation only if the overall duration exceeds a minimum time span, the ***sequence duration threshold***.

Approach II: Length of sequence

Another approach to differentiate between animations and skipped slides is to count the length of a sequence, i.e. the number of messages. As each displayed animation frame is represented by at least one framebuffer update, animations generally consist of many updates. On the other hand, delayed slide transmission or skipped slides commonly cause only a few updates and thus short sequences. Hence, animations and slides are distinguished by a ***sequence length threshold***. Note that all rectangles with identical timestamps must be considered to be one message because we have split a message that contains sequences of rectangles to a sequence of messages each containing a single rectangle only (discussed in Section 5.1.2).

Testing both approaches with different threshold values revealed only slight differences between the resulting indices. Again the example lecture (Figure 7.12 to 7.14) of the course "*Abstrakte Maschinen im Übersetzerbau*" is a good candidate for analysis, because during this lecture not only the simulator was used but the recording also contains several occurrences of skipped slides (e.g. around minutes 22, 30–32, 37-38), because the teacher switched back to recall previously shown slides and than switched forward to the current slide to continue his lecture.

We analyzed the index table generated with *sequence duration thresholds* of 5 an 10 seconds as well as with *sequence length thresholds* of 5 an 10 potential indices. Interestingly, the four different index computations have absolutely no influence on the generated index table for the periods during which the simulator was used. However, they differ in their classification of skipped slides. For both approaches, the longer sequences resulted in better indices where "better" means that the timestamp of the selected index refers to a framebuffer state which represents the slide to which the teacher's narration is related. This is because any potential slide indices which are caused by a *skipped slide* are correctly classified as *skipped slides*. However, the teacher had sometimes already started speaking while skipping slides and thus an index referring to the end of a sequence may cut off a little piece of the verbal narration.

On the other hand, applying shorter *sequence thresholds* causes the algorithm to classify the problematic sequences of about 10 seconds or 10 potential indices as *animations* and therefore they are replayed starting at their beginning, which results in skipping less verbal narration and thus provides a natural flow of the teachers speech. But creating a meaningful *visual index* (e.g. in the form of thumbnails), where the presented slide should give a hint on the addressed topic, the last potential index of these sequences is better suited. Otherwise the visual index may show a skipped and therefore irrelevant index. A reasonable solution out of this dilemma is to derive indices by use of the higher *sequence thresholds* in order to receive meaningful visual representations referring to the last slides (omitting skipped ones). Additionally, one has to ensure that the replay does not cut the corresponding narration, i.e. if accessing such a slide index, the corresponding timestamp less a short delay of 3–5 seconds

should be accessed instead of the exact timestamp. If the teacher said nothing while skipping slides, some silence is replayed, which naturally is less troublesome than starting in the middle of a sentence.

Note that there are only slight differences between the two classification approaches. If setting the higher thresholds (sequence of 10 indices or 10 seconds, respectively), the index table for that particular lecture (but similar results are acquired for other recordings) differs only in one value. Applying the lower thresholds (5 indices or 5 seconds), six of the 59 indices are classified differently, the first approach classifies a skipped slide whereas the other approach chooses an animation at four times. The other two indices are classified vice versa. So it is rather negligible which of the two approaches is applied because one might deliver a slightly better classification on one lecture but slightly less good results for another one. However, the density, i.e. the number of indices within the duration of a sequence, enables some kind of *content prediction*. If a sequence was caused by slide overlays commonly only one or two potential indices within 10 seconds are detected, but a real animation typically causes much more. The simulator usage, for instance, results in up to 15 potential indices per second.

7.4.2 User Events as Indicators

Although the *side-effect indices* that are caused by the RFB protocol messages are rather irrelevant (see Section 7.2.2), *user events* may be used as additional indicators in order to index a recorded session. Note that we rather *derive* indices from *user events* instead of using them directly as *side-effect indices*.

For instance, if a sequence of potential indices is accompanied by key presses commonly used to switch slides, it is rather unlikely that the sequence is caused by an animation and not by skipping slides. Due to security issues, the *TeleTeachingTool* does not log key events, but during the live presentation, the keys that are commonly used to switch slides, trigger automated removal of all current annotations in order to clear pages if switching to another slide. As those *RemoveAllAnnotations* events are logged, they provide similar hints to the key events themselves since *RemoveAllAnnotations* events will rarely, if ever, occur during animations. As the removal of all current annotations can also be induced by clicking on a dedicated *clear annotations* button without switching to another slide, a *RemoveAllAnnotations* event may furthermore suggest a break in the teacher's talk and thus a meaningful partitioning of a slide, which can be respected by adding an additional index (or sub-index).

Annotation events also give meaningful indices. Consider the freehand drawing feature. A short dash (e.g. underlining) may not be very interesting. But the first timestamp of a sequence of many freehand annotations potentially refers to a point within the lecture where the teacher had drawn and explained, for instance, a sketch, a formula or added some comments. Since *dynamic* freehand (or other) annotations are a special kind of animation, a classification of sequences is achieved analogous to the *animation detection* by analyzing the delays between subsequent events. Recall that the *TeleTeachingTool* handles annotations on a separate layer and stores them in the form of *symbolic representation* (see Section 6.1). Otherwise if storing anno-

tations *pixel-based*, for instance if using any of the drawing features of a recorded application, they cannot be accessed in an easy way as it is hard or even impossible to distinguish annotations from pointer movements, which we already declared to be rather meaningless for indexing purposes.

7.4.3 Conclusion

Generally analyzing *dynamic content* does not lead to as perfect indices as *slide detection* because the diversity of what can cause dynamics extend from designed animations, mouse movements and menu usage to any animation effects caused by arbitrary applications. However, calling this a problem of the *screen recording* approach would not be fair as the *symbolic recorders* typically are not able to record *dynamic content* at all (unless especially designed to be recorded with a specific recorder as is the case for a built-in *dynamic annotation feature*). Nevertheless, *dynamic content* can be distinguished from *static slide images*, and by filtering small updates caused by dynamic mouse movements or other irritations, meaningful *animation indices* can be achieved. Furthermore, we have stated how skipped slides, which are somewhat similar to a rather short animation, can be distinguished from longer running real animations (or other dynamic content).

7.5 Visual Representation of Indices

A suitable *presentation of indices* has a fundamental impact on the usability of the replay software. Besides the conventional *timeline navigation* (timeline slider and play/pause/skip buttons), the *TeleTeachingTool* offers a *graphical overview* of the automatically computed slide indices. Clicking on a small *preview image* (*thumbnail*) causes instantaneous playback of the corresponding slide. Additionally, accessing the previous and next slide is supported via buttons. A further *visual representation* of indices is a corresponding script that consists of one page per index. Each page shows a screenshot of the presented content and also may contain the corresponding annotations.

An index refers to a single *timestamp* but also represents the *part* (or *chunk*) of the recorded session from that particular timestamp up to the next index. Hence, a *preview image* should give a meaningful hint about the topic that is addressed during the *period* between two consecutive indices. Such an image is achieved by copying the state of the framebuffer at a certain timestamp. A scaled down copy is used as a thumbnail and 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).

Annotations give a good indication of the presented topics and the importance of the index. Consider the teacher adding additional comments to a slide, which revealed some comprehension problems. These explanations, and thus the index that refers to the corresponding slide, are of special interest for students during replay. Hence,

the visual representation of the index should already reveal the importance of the index by displaying the applied annotations. This is also the case for drawn sketches or tables. During recent years of recording lectures, we have recognized that the *freehand annotations* are of special interest for the visual representation of indices, but the *highlighting feature* is not that meaningful (for that purpose). Highlighting is of *momentary* importance as this feature is used for emphasizing and to focus the attention of the audience, but is not necessarily a meaningful hint on a thumbnail or a script page. Especially if many highlighting annotations are applied. Therefore we suggest that visual index representations should not contain highlighting annotations or at least highlighting annotations should be disengageable by students as is the case for the thumbnail overview of the *TeleTeachingTool*.

The *timestamp* of a slide index refers to the appearance of the slide and to the *beginning* of the teacher's explanations concerning that slide. Thus, annotations are not visible at the referred timestamp but will be added later. Consequently, we need some "*foresight*" to display relevant annotations within the visual index representation. If the teacher has removed some of the annotations and drawn others instead, displaying all annotations between two consecutive indices may look confusing. Hence, we gather all annotations that appear after the timestamp of the current index and the following index or the following *RemoveAllAnnotations* event, whichever occurs first. Note that an additional (sub)index for the remove event may be beneficial in order to present all annotations. This results in sets of annotations related to certain periods, which start at the timestamp of an index. Hence, annotations can be applied to the underlying framebuffer copy in the same way as to the real framebuffer during ordinary replay. As TTT annotations are handled on a separate layer and as annotations are distinguishable by their message types, the drawing of certain annotations types can easily be enabled or disabled within a thumbnail overview. Obviously, this is not possible if applying *pixel-based* annotations. In order to generate visual indices that include pixel-based annotations, we rather must conserve the annotations when they are available, which is at the end of a period between two consecutive indices. Hence, a thumbnail must copy the state of the framebuffer just before switching to the next slide. In order to eliminate irritations probably caused by slide switches, for instance, slow presentation software or if deleting annotations before switching, we suggest to store the framebuffer state of the following index minus a little delay of 1–5 seconds. Note that for a sequence of potential indices that has been classified to be caused by *skipped slides*, the displayed index is the *last* index of that sequence. Hence, the index minus one second probably will represent a skipped slide, which is not our intention. Therefore, rather the first potential index of such a sequence must be accounted as the end of the previous period. However, this can be achieved by storing different index tables for the automated thumbnail and script generation and to access slides during replay.

7.5.1 Automated script generation

The *visual representation of indices* is a useful basis to generate a *script* that corresponds to the recorded lecture. Certainly such a *pixel-based script* does not offer the editing and reusability features of the source documents (of the slide presentation)

or other symbolically represented documents (but the source documents can be used for such purposes). Such scripts are rather intended to be used as *additional learning material* for students, because not all teachers publish their slides (or other documents), which they have presented during the lecture. Even if slides were published once, they may no longer be available for older courses. However students may prefer to have a printed version in order to add comments or read them without the use of an electronic device, which is no problem if a script can be derived from an electronic lecture without access to the presented source materials.

An important feature of the generated script is the inclusion of *annotations* made during the live presentation. Any freehand comments or sketches are presented by such a script in contrast to the published (classic) slides, which commonly contain no annotations. Additionally, a script generated from a recording can present different document types. Consider a slide presentation that is paused to show and discuss some web pages or another document (type), all of which are then presented in a single script. Obviously, *dynamic content* cannot be presented fully in a static script since a screenshot will only represent a momentary state.

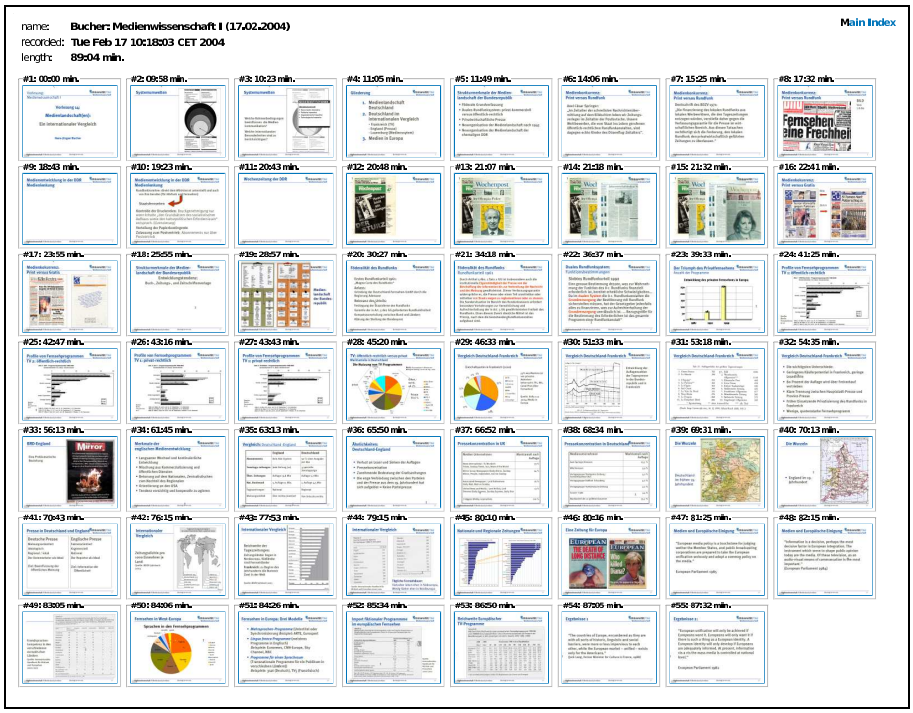


Fig. 7.15. Thumbnail overview of an automatically generated html script[7]

We have implemented the *automatic* creation of an *html script* by storing and linking slide images. For each index position, a copy of the framebuffer state is stored, once full scaled and once scaled down to thumbnail size. This is done analogous to

[7] Lecture 2004/02/17 of "*Medienwissenschaft I*" (Bucher, 2004)

thumbnail computation for the preview images shown with the *TeleTeachingTool* during replay. Certainly annotation layering as well as enabling and disabling annotation types is not as easy to achieve for (static) scripts as it is for the thumbnail overview within the TTT application (but might be provided by means of *dynamic scripting languages*). Our *script generator* rather applies the required annotations (all annotation types except highlighting) to the copy of the framebuffer before storing the (slide) image and the thumbnail. Hence, annotations are stored pixel-based as part of the presented images. If the drawing feature of the recorded presentation software is used instead of the TTT's built-in annotation system, annotations can also be made visible in the script by storing the state of the framebuffer just before the timestamp of the following index as suggested above.

The entry point of the script presents a *thumbnail overview* as shown in Figure 7.15. Each *thumbnail* is linked to a page with the corresponding full scaled (*slide*) *image* and each of the slide images has links to the previous and next index page as well as back to the overview, which are displayed at the top and the bottom of each page (Figure 7.16). The next page can also be accessed just by clicking on the full scaled image (to provide sequential progress). Furthermore, the overview provides some information about the recording like the *title of the course*, the *teacher's name* and the *date of recording*, which are extracted from the session name field and the start time of the recorded session.

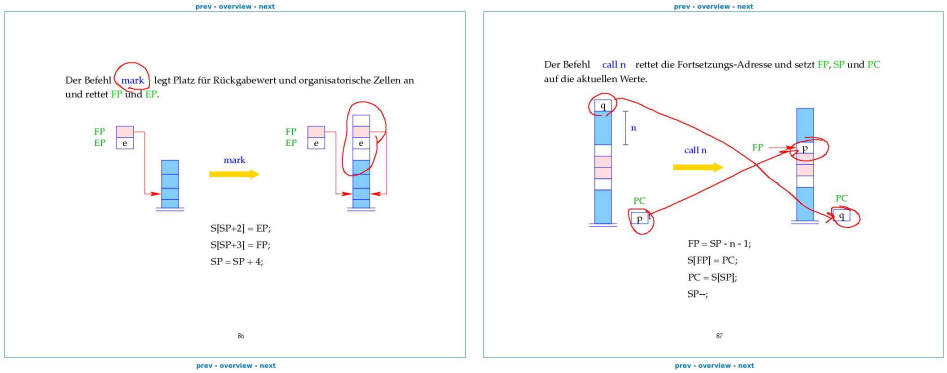


Fig. 7.16. Annotated pages of an html script with navigation links

Each automatically created script page has the same simple structure as the following example:

```
<html>
  <head>
    <title>Bucher: Medienwissenschaft I (17.02.2004) [42]</title>
    <link rel="stylesheet" type="text/css" href="style.css">
  </head>
  <body>
    <center>
      <b>
        <a href="medien1_2004_02_17.41.html">prev</a> -
        <a href="medien1_2004_02_17.html#42">overview</a> -
        <a href="medien1_2004_02_17.43.html">next</a><br>
        <a href="medien1_2004_02_17.43.html">
```

```
13            <img src="../images/medien1_2004_02_17.42.png" title="#42: 76:15 min.">
14          </a><br>
15          <a href="medien1_2004_02_17.41.html">prev</a> -
16          <a href="medien1_2004_02_17.html#42">overview</a> -
17          <a href="medien1_2004_02_17.43.html">next</a>
18        </b>
19      </center>
20    </body>
21  <html>
```

Any links and file names obey a *predefined naming scheme* that contains the *base file name* of the corresponding recording (which for our lectures corresponds to the possibly shortened course name and the date), followed by the *index number* and the *file ending*. The referenced *stylesheet* is also created by the *TeleTeachingTool* but stays the same for all scripts and pages. Therefore, any content provider (e.g. a university) may adapt or replace the given *stylesheet* according to their own requirements and likings. A *consistent naming* is also important to (automatically) interlink the results of an *online full text search* (described Section 8.3.1) or the *download pages* of our web archive with the corresponding *script pages*.

The generated structure of a sample overview page is given below. Lines 1–21 specify the title and display some metadata. The individual entries for each thumbnail start at line 23. The `<fieldset>` tag (since html 4.0) enables the grouping of elements. In combination with the *stylesheet* entry "`fieldset { display:inline }`" such grouping ensures that the number of thumbnails per line is adjusted according to the width of the browser window and thus gives a better look and feel. Most of todays browsers support this feature. Otherwise it may not be possible to group the thumbnails with the corresponding index numbers and timestamps, or only a fixed number of indices can be displayed per line. We also have implemented a script generation that does not rely on the `<fieldset>` tag and therefore can be used in order to support older browser versions.

```
1   <html>
2     <head>
3       <title>Bucher: Medienwissenschaft I (17.02.2004)</title>
4       <link rel="stylesheet" type="text/css" href="style.css">
5     </head>
6     <body>
7       <a href="../index.html" style="float:right;">Main Index</a>
8       <table cellpadding="5">
9         <tr>
10          <td>name: </td>
11          <td><b>Bucher: Medienwissenschaft I (17.02.2004)</b></td>
12        </tr>
13        <tr>
14          <td>recorded: </td>
15          <td><b>Tue Feb 17 10:18:03 CET 2004</b></td>
16        </tr>
17        <tr>
18          <td>length: </td>
19          <td><b>89:04 min.</b></td>
20        </tr>
21      </table>
22      <p>
23      <fieldset>
24        <legend> <a name="1"> <b>#1:</b> 00:00 min. </a> </legend>
25        <a href="medien1_2004_02_17.01.html">
26          <img src="../thumbs/medien1_2004_02_17.thumb.01.png" title="#1: 00:00 min.">
27        </a>
28      </fieldset>
```

```
29      <fieldset>
30        <legend> <a name="2"> <b>#2:</b> 09:58 min. </a> </legend>
31        <a href="medien1_2004_02_17.02.html">
32          <img src="../thumbs/medien1_2004_02_17.thumb.02.png" title="#2: 09:58 min.">
33        </a>
34      </fieldset>
35      <fieldset>
36        <legend> <a name="3"> <b>#3:</b> 10:23 min. </a> </legend>
37        <a href="medien1_2004_02_17.03.html">
38          <img src="../thumbs/medien1_2004_02_17.thumb.03.png" title="#3: 10:23 min.">
39        </a>
40      </fieldset>
41
42      .
43      .
44      .
45
46     <fieldset>
47      <legend> <a name="55"> <b>#55:</b> 87:32 min. </a> </legend>
48      <a href="medien1_2004_02_17.55.html">
49        <img src="../thumbs/medien1_2004_02_17.thumb.55.png" title="#55: 87:32 min.">
50      </a>
51      </fieldset>
52    </body>
53  <html>
```

We have implemented the automated html script generation in order to provide visual representations of the recorded lectures in our *web archive*. In order to produce a more printer friendly document, the script should rather by generated in the form of a *pdf* or similar document type, which can be done by generating pages including a single screenshot for each index.

7.6 On The Fly Analysis

Criterion C4c: Annotations associated with slides (or other elements) in a way that guarantees that annotations disappear when a slide is changed and are made visible again when returning to that slide later during presentation, requireindices to be available *during* a live lecture. However, our detection and classification algorithms are designed to be performed *after* the recording process is finished and all messages are available in advance. Hence, we must adapt the automated analysis to be performed *on the fly* while the presentation is still in progress. A suitable index computation algorithm must be able to determine useful indices by processing a *stream of sequential messages* and decide not only where to set indices, but where to set meaningful indices, and this in an *realtime fashion*.

Essentially, the automated post-processing consists of 3 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 a priori, 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 5–10 seconds or a *sequence length threshold* of 5–10. 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*). This duration is either the *sequence duration threshold* and thus 10 seconds, or a sequence of 10 potential indices (the *sequence length threshold*), which have a maximum delay of 10 seconds (*animation detection threshold*), and thus 90 seconds (although rarely reached in practice). For *on the fly* classification, the approach of examining the duration of a sequence is evidently better suited.

Nevertheless, we discuss *fast termination* of the other approach as well. Commonly, the delays between two consecutive indices are much shorter (in the case of animations or skipped slides) or much longer (for static slides) than the threshold of ten seconds. Static slides, which generate a single *potential index* only, can be classified after 10 seconds (as is the case for the other classification approach), because the *animation detection threshold* is exceeded and thus the "sequence" consists of a single index only. Real animations cause many *potential indices* with very short delays at high probability and therefore will exceed the *sequence length threshold* very soon. For the recorded simulator usage, which we have discussed in Section 7.4, a density of 5–15 *potential indices* per second is not uncommon. In this case, the classification can be achieved after one or two seconds (and thus even faster then the other approach which always must wait until the *animation duration threshold* of 10 seconds has passed). Skipped slides are commonly classified after no more than 15–20 seconds, because skipped slides may cause some potential indices (commonly 2–4 with very short delays) after which a pause follows that exceeds the *animation detection threshold* of 10 seconds. Therefore, a classification by examining the length of a sequence by the number of the included *potential indices* is practically achieved in less than 30 seconds, and for animations typically in less than 5 seconds. Note that this is unproblematic for the post-processing, because the offline computation is not performed in a timeline fashion according to the logged timestamps and can rely on all messages being available in advance.

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* or the *sequence length 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 considered to be valuable. A sequence is classified to be (part of) 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 as 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 somewhere in between.

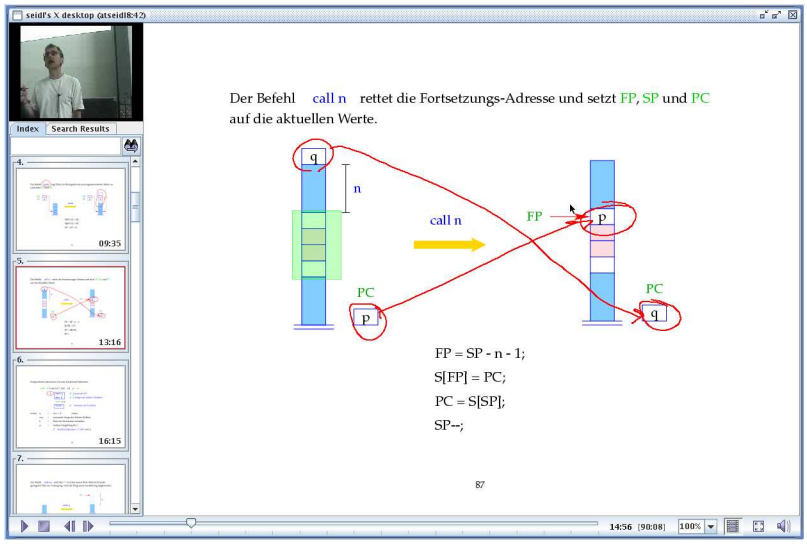


Fig. 7.17. TTT player with thumbnail overview

A *thumbnail overview* (Figure 7.17 to the left) 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 *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 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 little 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 a result each slide can have several sets of annotations, which refer to different remarks by 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 state (i.e. the contained pixels) 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 receiving a potential index, but to wait until the next framebuffer message arrives. This offers the possibility of observing the next header, 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, as an update can contain several rectangles but 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. Note that our optimized file format that provides fast access to rectangle headers is not valid here, because the input stream is the message stream received from a VNC server and thus obeys the original RFB protocol.

7.6.1 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 when new annotation are created. 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 keeping the entire 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 an *annotated screenshot*, or more precisely a screenshot of the (previously) presented content plus the corresponding annotations on a separate layer, to be shown whenever an index is classified as slide, and the *dynamic replay* of animations or other content otherwise.

7.7 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. While replaying a previously recorded session, *annotations* are displayed according to their *timestamps* in the same *timeline fashion* as they were recorded. For sequential playback the message timestamps are sufficient. If accessing a certain slide, for instance via *thumbnail overview*, the corresponding state of the framebuffer is computed and since then any subsequent messages are again replayed sequentially including all annotations (of the currently shown slide).

Recall that [Lauer and Ottmann, 2002] postulate that *during live presentations* annotations should be associated with slides so that annotations disappear when a slide is changed and are made visible again when returning to that slide later (which is our Criterion C4c). The first aspect of *removing the current annotations* whenever switching to another slide is supported by the *TeleTeachingTool* by applying *automated removal of annotations* triggered by the keys commonly used to switch slides (the arrow keys and the page up/page 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 consecutive indices (or a timestamp and the next event that removes annotations) as is done to gain the annotations for the *visual representation* of indices (thumbnails or script pages). *On the fly interlinkage* of annotations with already computed indices is achieved by *buffering annotation events*. Hence, if accessing a slide via *thumbnail overview* (or any other position within the timeline) all previously made annotations that are still valid for the given timestamp, are displayed immediately.

However, this is only a *loose interlinkage*, because indices are only referencing (indices of) slide changes without any knowledge of *slide content*. A slide shown twice during the presentation causes two independent slide indices with different sets of

corresponding annotations. A *real association* between annotations and slides (or any other content) would even make it possible to recall corresponding annotations whenever skipping back to previously annotated slides within the presentation software.

7.7.1 Content Interlinkage

In order to achieve *content 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* such as *cyclic redundancy check* (CRC32), is a relatively easy task. Different checksums relate to different framebuffers and, if chosen suitably, matching checksums should point to equal content at high probability.

Unfortunately, such a comparison will be problematic unless the 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*" [Winter 2005/06] of Prof. Dr. Johann Schlichter (25 recordings of approx. 90 min.), revealed a ***content matching threshold*** of 1.1% differing pixels as suitable to identify slides. Applying the same threshold to the recordings of the courses "*Compilerbau*" and "*Abstrakte Maschinen*" [Summer 2006] by Prof. Dr. Helmut 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 a designated 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 link(s) 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 but adjustable for special occurrences is practicable. At least 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 can also reduce irritations.

Pixel-based comparison of several framebuffer states (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 demonstrated that for most slides approximately 90–95% of the pixel values are set to the same color (assuming a single colored background). The results of a typical lecture by Prof. Seidl are given in Figure 7.18, which displays how much of the framebuffer is covered by the most frequently used color for each index.

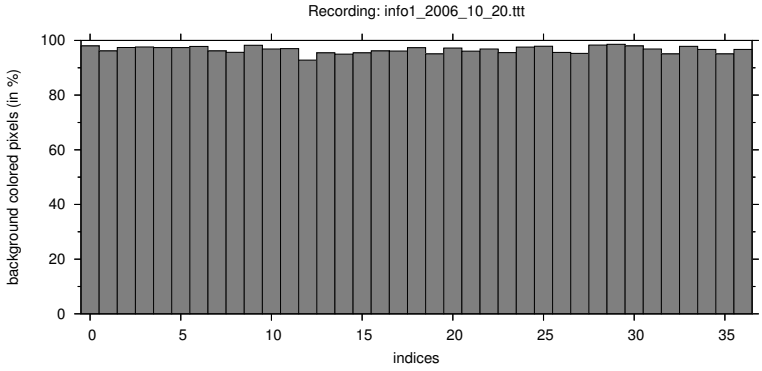


Fig. 7.18. Background pixels of presentations showing mainly text

Surveying lectures with slides that present content of a higher *pixel variety* reveal that even very complex slides rarely contain less than 40% of background colored pixels. The graph in Figure 7.19 corresponds to the lecture of Prof. Dr. Hans-Jürgen Bucher that is presented in the slide overview shown in Figure 7.15. Slides that present scanned newspaper articles or other complex pictures are evidently exposed by lower values.

The coloring of many pixels in the same color leads to very high compression rates even for simple and therefore fast compression schemes such as *run-length-encoding*. As the majority of comparison partners represent *unequal* slides, the comparison algorithm should detect and reject them as fast as possible, at best by a *single value comparison*. Examination of several dozen recordings revealed the *number of background pixels* to be a suitable criterion. Slides cannot match each other if the number of pixels that are colored in the most frequently used color, which typically is the background color, exceeds the previously mentioned *content matching threshold* of 0.2% or 1.1%, because differing background pixels are a subset of all differing pixels.

In order to determine the background color we compute a *color histogram*, i.e. counting how many pixels are colored by each color. Hence, each pixel must be

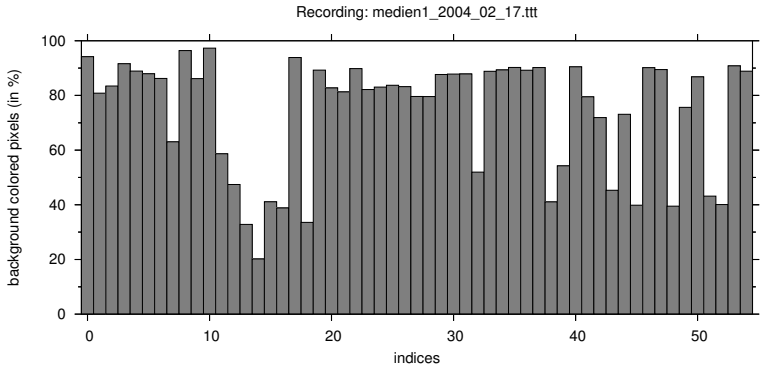


Fig. 7.19. Background pixels of presentations showing very complex slides

accessed once, but this is also the case for any other comparison algorithm. At least this can be combined with constructing additional *data structures* to perform an efficient pixel comparison, whenever required later (for instance *quad trees* [Hunter and Steiglitz, 1979]). The *complete* comparison of all pixel values is carried out only if the number of background pixels almost matches. Hence, the main case of comparing non-matching framebuffer contents can be achieved in time $\mathcal{O}(w \cdot h + n)$ in order to compute the histogram and perform the single value comparisons instead of performing all pixel comparisons which would result in $\mathcal{O}(n \cdot w \cdot h)$, where w and h denote the *framebuffer width* and *height*, respectively, and n denotes to the *number of indices*.

Note that the suggested *quick rejection* will probably fail if *color cycling* or *high colored background images* are used instead of a (mainly) solid background, but their usage for VNC environments is discouraged anyway, because they result in rather bad compression ratios and thus high bandwidth usage (Section 3.2.6). Another approach could be a *similarity hash*, but a suitable hash function needs to be ascertained first.

7.8 Content Prediction by Color Histograms

Through the examination of the background color of recorded lectures we have detected that the color histogram exposes information about the framebuffer content. For simple slides over 90% of the pixels are in the background color, but more complex slides achieve values of approximately 55-85%. A desktop with a taskbar, 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 the framebuffer represents a fullscreen video or high colored picture. 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 (e.g. a photo).

Figure 7.20 displays the analysis of one example recording ("*Informatik II*", 04/15/2005 by Prof. Seidl). It shows how much of the framebuffer is covered by

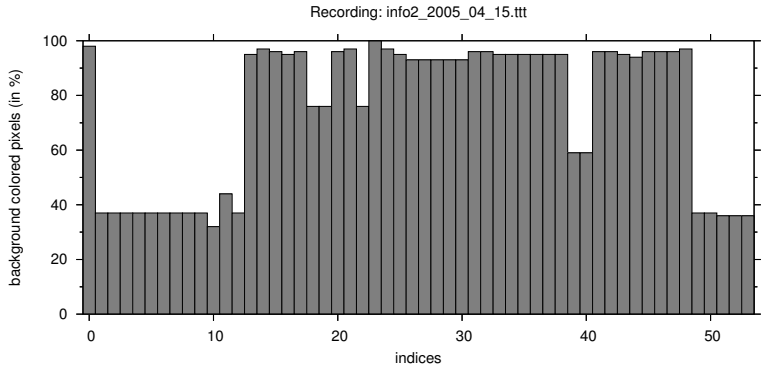


Fig. 7.20. Background pixels of presentations showing slides and simulator usage

the most frequently used color for each index. During the corresponding lecture the desktop with some windows was visible at the beginning (up to index no. 13) and the end (no. 50-54). The middle part consisted of a slide presentation and some slides contained images (no. 19, 20, 23, 40 and 41). Index no. 24 was a plain whiteboard page and thus resulted in 100% background pixels.

Further research is required in order to the possibilities of *content prediction* regarding *color histograms*. In order to classify slide images (or screenshots in general), we must determine classes of slides (and other content) first and furthermore appropriate thresholds for each class must be ascertained. Additionaly, an appropriate user interface must be designed that is capable of presenting content classes and enable retrieval of certain content. Our lecture archive contains about 400 lectures with approximately 10,000–20,000 slide images in total and thus is a suitable basis for further research.

7.9 Summary

Sequential playback and *timeline navigation* is not sufficient to provide electronic lectures of high quality. *Structured* electronic lectures rather provide *navigation marks* in order to access certain points of interest. *Screen recording* offers a *flexible technique* for lecture recording as it allows virtually any material displayed during a presentation to be captured but is criticized for producing *unstructured* electronic lectures only. Adding some kind of structure by *manual indexing* during a live lecture or afterwards is not in accordance with our *transparent* and *lightweight* content production approach.

In this chapter we have presented different approaches regarding how to acquire *navigational indices* for pixel-based recordings *fully automated* by *deriving indices* from the messages of a recorded VNC session. The *by-byte* analysis gains meaningful indices for *homogenous* presentation content but fails for *inhomogeneous* recordings. With the *by-area* approach and the given *empirically determined thresholds*, slide indices for pixel-based recordings can be generated for any of our recordings. The

usage of *relative* values instead of *absolute* ones as s basis for the detection algorithms leads to much better results, because relative values provide some kind of *abstraction* from the presented content. Applying a *slide detection threshold* of 20% in combination with an *error correction* (i.e. accumulating messages of almost identical timestamps and dropping skipped slides) results in a *slide detection rate*, which (almost) matches the real slide switches. Hence, if recording *slide presentations*, the automated slide detection for *pixel-based recordings* produces *slide indices* that are (almost) as good as those of *symbolic recordings*. Thus, we have eliminated one main drawback of the screen recording approach.

A *thumbnail overview* gives a *visual representation of indices* that, in combination with *fast random access*, offers an easy to use *slide based navigation*. Storing *full scaled visual indices* enables the extraction of a script directly from pixel-based recordings without any access to the presented source documents. As an example, we have addressed the automatic generation of an html script by interlinking the acquired screenshots but other document forms can be produced rather similarly. Furthermore, we have discussed *content prediction* in the form of *animation detection*, accessing notes and sketches created via *freehand drawing*, and by analyzing *color histograms*. An examination of the background color of recorded lectures has revealed that color histograms give information about content, but more research is needed to achieve suitable thresholds for content prediction and to integrate appropriate search and navigational features in a reasonable way. Furthermore, analysis of dynamical content should be improved.

Another advantage of the *symbolic representation* is the *association of annotations with slides*. However, *TTT annotations*, which are also stored symbolically, can be *interlinked with slide indices* according to their timestamps. Therefore, the indexing algorithm, which is originally designed for post-production only, is transferred to *on the fly* usage to achieve live access to previously annotated slides. By *comparing framebuffer contents*, the annotations can even be linked to *pixel-based content* (slides images). In order to do so, we have suggested possibilities to perform efficient *screenshot comparison*.

In summary, we have extended the flexible approach of preserving live lectures via *screen recording* in a way that it also produces *structured electronic lectures* as commonly is only possible by use of symbolic representation. Now screen recording is no longer just a *teacher friendly* recording solution but also provides easy to use and *student friendly* navigational features of high quality.

8

Retrieval and Metadata

Students rarely want to replay complete sessions but rather want to select those parts that address certain topics [Zupancic and Horz, 2002, Schütz, 2003]. Besides *Random access* and *indexing structures*, full-fledged *electronic lectures* must also support appropriate *retrieval features* in order to *locate* content within *electronic lectures* and *databases of lectures*.

For the recording of *VNC sessions*, [Li et al., 2000a] suggested to *log key events* in order to enable *retrieval of the entered text*. Their intention was not to record dedicated presentations (as we do) but rather to conserve user interactions with the command shell or applications such as editors. Hence, the reviewing user may see previously entered commands and can request the timestamp, when they where entered. It is doubtful how meaningful it is to access such timestamps because the result is already visible on screen and no additional information is given as their recording environment only preserves the VNC session but no verbal narration. However, it might be useful to search for a certain command (or other text) which is not yet visible but known to be entered during the session. Such retrieval will fail if only exact matches are presented. Consider a user entering some text and deleting and correcting a mistyped key. This problem can be solved by *normalizing* the sequence of *key events*, i.e. deriving the final character sequence. Furthermore, logging of *key events* of desktop sessions is safety critical as any entered password will by logged as well. Hence, we do not see *key event retrieval* as an adequate retrieval feature for electronic lectures especially as typically only very little text is entered during a presentation. Meaningful *text retrieval* should rather address the *presented (slide) content*, either by use of *metadata* and *textual annotations* that describe certain (parts of) electronic lectures or in the form of *full text search*.

In this chapter, we will explain how *full text search* can be in performed for *pixel-based recordings* by automatically extracting a *search base* from *electronic lectures*. Furthermore, we discuss a simple *metadata model* and how *metadata* is meaningful in order to *categorize lectures* and provide *cross lecture retrievability* for *databases of electronic lectures*.

8.1 Full Text Search for Pixel-based Recordings

A major advantage of *text-based electronic media* is an easy to use *full text search*. By specifying a *search pattern*, typically a string, a search engine offers a list of all identified matches or subsequent searches offer the respectively next occurrence. Electronic lectures that store *symbolically represented data* (including text) obviously can offer *full text search* very easily. For screen recording systems, which store *pixel data* only, the presented text has to be acquired somehow. One way would be to make use of the (textual) sources of the presentation, which commonly are symbolically represented. The so obtained textual content must be *logically interlinked* with the pixel-based recording. If a teacher presents her/his slides strictly in order, an automated process may extract and link textual content to slide indices of the recording. However, if the teacher skips back to previously shown slides, accesses any slide directly (e.g. by entering the corresponding slide number) or if the indices do not exactly correspond to the presented slides, for instance if a slide was not detected by the indexing algorithm, such an automated process probably must fail. Furthermore, as screen recording allows arbitrary applications to be used during presentations, there may be various different source documents and formats. This restricts the practicability of providing *text retrieval* by use of the presented *source documents* as the process can hardly be automated and manual processing is laborious and thus discouraged for *lightweight lecture recording*.

8.1.1 Text Extraction

Instead of accessing the *source documents* that were shown during the live presentation, we rather make use of the recorded pixel-based *slide images*. *Slide images* are independent of the formats of the source documents. Hence, any presented content can be handled in the same way, which is preferable for our flexible recording approach. *Slide images* are acquired by automated indexing and storing *screenshots* of the corresponding *framebuffer states* as described in Chapter 7. In fact, the pages of an automatically generated html script (Section 7.5.1) already provide such *index screenshots*.

Optical character recognition (OCR) (also known as *digital character recognition*) algorithms are designed to translate images of handwritten or typewritten text into machine-editable text. Via sophisticated pattern matching, pictures of characters are translated into a standard encoding scheme representing them (e.g. ASCII or Unicode). Traditional *character recognition* algorithms are designed for *printed sources* (usually captured by a scanner). These algorithms are optimized for scanned *high resolution images* of 300–400 dpi (dots per inch). As scanning is typically an error-prone process, *error correction techniques* do a re-alignment to ensure a horizontal character layout and try to compensate for shades, spots or other visual irritations.

In comparison to high resolution scans, *screenshots* of typically 1024×768 pixels offer a rather *low resolution* (comparable to 72–100 dpi in printed sources), which may impair text recognition results. On the other hand, screenshots are *digital sources* with exact horizontal alignment and without the typical irritations caused by scanned

analog documents. In a brute force test we have applied commercial OCR software applications (Scansoft's[1] *OmniPage Pro 11* [OmniPage, 2006] and ABBYY's *FineReader OCR 7.0* [Finereader, 2006]) to the pages of our automatically created script. We achieved mostly good results for textual content but results for formulas and sketches were rather bad. Badly recognized content is not very useful in the original intention of OCR software, which is to provide editable, printable and re-usable *digital documents* (as copies of analog sources). However, our intention is not to edit or present the extracted data but to acquire a *search base* for retrieval purpose. Students will commonly specify *keywords* as search patterns and not formulas. Hence, our *search base* should contain any meaningful text that is likely to be searched for. Any other words or characters that are stored in the *search base* are unproblematic, even if special characters and "strange signs" are included. For example, take a look to the following ASCII text that was generated by applying character recognition to the slide displayed in Figure 8.1:

```
Exkurs 2: Vollständige Verbände Eine Menge DD mit einer Relation
E [D ID ist eine partielle Ordnung falls für allen, b, c E ID gilt:
n E n Refexivität a E b b E a a=b Anti ? Symmetrie a E b 1, a E c
Transitivität Beispiele: 1. D = 2{ °'‘l mit der Relation : Ä.b - aG
```

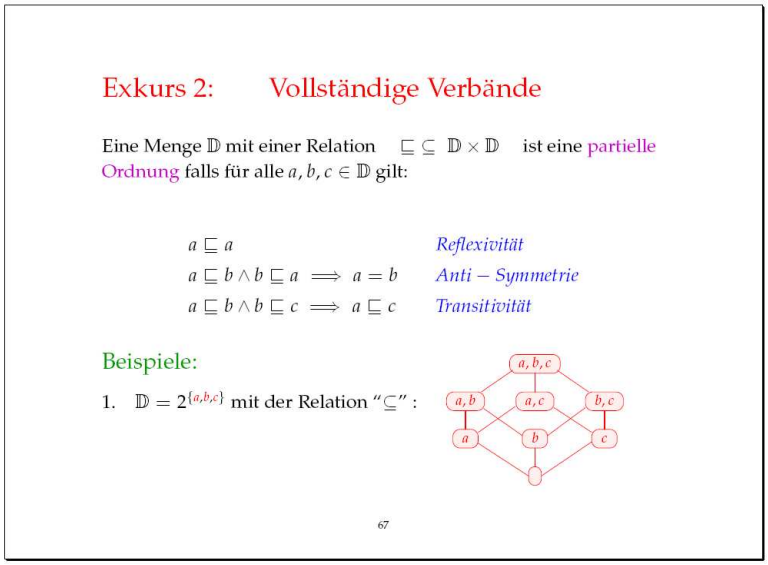


Fig. 8.1. Slide that corresponds to OCR output

Most text, including *keywords* that possibly will be searched, for instance "Verbände", "Reflexivität" or "Symmetrie", are recognized correctly. However, the useless character sequences that are deduced from the formulas or the figure at the

[1] *Scansoft* recently changed its name to *Nuance* but the OCR software is still sold as *Scansoft's OmniPage*

bottom of the slide are only presented here for demonstration purpose but will normally be hidden in the *search base* and thus will never be visible to students and thus cannot confuse students. In fact, for retrieval purposes it is sufficient to obtain the information whether a sequence is included in a certain part of the search base or not. Only the statement "*is included*" leads to *search results*. Unrecognized content that is never queried is irrelevant but it is unproblematic if such content is stored in the search base. Words that are potentially searched for but are not recognized correctly will not be found and thus cause fewer search results. However, students might be used to getting not all possible search results, because this is also the case for many other retrieval systems (consider web searches) but at least the presented results are relevant.

Extracting *ASCII text* for each slide enables *slide-based search results*. Hence, search results can refer to the pages that match the search pattern but typically also an emphasizing of the search result within the page is desirable, which requires the corresponding coordinates. Sophisticated OCR algorithms can not only extract the characters but also the layout of the input documents. In order to store the layout, OCR applications generally support output formats that are designed for *word processing* purposes, e.g. *Rich Text Format* (RFT), various *Microsoft Office* formats or the *Portable Document Format* (PDF). Unfortunately, these formats typically do not contain *absolute coordinates* but rather blocks of text with assigned font types, styles and sizes. The layout of words is *relative* within each block and the blocks are positioned *relative* to other blocks. Hence, calculating *absolute pixel coordinates* demands the complex parsing and processing of a word processor, which probably is overkill for our purpose.

Some (mostly more expensive) OCR applications also support *Extensible Markup Language* (XML) as an output format (not to be mistaken with Microsoft's *WordML*, which also uses the file extension ".xml"). Currently, we use Scansoft's *OmniPage Pro 14 Office* [OmniPage, 2006], which generates XML output files that include the coordinates and the dimensions of each recognized word (or character, depending on the set properties). Note that for our sources the recognition results of ABBYY's *FineReader OCR 7.0* [Finereader, 2006] are better but *FineReader OCR 7.0* does not support XML output. Hence, we do not suggest *OmniPage* to be the best candidate in order to extract a *search base* from slide presentations but will rather discuss *OmniPage*'s XML output format as an example. Other OCR applications with appropriate output formats may be used as well.

Now take a look at the XML file that corresponds to the slide shown in Figure 8.1:

```
<?xml version="1.0"?>
<!--XML document generated using OCR technology from ScanSoft, Inc.-->
<document  ssdoc-vers="SSDOC1.0" ocr-vers="OmniPage Pro 14"
           xmlns="x-schema:http://www.scansoft.com/omnipage/xml/ssdoc-schema2.xml">
  <page width="4896" height="3523" x-res="300" y-res="300" bpp="1"
        orientation="0" skew="0" filename="Optimierung_2006_10_17.44.png" language="1">
    <region reg-type="horizontal">
      <rc l="643" t="425" r="4181" b="2885"/>
      <paragraph para-type="text" align="left" left-indent="0" right-indent="0"
                 start-indent="0" line-spacing="336">
        <ln baseline="600" ff="Garamond" fs="900">
          <wd l="696" t="485" r="1210" b="605">Exkurs</wd>
          <wd l="1262" t="490" r="1373" b="605">2:</wd>
          <wd l="1742" t="485" r="2664" b="653">Vollständige</wd>
```

```
    <wd l="2731" t="485" r="3427" b="605">Verbände</wd>
    </ln>
</paragraph>
<paragraph para-type="text" align="left" left-indent="0" right-indent="0"
           start-indent="0" line-spacing="156">
  <ln baseline="946" ff="Bookman Old Style" fs="500">
    <wd l="691" t="869" r="912" b="950">Eine</wd>
    <wd l="960" t="869" r="1306" b="984">Menge</wd>
    <wd l="1344" t="864" r="1426" b="950">DD</wd>
    <wd l="1464" t="869" r="1637" b="950">mit</wd>
    <wd l="1670" t="869" r="1930" b="950">einer</wd>
    <wd l="1963" t="869" r="2395" b="950">Relation</wd>
    <wd l="2558" t="874" r="2635" b="946">E</wd>
    <wd l="2846" t="864" r="2928" b="950">][D</wd>
    <wd l="3082" t="864" r="3163" b="950">ID</wd>
    <wd l="3312" t="869" r="3442" b="950">ist</wd>
    <wd l="3470" t="869" r="3682" b="950">eine</wd>
    <wd l="2147483647" t="2147483647" r="0" b="0">partielle</wd>
  </ln>
  <ln baseline="1104" ff="Bookman Old Style" fs="500">
    <wd l="691" t="1027" r="1166" b="1138">Ordnung</wd>
    <wd l="1210" t="1022" r="1416" b="1109">falls</wd>
    <wd l="1459" t="1022" r="1603" b="1109">für</wd>
    <wd l="1642" t="1027" r="1934" b="1123">allen,</wd>
    <wd l="1968" t="1022" r="2045" b="1123">b,</wd>
    <wd l="2078" t="1051" r="2122" b="1109">c</wd>
    <wd l="2170" t="1042" r="2232" b="1114">E</wd>
    <wd l="2280" t="1022" r="2362" b="1109">ID</wd>
    <wd l="2400" t="1027" r="2587" b="1142">gilt:</wd>
  </ln>
</paragraph>
<paragraph para-type="text" align="left" left-indent="504" right-indent="0"
           start-indent="0" line-spacing="192">
  <ln baseline="1560" ff="Times New Roman" fs="600" char-attr="italic">
    <wd l="1229" t="1507" r="1277" b="1565">n</wd>
    <wd l="1330" t="1488" r="1406" b="1560" ff="Bookman Old Style" fs="500"
        char-attr="non-italic">E</wd>
    <wd l="1454" t="1507" r="1502" b="1565">n</wd>
    <wd l="2147483647" t="2147483647" r="0" b="0">Refexivität</wd>
  </ln>
</paragraph>
<paragraph para-type="text" align="left" left-indent="504" right-indent="0"
           start-indent="0" line-spacing="192">
  <ln baseline="1752" ff="Times New Roman" fs="600" char-attr="italic">
    <wd l="1229" t="1699" r="1277" b="1757">a</wd>
    <wd l="1330" t="1680" r="1406" b="1752" ff="Bookman Old Style" fs="500"
        char-attr="non-italic">E</wd>
    <wd l="1454" t="1670" r="1502" b="1757">b</wd>
    <wd l="1642" t="1670" r="1690" b="1757">b</wd>
    <wd l="1742" t="1680" r="1819" b="1752" ff="Bookman Old Style" fs="500"
        char-attr="non-italic">E</wd>
    <wd l="1862" t="1699" r="1910" b="1757">a</wd>
    <wd l="2256" t="1670" r="2530" b="1757">a=b</wd>
    <wd l="2770" t="1670" r="2990" b="1757">Anti</wd>
    <wd l="3029" t="1723" r="3110" b="1728"> </wd>
    <wd l="3149" t="1675" r="3662" b="1790">Symmetrie</wd>
  </ln>
</paragraph>
<paragraph para-type="text" align="left" left-indent="504" right-indent="0"
           start-indent="0" line-spacing="192">
  <ln baseline="1944" ff="Bookman Old Style" fs="600" char-attr="italic">
    <wd l="1229" t="1891" r="1277" b="1949" ff="Times New Roman">a</wd>
    <wd l="1330" t="1872" r="1406" b="1944" fs="500" char-attr="non-italic">E</wd>
    <wd l="1454" t="1862" r="1502" b="1949" ff="Times New Roman">b</wd>
    <wd l="1642" t="1862" r="1690" b="1949" fs="500" char-attr="non-italic">1,</wd>
    <wd l="2251" t="1891" r="2299" b="1949" ff="Times New Roman">a</wd>
    <wd l="2352" t="1872" r="2429" b="1944" fs="500" char-attr="non-italic">E</wd>
    <wd l="2477" t="1891" r="2520" b="1949" ff="Times New Roman">c</wd>
    <wd l="2147483647" t="2147483647" r="0" b="0" ff="Times New Roman">Transitivität</wd>
  </ln>
</paragraph>
<paragraph para-type="text" align="left" left-indent="2592" right-indent="288"
```

```
            start-indent="-2592" line-spacing="276">
        <ln baseline="2299" ff="Bookman Old Style" fs="500">
          <wd l="691" t="2203" r="1272" b="2338">Beispiele:</wd>
          <wd l="691" t="2467" r="763" b="2549">1.</wd>
          <wd l="898" t="2462" r="979" b="2549">D</wd>
          <wd l="1022" t="2506" r="1114" b="2534">=</wd>
          <wd l="1157" t="2443" r="1253" b="2549">2{</wd>
          <wd l="1325" t="2443" r="1445" b="2525">°'‘l</wd>
          <wd l="1488" t="2467" r="1670" b="2549">mit</wd>
          <wd l="1704" t="2467" r="1877" b="2549">der</wd>
          <wd l="1915" t="2467" r="2347" b="2549">Relation</wd>
          <wd l="2635" t="2496" r="2645" b="2549">:</wd>
          <wd l="2832" t="2443" r="3014" b="2539" char-attr="italic">Ä.b</wd>
          <wd l="3806" t="2515" r="3821" b="2539" char-attr="italic">-</wd>
        </ln>
        <ln baseline="2760" ff="Arial Narrow" fs="500" char-attr="italic">
          <wd l="3307" t="2482" r="3408" b="2779">aG</wd>
        </ln>
      </paragraph>
    </region>
  </page>
</document>
```

The `<wd>` tags reveal the coordinates of recognized words (character sequences) by specifying the "`l`" (left), "`t`" (top), "`r`" (right) and "`b`" (bottom) parameter of the following character sequence. Note that the *parameter values* correspond to the *resolution* that is specified in the `<page>` tag and therefore must be translated into the *pixel resolution* of the screenshots. Furthermore, the `<page>` tags give the *mapping* of the recognized text to the *slide indices*. The other tags are not required for our intention of emphasizing the search results within slide images.

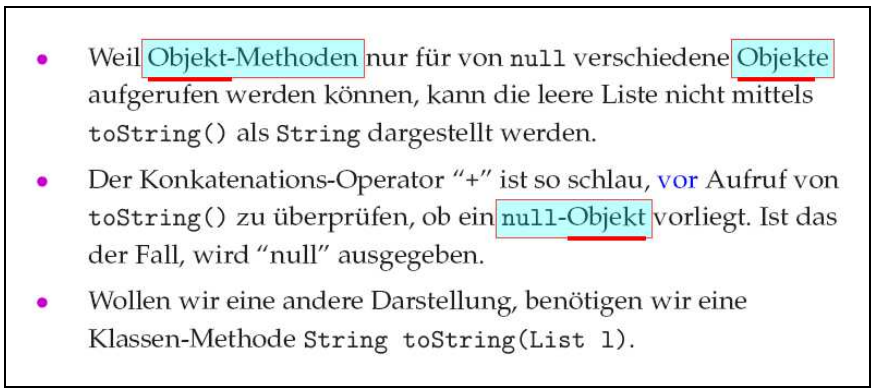


Fig. 8.2. Emphasizing search results and the search pattern "objekt"

In the TTT environment, emphasizing is done by placing *highlight annotations* automatically, for instance like those displayed in Figure 8.2. As users should not be forced to enter entire words, we perform *substring searches*. Therefore it is advisable not only to highlight complete words that contain the *search pattern* but also to mark the search pattern *within* that word, for instance by underlining it. Without marking the search pattern, it might not be evident at a glance why the system presented a particular search result. If the search pattern matches the beginning of a recognized word and/or is a semantical subword as "objekt" in Figure 8.2, it is obvious to the user why this result is presented but consider a search for "paris" and

receiving "comparison" as result. Obviously "comparison" is not the expected result but due to the underling it is clear why it is presented. Otherwise the user might be irritated by such a result.

A problematic aspect of underlining the *search pattern* is that the XML file only specifies the absolute coordinates and dimensions of the *entire word* and not of the specific substring we want to underline. *OmniPage* also supports coordinates *per character* but this results in much larger XML files (approx. 50 times larger) as one `<wd>` tag must be specified for each character. We rather suggest to *interpolate* the required coordinates. If assuming characters of equal widths this is done by dividing the width of the recognized word by its length, i.e. the number of characters, in order to achieve the character width in pixels and then adding that amount n times to x-coordinate of the word (the "`l`" parameter of the `<wd>` tag), where n is the position within the word minus one. As a result we achieve the x-coordinate of the beginning of the matched substring. The end is determined similarly and the y-coordinates are given by the "`b`" parameter of the `<wd>` tag. Note that some additional space is added between the emphasized characters and the underline (as well as for the border of the highlighting annotation).

Such *interpolation* will slightly misplace the underline, because typically *proportional fonts* are used (instead of *fixed size fonts*) and hence, the width of characters vary. Regarding the character width for several fonts requires knowledge of the applied font (which probably is provided by the OCR software) and also knowledge of any character widths of any possible fonts and therefore is too complex considering the issue. Since the letters "m" and "w" are typically wider for any font than "i" or "l" (with only minor differences between fonts), we rather suggest *interpolation* by use of well defined *default widths* instead of one width for all letters. The following table lists the proportional widths (of Java's default "Serif" font under SuSE Linux 9.2):

	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/	0	1	2
0.5	0.4	0.6	1.0	1.0	1.1	1.1	0.4	0.5	0.5	0.8	1.3	0.4	0.9	0.4	0.8	1.0	1.0	1.0
3	4	5	6	7	8	9	:	;	<	=	>	?	@	A	B	C	D	E
1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.4	0.4	1.3	1.3	1.3	0.8	1.4	1.2	1.0	1.1	1.2	0.9
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
0.9	1.2	1.2	0.5	0.7	1.1	0.9	1.4	1.2	1.2	0.9	1.2	1.1	0.9	1.1	1.2	1.1	1.5	1.1
Y	Z	[	\	]	^	_	‘	a	b	c	d	e	f	g	h	i	j	k
1.1	1.0	0.5	0.8	0.5	1.0	0.8	1.0	0.9	1.0	0.8	1.0	0.8	0.6	0.9	1.0	0.5	0.6	1.0
l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~
0.5	1.5	1.0	1.0	1.0	1.0	0.7	0.8	0.6	1.0	0.9	1.3	0.9	0.9	1.0	0.6	0.6	0.6	1.3

With these *proportional character widths* or *relative widths* the underline for the example "comparison", which has a relative width of 9.2, can be computed by the relative widths of the three substrings "com" ($3.3 \approx 35.9\%$), "paris" ($3.9 \approx 42.4\%$) and "on" ($2.0 \approx 21.7\%$) instead of the fixed sizes, which would be 3:5:2 or 30%:50%:20% (since "comparison" has 10 characters). The five letters of "paris" are only slightly wider than the three letter substring "com".

8.1.2 Interlinkage of Search Base and Indices

Applying *optical character recognition* to automatically generated slide images delivers a *textual basis* that is required to provide *full text search*. The *logical interlinkage* between the extracted text and an electronic lecture is automatically given by the *slide indices*. The *indices* provide the partitioning of the electronic lecture and thus correspond to the produced *slide images*. The *output* of the *optical character recognition* also corresponds with the *slide images* and thus the *slide indices* as well. Hence, each portion of the search base is associated with an index. If using the XML format of *OmniPage*, each `<page>` tag contains the recognized characters for a single page and subsequent `<page>` tags correspond with subsequent indices. If storing the output of the OCR application as ASCII text either one file per slide can be stored or, as we prefer due to resulting in fewer files, one file that includes the text of all pages and individual pages are divided by a *form feed* (ASCII character `0x0C` *hex*).

The retrieval algorithm returns the *indices* of the slides where the corresponding text matches the search pattern and, if coordinates are provided, also emphasizes the appropriate areas. As each search can be performed in a few milliseconds, the search results are updated while the user enters a keyword. Therefore it is not necessary to enter entire words and start the search process by pressing a dedicated *search* button. In fact, the user gets *permanent feedback* after each additionally entered character and hence can stop whenever an appropriate result is presented or the entered character sequence cannot be found. The *TeleTeachingTool* supports accessing the *matching indices* either by use of a clickable *thumbnail overview* that only displays the matching pages as well as by performing the search multiple times and thus accessing the respective next occurrence. The matching words are emphasized within the presented slides and the thumbnails.

The described search function is *slide-based*. However, we can make it more fine-grained. Hereunto, we need adequate indices to divide the recording into smaller sections. Presentations with overlays generate more frequent framebuffer changes, all of which can be used as indices not only for navigation but for splitting sections. As suggested in Section 7.4.2, the *RemoveAllAnnotations* events also offer a more fine grained partition.

Indices provide the *logical interlinkage* between text and the recorded desktop. Search results are slide indices and thus we get the beginning of the teacher's comments about a slide (matching the search pattern), but not the precise comments about the searched string. The built-in annotation system of the TTT offers a possibility of an interlinkage with the audio recording. The highlighting feature is generally used for marking words or text parts, which the teacher comments on at exactly that moment. And as the presenter only points out important elements, we may have localized comments about interesting words, which are more likely to be searched. Using timestamps of highlighting events and applying text recognition to highlighted areas results in text that can be linked to the timestamps of the corresponding annotation events and thus possibly to audio comments about the highlighted text, which provide more precise search results. Pointer stop positions may also be used to create such interlinkage since the pointer can be used to point to something. Although detection via pointer events or animation detection can be

done, it is hard to identify which area the pointer is pointing to or if it is pointing to something important at all. Therefore results may be very error-prone and thus are not useful.

Another possibility for interlinking the *verbal narration* and presented *textual content* would be to use an *audio transcript* of the recorded audio stream, i.e. a *textual extraction* acquired by *speech recognition*. As [Hürst, 2003] suggests, such an *audio transcript* is another *retrieval option* for multimedia-based data such as an *electronic lecture*. *Speech recognition* typically results in worse recognition rates than *character recognition* but, analogous to the search based that is produced by character recognition, badly recognized words will not harm the retrieval process, because such words will typically not be searched and thus will never be presented to the student. Due to the redundancy in spoken words and as recognition errors mostly affect small filling words, the influence of recognition errors for retrieval purposes is rather negligible [Garofolo et al., 2000, Hauptmann and Wactlar, 1997, Thong et al., 2000].

8.1.3 Recognition Improvements

In order to improve the recognition results, we have tested the influence of different *input formats*. The input to the OCR software is screenshots as given by the automatically generated html script. However, as the pages of the script might be *annotated* but annotations may confuse the text extraction process if annotations overlap with characters, we will not use the script pages but store slide images that are *optimized* for character recognition without irritating annotations and mouse cursors (unless recorded pixel-based as part of the framebuffer).

Additionally we have tested different *color formats*. The outputs of the tested OCR applications (Scansoft's *OmniPage Pro 11* [OmniPage, 2006] and ABBYY's *FineReader OCR 7.0* [Finereader, 2006]) were very similar for *full colored* and *grayscale* sources. *Black and white* input offers better contrasts, which reduces the error rate. Unfortunately all characters written in light colors were reduced to white and therefore were no longer distinguishable from the background. Hence, if using black and white images, the contrasts between pixels must be respected in order to keep all characters visible, which is problematic if using differently colored texts and backgrounds. Note that these tests were performed during our earlier research at the Universität Trier. The later long term usage at the Technische Universität München exposed that the results of Scansoft's *OmniPage Pro 14 Office*, which supports XML outputs, are less good if compared to *FineReader*.

These results are application dependent and different pieces of OCR software may reveal other results. Hence, we cannot state the perfect input format for *optical character recognition*. However, this is not the intention of our research and especially not of this thesis. We rather want to demonstrate that *full text search* does not necessarily require storing electronic lectures symbolically or to access the presented source documents.

8.1.4 String Distance Metric and Stemming

In order to match incorrectly recognized search words with a user's query or to correct misspelled or mistyped user queries, we have tested the well-known *Damerau-Levenshtein-Metric* [Damerau, 1964, Levenshtein, 1966], which is used to determine how similar two strings are. The *edit distance* between two strings is defined to be the minimum number of operations needed to transform one string into the other, where an operation is an *insertion*, *deletion*, or *substitution* of a single character. However, this resulted in too many unwanted results, especially if regarding that the user's query and the recognized text may contain errors. Consider, for example, the words "Zelle" (German word for "(memory) cell") and "Keller" (German word for "stack"), which are often used in the *compiler construction* lectures. The *edit distance* is only two:

$$\text{“Zelle”} \longrightarrow_{substitute} \text{“Kelle”} \longrightarrow_{insert} \text{“Keller”}$$

Now recall that we perform a *substring search*. Hence, the distance between "Zelle" and "Kelle[r]" is only one due to omitting the last character. Although one is the smallest possible distance of the *Damerau-Levenshtein-Metric*, such results probably irritate the user, especially if the *search patterns* are not marked with the *search results* or if the *search results* are not emphasized at all. Similar irritations are achieved by applying "stemming" algorithms, which determine a *stem form* of a given inflected or derived word form.

Presenting incorrect or unwanted results may confuse the student (as is the case for meaningless indices). As our first priority is to *present meaningful search results*, it is better to find fewer but correct results, than to find any results but also present some irrelevant and thus confusing ones. Hence, we prefer *exact matches*. However, special characters (including unprintable ones) may be ignored and German umlauts may be treated like the corresponding standard vocals.

8.1.5 (Semi-)Automated Workflow

Our aim is to provide *full-fledged electronic lectures* by a *lightweight* lecture recording process. Hence, the *extraction* and *interlinkage* of a *search base* should be automated as far as possible. The indexing and the generation of the slide images, which are used as input files for the OCR software, is automated by the *TeleTeachingTool* as described in Chapter 7. The *recognition process* is also automated within the OCR software. All *slide images* are selected as input. Then the automated recognition algorithm is applied. Optionally the user can correct the recognition results manually. Finally the results are stored in the selected file format. Nevertheless, the OCR application must be started and the reading of the input files must be initiated manually.

Instead of using an external OCR application, an integrated solution would rather by preferable. However, the results of sophisticated commercial OCR applications are remarkably better (due to the applied algorithms and large dictionaries of up to

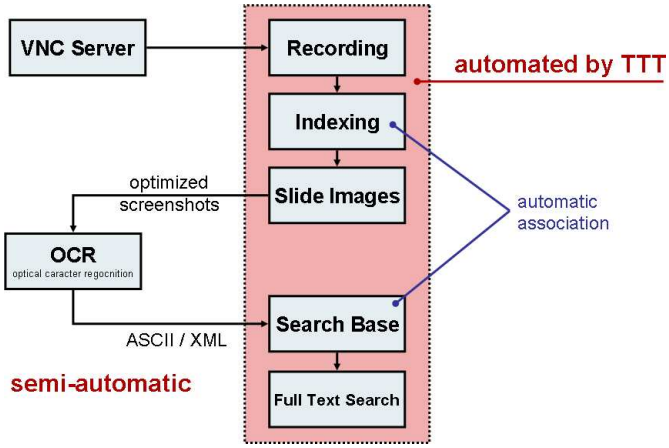


Fig. 8.3. Workflow to integration a search base

100,000 words) if compared to freely available *optical character recognition* solutions and moreover many less advanced algorithms must be trained before achieving good results. Other solutions to increase automation would be an appropriate programming interface to an external OCR application or the usage of batch processing, which is offered by some commercial OCR applications. Due to the high pricing of such OCR applications, we currently have no access to appropriate systems and therefore cannot test these approaches.

Nevertheless, the current semi-automated solution of the *TeleTeachingTool* integrates *full text search* functionality into *pixel-based recordings* within a few minutes demanding a few manual operations only (mainly limited to initiating certain tasks). By providing *full text search* for electronic lectures that are produced by the flexible *screen recording* approach, we have eliminated another main drawback when compared to the *symbolic recording* approach.

8.2 Lecture Profiling and Metadata

VNC session recording preserves the presented desktop, which is sufficient for replaying a recorded presentation. However, in order to produce and also to describe *asynchronous electronic lectures* some additional information, commonly called *metadata*, is required, for instance the *name of the teacher* and the *topic* of the lecture or presentation. As lectures are typically part of a course or lecture series, each *electronic lecture* should also reveal the corresponding *course* as well as a *sequence number* or *date* so that the lectures can be classified and arranged in order. Some metadata, for instance the *duration* of the lecture, are given implicitly by the recorded session (the *duration* is obviously the distance between the smallest, typically zero, and the largest timestamp). Other data must be specified explicitly, for instance the *teacher's name*.

8.2.1 Metadata by Lecture Profiling

In order to initiate a recording process, several parameters, e.g. the VNC server exporting the desktop to be captured, the color depth or enabling/disabling video recording or transmission, must be specified. The system should offer meaningful predefined values as far as possible and if the recording environment preserves the last used parameters, the values need to be specified only once. However, if the same recording setup is used by several people for multiple lectures all demanding different parameters, errors are likely to occur. Even if unable to avoid hardware related failures, like pulled plugs or empty batteries in microphones, the recording software should at least give suitable guidance on the software side to reduce errors. Therefore we suggest *lecture profiling*.

A *lecture profile* is a set of parameters, which contain settings that are essential for recording purpose. Additionally, a profile may contain useful *metadata* like the title of the lecture (series) and the teacher's name. As the *ease of use* is very important a suitable *user interface* for *lecture profiling* is required. We have implemented a concise *lecture profiling user interface* for the *TeleTeachingTool* (Figure 8.4). Whenever a lecture of a certain lecture series should be recorded, it is sufficient to select the appropriate entry from the list of all previously used lecture titles and all other parameters will be automatically loaded again. A designated loading and storing of profiles via menus or buttons is not encouraged as novice users should not be overloaded with terms and means of *lecture profiling* but should rather be able to record something forthright and notice later that parameters are stored for their convenience.

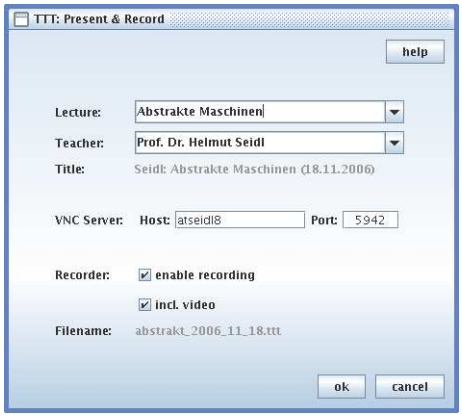


Fig. 8.4. Lecture Profiling GUI of the TeleTeachingTool

In order to achieve *consistent naming* of files, our profiling mechanism generates titles and filenames automatically depending on the title of the lecture, the teacher's name and the current date. Subsequent recordings are distinguished by an additional sequence number at the end of the filename, which is useful, for instance, if a teacher prefers to make a pause during a lecture or if recording a series of talks during seminars or conference events. Not only the names of the recordings should

be consistent but also the names of the corresponding *search base* used for *full text search* and the automatically generated script, which consists of several html and image files. Manual naming is not only an unnecessary but also an error-prone task. *Consistent naming* is essential to provide *structured databases* of lectures and to *categorize* lecture recordings as is required in order to achieve a suitable representation of *cross lecture search* results, which will be discussed in the following sections.

8.2.2 Dublin Core Metadata

There are several approaches and discussions in the research community about standardizing metadata specifications for e-learning issues. As an example of a possible metadata set and how this set relates to our *electronic lectures* we will give an short overview of an simple model, the *Dublin Core Metadata Element Set* (also known as *Simple Dublin Core*) [DublinCore, 2003], which provides a simple, rather loosely-defined set of elements. The *Dublin Core Metadata Initiative* (DCMI) is also working on a set of terms which allow the *Dublin Core Metadata Element Set* to be used with greater *semantic precision* (*Qualified Dublin Core*). The *Dublin Education Working Group* aims to provide refinements for the specific needs of the education community. Details can be found at the Dublin Core website [DCMI, 2006]. Another common but more complex model would be *Learning Object Metadata* (LOM) [Wayne Hodgins, 2002].

The *Dublin Core Metadata Initiative* (DCMI), an organization dedicated to promoting the widespread adoption of interoperable metadata standards, has introduced the *Dublin Core Metadata Element Set*, Version 1.1 [DublinCore, 2003] as a standard for cross-domain information resource description. It contains *15 optional elements*, which can be more or less used to describe our electronic lectures. Some of the metadata is provided by the lecture profiles, others can be acquired from the recordings themselves (maybe after analysis algorithms have been applied), for example date, duration, indices or the full text search base. In particular, the elements are:

Title: The title of the lecture as specified by the profile.

Creator: The teacher's name is also specified by the profile.

Subject: The main topic should be given by the lecture's title as well but admittedly this "subject" might be a little vague as it is intended as the subject for a series of lectures and not for a single recording.

Description: The *full text search base* provides detailed information. However, more suitable descriptions like chapter headlines or summaries cannot be extracted automatically for pixel-based recordings (at this time). The DCMI specification allows non textual resources, like index thumbnails, to be added as well.

Publisher: The university, institute or company associated with the recording. Actually not supported by our lecture profiling system as practically all lectures produced with the same setup are provided by the same university but an appropriate "Publisher" or "University" field can be added to the profiling system easily.

Contributor: Same as publisher.

Date: The start time (and thus the date) of each recording is stored in its header.

Type: According to the recommendation given by the *DCMI Type Vocabulary* [DublinCoreVocabulary, 2004] one of several types could be assigned: "*Collection*" as a recording comprises several data streams, thumbnails or search base as text; "*Event*" describing the local lecture that was recorded; "*InteractiveResource*" which is used (among others) for multimedia learning objects or the type; or "*MovingImage*" could be assigned. However, regardless of which of these types is used, it should stay the same for all recordings and therefore the type element is irrelevant for the purpose of categorizing the recordings of a lecture archive.

Format: The DCMI recommends the use of *MIME* types (Multipurpose Internet Mail Extensions), which are defined for standard formats. However, lecture recording formats are not standardized at this time (unless standard video formats are used). Like the *type* element, the *format* element should be the same for all lectures and therefore is not essential as long as only electronic lectures are handled.

Identifier: The filename(s) or the filename base (without endings) should be a *unique identifier* within a lecture archive.

Source: This element could describe the sources (slides) of the presentation. However, a pixel-based recording approach abstracts from the formats of the source documents as it supports the parallel use of arbitrary sources. Hence, the source format is typically unknown for a electronic lecture and in fact is already unknown, because it is not important, to the recording environment.

Language: Right know we have recorded lectures mainly in German and some in English. Distinguishing them would be useful in order to categorize lectures and to restrict searches to a certain language, but this is not supported right now. However, the keywords specified to perform the full text search lead to results in the same language (except for internationally used terminology). An appropriate field can be added to the *profiling set* if desired.

Relation: This element should refer to related resources. Due to the predetermined *consistent naming* that is provided by the *lecture profiling system*, all recordings that belong to the same lecture series have the same filename except for the date component. This allows related lectures to be derived for a given recording. This is also the case for other automatically generated materials such as the html scripts.

Coverage: Typically, *coverage* will include a spatial location, a temporal period or a jurisdiction and therefore could refer to metadata used for the elements Publisher or Contributor (a university in the sense of company and location), the recording date or period. However, this element is very vaguely defined.

Rights: If the electronic lecture supports some kind of *rights management*, e.g. encrypted files, this can be noted here. Actually, neither our recording environment nor our lecture archive supports *rights management*. Lectures are freely available to the public. Note that download could also be limited by the web archive but in

such a case it does not influence the recordings and its metadata but the publishing process.

Note that any fields may be added to a *profiling system*. However, regarding the *ease of use*, the number of entries that must be specified by the teacher (or a technician) should be limited as far as possible and therefore one should consider seriously which entries are supported.

8.3 Cross Lecture Search

Lightweight lecture recording enables large *multimedia databases* to be built up containing hours of *asynchronous electronic lectures* rather quickly. Currently our archive contains more than 400 recordings resulting in approximately 600 hours of recorded presentations. Hence there is an increasing need for techniques not only to localize specific information within a single *electronic lecture* during replay but also to find a lecture that addresses a specific topic. Furthermore, an adequate presentation of the results is needed, which should offer *easy to use navigation* and a *rating system*.

8.3.1 Online Full Text Search

Our archive of lectures offers *cross lecture searchability* provided by a *PHP script*[2]. The *cross lecture search base* is composed of a set of all individual *search bases* of the single lectures. A *search base* is stored within the extended header of an *electronic lecture*. However, instead of extracting the *search bases* from the archived *electronic lectures*, we use the output of the OCR application directly. While packing and uploading our recordings, a script also copies the OCR output files to a dedicated directory on our web server where they can be accessed by the PHP script. In future, this packing and uploading process should be integrated into the *TeleTeachingTool*.

Search results are formatted as *html pages* (Figure 8.5), which presents the titles of the recordings that match the *keyword* and the list of *matching indices* for each lecture, all of which are linked to the automatically created *html script*. The *consistent naming*, which is achieved by the *lecture profiling* is essential to perform the interlinkage between *search results* and the *script pages* as otherwise links would refer to nonexistent files. *Metadata* such as the *lecture titles* or the *dates* and *durations* of the recording, are not included in the *search base* but can be extracted from the corresponding html files, because the structure of the html files is known as they were generated by the TTT.

Currently, following a link leads to a *static script* without any possibility to highlight search results or to access other results within the presented script page. *Dynamic web pages* offer a more suitable technique for presenting search results by combining

[2] a scripting programming language; PHP is a recursive backronym für *Hypertext Preprocessor* and originally stands for *Personal Home Page Tools*

the screenshots, thumbnails and metadata of the *static scripts* with a *dynamic link structure* that corresponds to the delivered search results. Furthermore, a *dynamic representation* enables the search results to be emphasized within each html script page. An appropriate system that provides a *cross lecture full text search* by generating *dynamic web pages* is currently being developed and implemented as a student project.

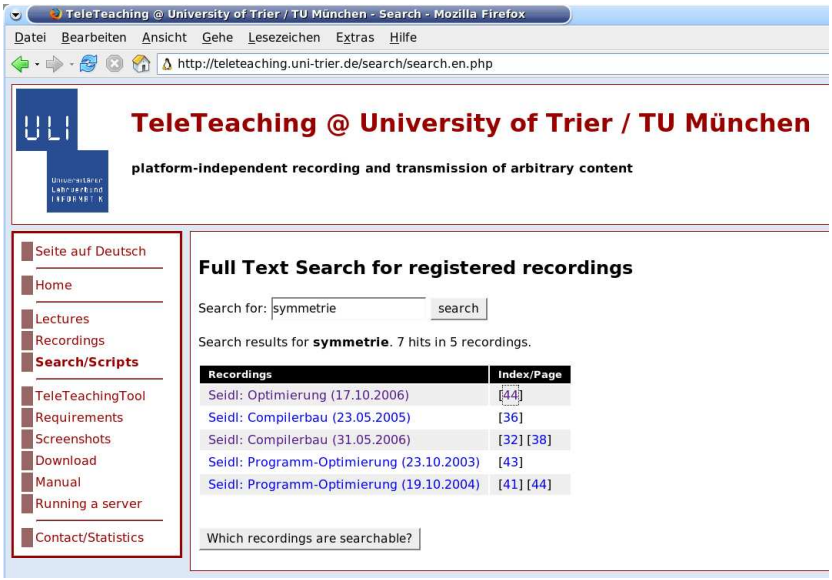


Fig. 8.5. Online Full Text Search

8.4 The Search&BrowsingTool

The *Search&BrowsingTool* (SBT) is an extension of the *TeleTeachingTool* and was implemented during a system development project by two students [Gruber and Leiter, 2006]. The goal was to provide *cross lecture searchability* with an easy to use graphical user interface. The SBT supports three kinds of sources: At first it can handle any *local lectures* that are accessible as (directories of) TTT files. Furthermore, it can search and browse through lectures available at our *online archive*. And finally an *offline search base*, which contains any information and metadata that are necessary to describe lectures for searching purposes, can be exported and imported and thus allow offline students to check contents, which they may access later (online or via DVD).

In order to perform a *local search*, the SBT gathers all TTT recordings located in specified directories. Lectures are not loaded completely as this would be very time consuming and memory intensive. Instead, only the *headers* of recordings including *metadata* (e.g. date and title), *indexing structures*, *search bases* and *thumbnails* are

read. As a further improvement this task is not performed for each search but only once. The SBT creates a *combined search base* that stores all needed data and is imported during startup and validated by use of checksums with the existing lecture files. Entries for new or modified files are generated or modified respectively. This *combined search base* is not only useful to enhance startup and search performances but additionally can be exported and distributed for students' usage as an *offline search base*.

The *online search* must be performed in a different way. As the recorded lectures consist of up to three files (desktop, audio and optional video stream), they are packed within a single *ZIP archive* in order to provide easier download. Obviously, downloading and extracting all available lectures just to perform a search is not feasible. Instead we make use of the existing functionality of the *online search feature* already. The *search string* of each performed query is sent as an HTTP request to a web service, which performs the *online full text search* and returns the results as an XML file. The returned XML document contains a list of results, each of which consists of a corresponding lecture title, a matching index (number) and a filename base. The *filename base* is valid for all files related to a certain recording and can be extended by appropriate file endings. Due to the *consistent naming* and the known *directory structure* of the server, the SBT can additionally access the automatically generated html script (including screenshots and thumbnails).

8.4.1 Views

Applying *full text search* to a single electronic lecture produces no more results than the maximum number of indices available, which are typically about 30–50. Therefore a *thumbnail overview* is a suitable visual representation of the search results. Note that there might be more matches within a single page but nevertheless each index is presented only once by the thumbnail overview.

Giving an appropriate representation of the results of a *cross lecture search* is more challenging. A simple (thumbnail) list of matching indices could be sufficient for say five to ten matches but is absolutely inadequate for a high number of results. The keyword "stack" for example is frequently used during our recorded lectures and therefore produces over 200 matches within about 80 lectures of a dozen different courses. The keyword "Beispiel" (German word for "example") even produces far more than 2000 results. Besides the memory consumption of about 10 Mbyte compressed files, which relate to 40–160 Mbyte uncompressed pixel values (8–32 bit per pixel), a *thumbnail overview* with 2000 entries can hardly be surveyed in a reasonable manner. Just presenting a list of results without displaying the thumbnails is also not very comfortable. Hence, another representation is needed.

In order to respect the varying numbers of results, we will categorize lectures by their metadata and introduce several representations, called ***views***, adequate for different purposes. Common for all *views* is a *tree representation* (see Figure 8.6 left hand side) that is labeled differently on each level, starting with the root node, which represents all lectures. The subsequent levels show the *categories* (title of lecture series), the *semesters* (as each series can be recorded multiple times in different

years) and finally *single lectures* by date, which are the leaves of the tree. A number next to each entry displays the number of *search results* that were found in the associated subtree.

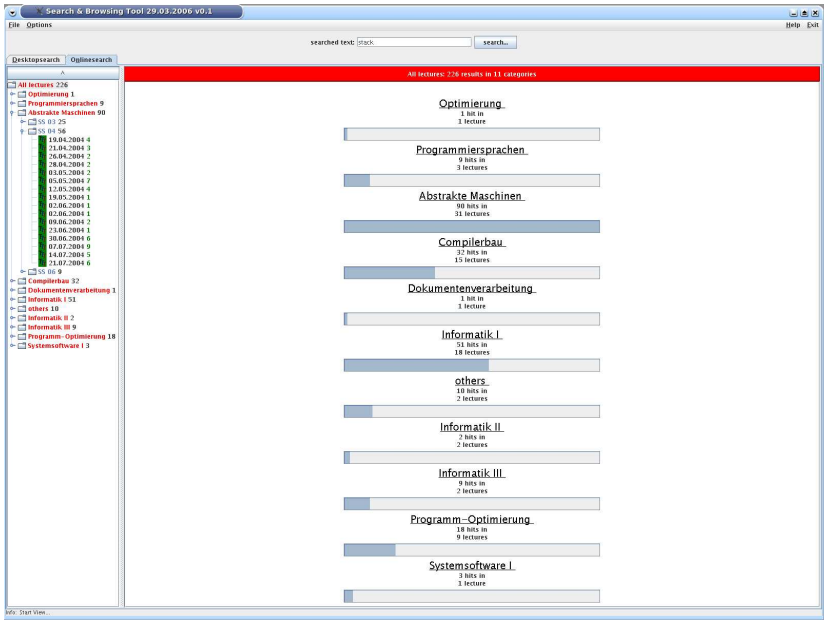


Fig. 8.6. Search results: Overview

The representation displayed in the main window depends on the number of results and the selected level within the tree. The top level representation, the ***overview***, which is displayed in Figure 8.6, shows a top to bottom list of *lecture categories*, which are entitled by the name of the lecture series. In the presented figure, we have searched for "stack" and received 226 results in 11 categories, which are listed as "*Optimierung*", "*Programmiersprachen*", etc. Additionally, the number of matching lectures plus the absolute number of matches in that category are listed. The *graphical bar* visualizes the number of results in relation to the maximum matches (of a single lecture) and thus provides some kind of *rating* for each entry. In the example the maximum is given by 90 matches (in 31 lectures) in the third category, which is entitled "*Abstrakte Maschinen*". The first entry in the list has only 1 match, the second has 9 matches. Hence, the bars are accordingly shorter. Clicking on an entry, either in the *view* in the main window or in the *tree representation*, accesses the next, more fine grained level and the *view* will be updated correspondingly. Note that higher levels can be accessed by clicking on a corresponding tree entry.

Level Views

Besides the *overview*, we offer the ***category view*** that represents a list of lecture series, the ***semester view*** for all lectures of one category within a certain semester, and finally the ***lecture view***, which corresponds with the leaves of the tree and

thus single lectures. The first two of these *views* show a top down list of entries with *rating bars* similar to the *overview*. Selecting the third entry in the example leads to the *category view* displayed in Figure 8.7. The course was recorded three times (summer 2003, 2004 and 2006). Again the number of matches is visualized by *bars*. The *semester view* (Figure 8.8) looks analogous except that the list entries represent single lectures, all of which were recorded in the same semester, and each entry is entitled by the respective recording date.

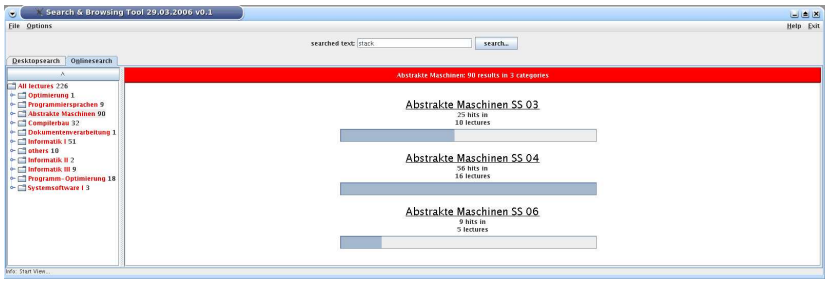


Fig. 8.7. Search results: Category View

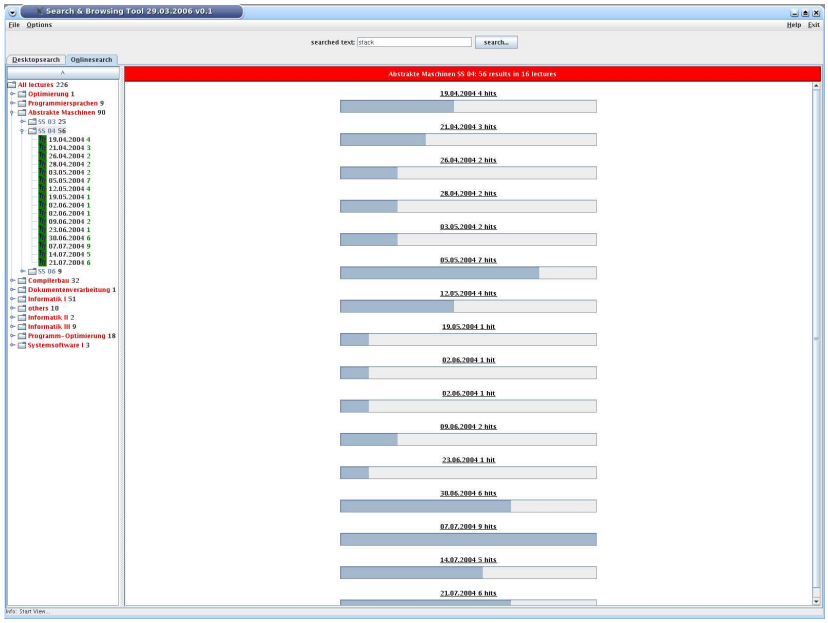


Fig. 8.8. Search results: Semester View

The ***lecture view*** (Figure 8.9) represents a single lecture and displays a left to right *thumbnail overview*, analogous to the representation used as the overview page of the *html script* (Figure 7.15 on page 147) but showing only matching instead of all indices. Thumbnails are entitled with index numbers.

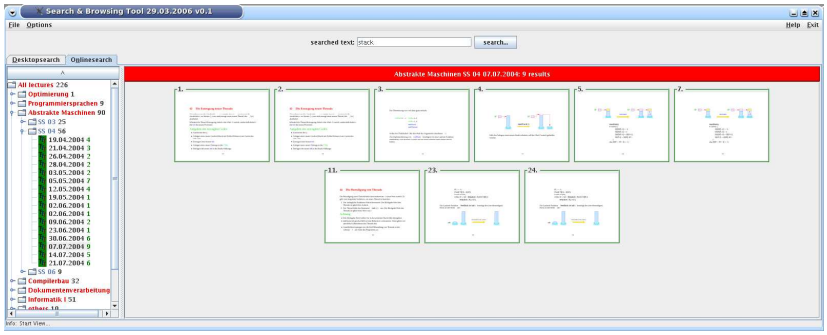


Fig. 8.9. Search results: Lecture View

Limited View

Besides the level related *views*, we provide one view for the special purpose of representing a small number of results only, the ***limited view***. It combines the top down lists with per lecture thumbnails as displayed in Figure 8.10. Each row displays the thumbnails of a single lecture and is entitled with the corresponding lecture name, semester, date and the number of results. If more matches for a lecture are found than will fit into a single row, only the first thumbnails are displayed and an additional "*more ...*" link appears, which enables access to all thumbnails of that particular lecture (by switching to the *lecture view*). *Graphical bars* for rating are not necessary because the number of the displayed thumbnails within each row is equivalent to the bar rating. The *limited view* can appear on any level of our tree, whenever the number of results falls below given *thresholds*. Preassigned, but adjustable by the user, are a *maximum number* of 64 results and an *upper bound* of 16 lectures. If the number of search results does not exceed these *thresholds*, the *limited view* will be displayed instead of the *view* that is associated with that particular tree level normally.

The *tree representation* of search results in combination with different *views*, which display either top down lists with rating bars or left to right thumbnails, and the special treatment of a small number of matches by use of the *limited view*, provide a *structured representation of search results* with different levels of granularity. Furthermore, the asynchronous electronic lectures are *categorized* by use of *metadata* (title, filename and date) in order to give useful hints to the user, who possibly is interested in a certain course or a certain semester only.

8.4.2 Accessing Search Results

Actually students do not only want to search and find topics but certainly want to access them as well. As the *cross lecture search* can use different *search bases* (local, online, offline), appropriate activities must be performed. The colors of the thumbnail borders and the tree leaves indicate from which of the three possible *search bases* (local: *gray*, online: *green*, offline: *red*) a lecture originates and applicable actions can be accessed via each thumbnail's *context menu*. If the electronic lecture is locally

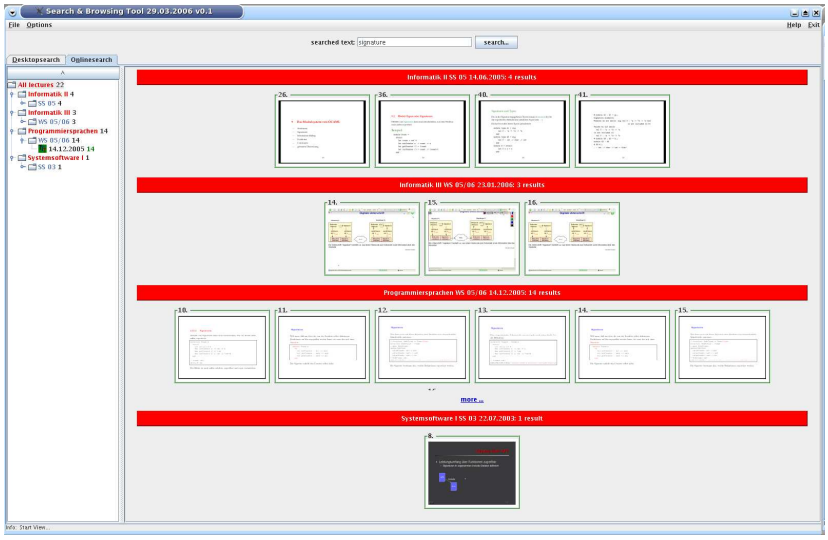


Fig. 8.10. Search results: Limited View

available in the user's file system, the *replay* can be initiated, starting at the index that matches the selected search result. Otherwise the appropriate lecture can be *downloaded* from the web archive to be unpacked and replayed locally. Additionally, the *Search&BrowsingTool* can display a *fullscreen image* of the selected thumbnail or access the associated *html script* in a web browser. The screenshots and the html scripts are available online. The thumbnails are read from local files if present or from the web archive as well.

8.4.3 Browsing

The *Search&BrowsingTool* does not only offer *retrieval features*, but, as the name suggests, additionally offers the functionality of *browsing* through the contents of local and online lecture databases. *Browsing* is performed as a search without specifying a keyword and thus delivers any lecture (indices) as result. The presentation of the indices and accessing content (replaying or downloading lectures or accessing the script) stays the same. Additionally the SBT offers the possibility of comparing the *local lecture database* with the lectures that are available in the *online archive*. Via HTTP request a web service delivers the lectures which are available for download. In synchronization with the recordings that are found on the local file system, an overview displays the list of available lectures and marks which of them are already locally available. The user can now select missing lectures and initiate their download.

8.5 Searchability by Web Search Engines

Searchability by conventional search engines by generating electronic lectures, which are indexable by *web search engines*, is also a desirable feature in order to improve the retrievability of such learning material [Mertens and Rolf, 2003]. Until appropriate *standard formats* or *metadata models* are available, we suggest making use of the data formats that are currently indexed by *web search engines*, which are *textual documents*. Due to the recognition artefacts, our *search bases* are not suitable to be presented but nevertheless give a meaningful *abstract description* of the presented content. Integrating the *search bases* and other *metadata* as appropriate `META` tags into *html documents*, which enables the specification of, for instance, the author, a content description and keywords, allows *web search engines* to index such textual content. As such indexable meta tags can be added to the automatically generated html scripts as well as to the download pages, the suggested searchability by conventional web search engines can be achieved. If a page of one of our html scripts is referred as a result in regard to a performed web search, the user will not see the textual content, which is hidden in the meta tags but nevertheless be led to this result, but rather the pixel-based presentation of it, which is equally sufficient (to the user). If a *download page* is presented as a web search result it might not be evident why this page is presented. Therefore a meaningful short description is advisable but we currently do not see how this can be fully automated. However, teachers (or their staff) probably can be asked to deliver at least a short course description, which commonly is available at their web pages anyway.

8.6 Summary

This chapter considered the Criterion C8: Information Retrieval and Criterion C5: Metadata. *Screen recording* offers a flexible technique for *lightweight lecture recording* and *automated indexing* compensates for many drawbacks caused by the missing structure. In order to provide *full text search* for *pixel-based recordings* we suggest the use of *optical character recognition* in order to extract *search bases* from the *electronic lectures*. Instead of accessing various source documents of arbitrary document formats, our approach enables us to derive a *search base* directly from an electronic lecture. The process of storing slide images as input for the *optical character recognition*, the extraction of text (and coordinates) and the association of search base parts with indices is performed semi-automatically. Providing an appropriate programming interface, the character recognition could be fully integrated to provide full-fledged *electronic lectures* with *indexing structures* and *full text search* without manual post-processing.

During playback students can specify keyword(s) to initiate *full text searches*. The results refer to slide indices of matching pages and are presented as clickable *thumbnails*, which are linked to the corresponding position within the recording. Applying an *optical character recognition* application that provides *XML output* (or appropriate other formats) with *coordinates*, even the occurrences of the keywords within each slide (image) can be emphasized (including underlining the matching substring).

In order to find relevant recordings within large databases of electronic lectures, additional retrieval features are required. *Cross lecture search* is addressed in the form of an *online search* that is implemented as a PHP script and is available at our web archive. Extending the currently available static representation of search results to a *dynamic* version will improve the usability. Another approach of providing *cross lecture searchability* is shown with the development and implementation of the *Search&BrowsingTool*. It *categorizes* the archived electronic lectures by use of *metadata*, which is provided by the recorded lectures or related web pages and offers different graphical representations, the *views*, for different category levels and numbers of entries within these levels. As the *Search&BrowsingTool* supports different search bases (local, online and offline) the environment furthermore provides different possibilities to access the found entries by download, replaying or presenting static images. Furthermore, we have discussed possibilities to provide *searchability be conventional web search engines*.

A *consistent naming* of all related files is essential for any *automated processing* or to achieve an *interlinkage* between various elements. In order to avoid error-prone manual naming, we have introduced an easy to use *lecture profiling* system, which automatically names all files to fit a predetermined scheme. Moreover, each *profile* provides the *parameters* that are required to perform lecture recording within the *TeleTeachingTool* environment. And finally the profiles contain some additional *metadata*, like lecture titles or teachers' names, which can be used to enrich the representation of search results and which are needed to categorize the hundreds of lectures available in our archive.

The presented work is designed to be used in our *TeleTeachingTool* environment but can be adapted for other recording tools or lecture archives. However, the use of *standardized metadata* and *interfaces*, describing how to perform *full text search* and how to *access content*, would be beneficial to create a commonly applicable infrastructure. This thesis does not intent to provide a sophisticated wide ranged *metadata model* nor to address detailed *retrieval issues for large databases* (of electronic lectures) but rather proves that the *flexible screen recording approach* does not necessarily conflict with *searchable electronic lectures*. By providing the *basic retrieval features* for *pixel-based recordings* in the form of a *search base*, *structured recordings* and some useful *metadata*, we give the basis to apply research results that previously were valid for symbolic recordings only. For instance, [Hürst, 2003] claims that the established retrieval systems, which are commonly designed to process large textual databases, are not suitable to handle slide presentations and therefore he discusses the *classification and presentation of search results* for *sequences of slides*. Another retrieval aspect for electronic lectures is the *retrieval of spoken words*. [Hürst, 2003] suggests applying *speech recognition* to the audio stream and extract an *audio transcript* of the recorded *verbal narration*. *Audio retrieval* is not a dedicated topic of *screen recording* but rather can be applied to any other recording approaches and is addressed further in [Garofolo et al., 2000, Hauptmann and Wactlar, 1997, Thong et al., 2000, Hürst et al., 2003].

In summary, we have diminished another drawback of the flexible *screen recording approach* by developing and implementing a system that produces *pixel-based* but

nevertheless *searchable lecture recordings*, and this in an *automated* fashion. Hence, we have further improved the *usability* of *pixel-based electronic lectures*.

9

The TeleTeachingTool

The *TeleTeachingTool* (TTT) offers an *implementation* of most of the *ideas and concepts* that are suggested in this thesis and was developed for everyday usage in close relation to our own requirements and experiences. The TTT is a *lecture recording, transmission* and *replaying environment* with integrated *automated post-processing* functionality based on *flexible, pixel-based desktop capturing technology* but unlike other screen recorders, the TTT offers advanced navigational features such as *slide-based navigation* and *visible scrolling* and furthermore enables *full text search*. Each *electronic lecture* that is produced with the TTT consists of three streams: the *audio stream* that preserves the teacher's verbal narration, the (optional) *video stream* that shows a video of the teacher, and the *desktop stream* that delivers the *framebuffer updates*. We use standard formats for audio and video transmission and recording. The *desktop stream* is transmitted by use of a *modified RFB protocol* in order to achieve a higher degree of scalability as suggested in Chapter 4. The *desktop recording* (file ending "`.ttt`") consists of a *header* and *logged messages* as suggested in Chapter 5 (plus optional *extensions*).

In order to prove the *ease of use* and the *intuitive operation* of the *TTT environment*, we will first describe how to use the *TeleTeachingTool* from a student's and a teacher's point of view and address the (optional) *post processing* and *publishing* possibilities. Furthermore, we give some usage scenarios of how we are using the TTT environment in order to record lectures. Afterwards, we address the *Java Media Framework* (JMF), which is used to capture, handle and replay the audio and video streams and furthermore, we give a specification of the *file format* for recording the *desktop stream*.

9.1 TTT Viewer for Students

The *menu bar* of the *TeleTeachingTool* consists of different main entries, which are entitled "Student", "Teacher" and "Post Processing", and relate to certain groups of people and tasks.

A student can either open and replay an *asynchronous electronic lecture*, i.e. a previously recorded lecture, or connect to a *synchronous electronic lecture*, i.e. a live transmission.

9.1.1 *Asynchronous Electronic Lectures*: Replaying recorded lectures

After selecting the "Student→open..." menu entry (Figure 9.1), a *file request dialog* is prompted and the student can specify which recording should be replayed. Afterwards that recording will be loaded and the replay will be initiated.

Fig. 9.1. Opening a recorded lecture

Figure 9.2 displays the *Replay GUI* of the TTT. The recorded *presentation* (including the *annotations*) is dynamically replayed within the *main window*. The *teacher video* (if available) is shown in the top left corner above the *thumbnail overview*, which displays small preview images of the recognized *indices* (typically corresponding to *slides*). If no video is available or if the video is turned off, the additional space is taken by the *thumbnail overview*. The *thumbnail overview* emphasizes the currently presented slide by a red border, which is automatically updated in relation to the replayed sequence. The user can watch other indices by use of the scroll bar and access each slide simply by clicking the corresponding thumbnail (*slide-based navigation*).

The *control bar* at the bottom provides the standard controls, which are *play/pause* (depending on the current state), *stop* (reset replay to the beginning) as well as accessing the relative *previous* and *next index*. The main part of the *control bar* is occupied by the *timeline slider*, which represents the *timeline* of the lecture from the beginning (left) to the end (right). The *knob* of the slider represents the *current playback time* and its position is automatically adjusted in relation to the replayed sequence. The student can set the *replay time* to any point in time (within the lecture's duration) just by clicking the corresponding position on the *timeline*. Furthermore, the student can drag the knob along the timeline in order to browse through the lecture by *visible scrolling*, i.e. the display is updated instantaneously while dragging the knob. Left of the *timeline slider* are two *labels* representing the *current replay time* and the *duration*, respectively.

The controls in the right bottom corner provide possibilities to adjust the representation of the playback window as well as the volume. The *volume controls* are only presented after pressing the *volume button* and allows the *volume level* to be adjusted via a slider or to be temporarily *muted*. The other control buttons are used

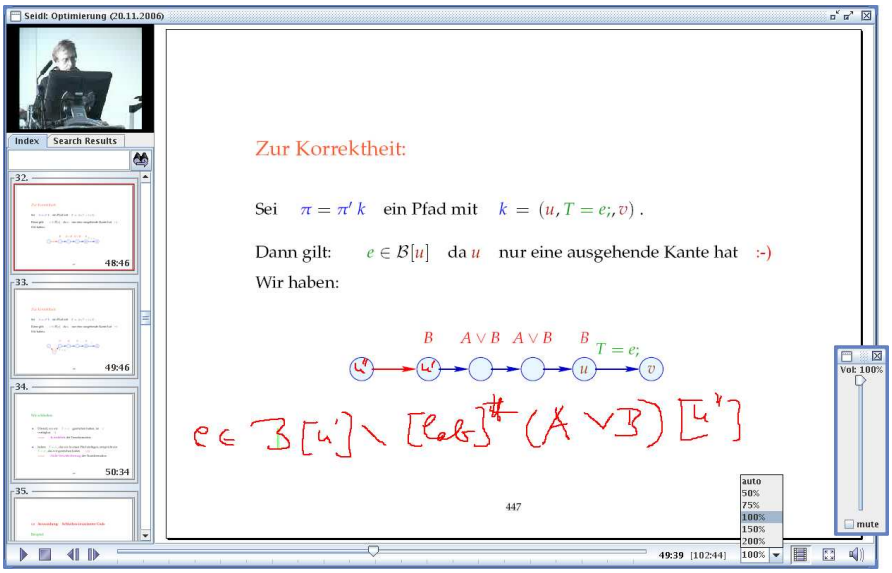


Fig. 9.2. Replaying a recorded lecture

to switch between the *windowed* and the *fullscreen mode* and to hide (or display again) the *thumbnail overview* and the *video component*. Furthermore, the *scaling* of the *main window* (which displays the lecture) can be adjusted, either by selecting one of the a predefined *scaling levels*, which are "50%", "75%", "100%", "150%", "200%" or "auto" (i.e. automatic adjustment according to the window's size), or by specifying the scaling level manually, i.e. by entering the desired level. If the *main window* is smaller than the (probably scaled) resolution of the lecture, the student can select which subregion should be displayed.

Full text search is initiated by entering a *search pattern* (a string) in the *search field*, which is displayed above the *thumbnail overview*. A *full text search* will be performed while entering characters, i.e. the *search results* will be updated whenever the next character is entered. The *search results* are presented within the *thumbnail overview*. *Preview images* of all *matching indices* will be displayed and, if supported by the lecture, the corresponding matches are emphasized within the *thumbnails* and the *main window*. The corresponding slides can be accessed by clicking on a *search result* in the *thumbnail overview* or the *next search result* will be presented whenever performing the (same) search again by hitting the "enter" key (in the *search field*) or by pressing the *search button* left of the *search field*. Selecting the *index tab* at the top of the *search field* will redisplay again all *indices* within the *thumbnail overview* (instead of the *search results* only). The user can switch back to the *search results* by selecting the appropriate tab.

9.1.2 *Synchronous Electronic Lectures*: Attending live lectures

Via the "Student→connect..." menu entry (Figure 9.3), the student can either specify a *TTT Server* manually or select an entry from a list of previously used or pre-assigned *TTT Servers*.

Fig. 9.3. title

If connecting manually, a *Connect Dialog* (as shown in Figure 9.4) will appear and ask for the *name* (*host*) and *port* of a *TTT Server* and furthermore offers the possibility of choosing whether the transmission should be provided as *unicast* or *multicast*. *Multicast* is preferable but might not work for all network connections (see Section 4.2.10 on page 74).

Fig. 9.4. Connection dialog

If a connection to the specified *TTT Server* can be established, the *live replay* will start. Since *navigational controls* are meaningless for live attendance, the *Live Replay GUI* will comprise the *main window* (which displays the lecture and annotations), the *video component* (that displays the live video of the teacher), the *volume controls* and the controls for the *scaling*, the *fullscreen mode* and to enable/disable the *video component*.

9.2 Teacher Component: Presenting and Recording

The *presentation and recording process* is initiated by selecting the "Teacher→present & record" menu entry as shown in Figure 9.5.

Fig. 9.5. Starting the presentation and recording process

Now the *profile dialog* will be opened (Figure 9.6). For a *quick start* it is typically sufficient to specify only the *VNC Server*, which delivers the desktop that should be presented, by entering its *host name* and *port*. However, the teacher can adjust additional parameters. It can be selected whether the lecture and the video should be recorded or not, and the names of the teacher and the lecture or course can be set. The *lecture name* will be used as the *profile name* and all options are automatically stored within this profile and loaded whenever that profile is selected again (by use of *Java*'s own preferences backing store/registry). In order to present and record the next lecture within the same series it is sufficient to select the appropriate *lecture name* as shown in the left screenshot in Figure 9.6 and the coressponding *lecture profile* will be loaded implicitely and any fields will be filled with the previously used values of that profile. There is no need to explicitly store and load *lecture profiles*.

The *teachers' names* are also cached and can be selected from the list whenever a new profile is generated. As most teachers will use the same parameters for their different courses, they can select an already specified profile and generate a new profile just by changing the *name* of their lecture. A *new lecture profile* will automatically be stored under the new name. Note that the other (old) profile will stay unmodified.

In order to reduce the number of parameters that must be specified in order to initiate a recording process and to provide *consistent naming*, the TTT sets the *title of the lecture* and the *filename* of the produced recording(s) according to a predefined name scheme, which regards the *lecture* and *teacher names*, the *current date* and a sequence number (if recording more than one lecture (part)).

As entering and selecting parameters in the *profile dialog* is rather intuitive and as any storing and loading of *lecture profiles* is performed automatically, there is no need to introduce the concept of *lecture profiling* to the teachers. In fact, it is even not required that they know the presence of such a concept. The next time they start the *TeleTeachingTool* they will notice that the parameters have been cached.

Nevertheless the *TeleTeachingTool* offers the option to *import* and *export profiles* if they should be transferred to another machine or stored permanently. Furthermore, profiles can be removed from the list. These features are accessible via appropriate menu entries, which are shown in Figure 9.7.

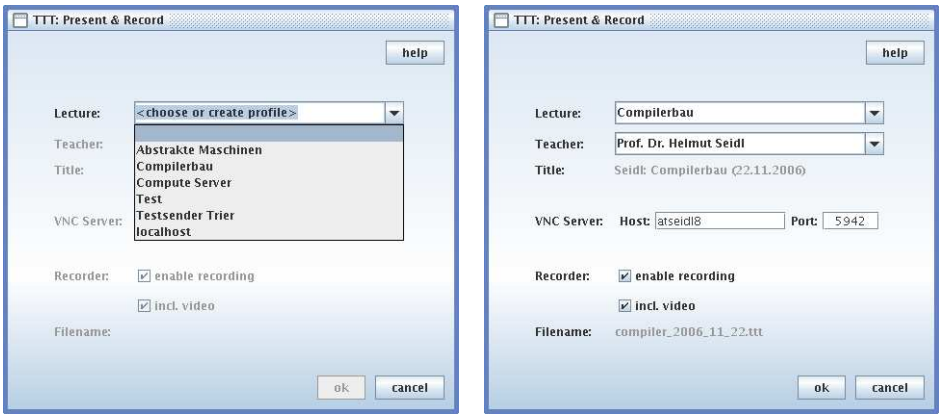


Fig. 9.6. Lecture parameters and profiles

Fig. 9.7. Import and export of *lecture profiles*

9.2.1 Presentation Controls

After the teacher has specified the session parameters or selected the appropriate *lecture profile*, the connection to the *VNC Server* and, if enabled, the *audio* and/or *video devices* are initialized. The teacher will see a *presentation GUI* as shown in Figure 9.8. In the *main window* the presented desktop will appear, which can be controlled by use of the mouse and the keyboard like any other desktop. Mouse movements, menu selections or any applications are presented dynamically within the *main window*.

The *control bar* above of the *main window* provides the *annotation controls* (left hand side) and the *recording controls* (to the right). The *reconnect button* (outmost right) can be used to reset the connection if any network problems occur. Beside the *reconnect button* is a button to switch between the *windowed* and the *fullscreen mode*, which typically is preferred as it removes any borders and thus provides more space for the presentation.

Starting a recording is initiated by simply pressing the red *recording button* and stopped by clicking the *stop button* (only enabled while recording is in progress) left of the *recording button*. After the recording process is terminated the teacher can instantaneously replay the recorded session by pressing the *play button* or initiate another recording process by pressing the *recording button* once more. Note that the sequence number of the filename will automatically be increased by one.

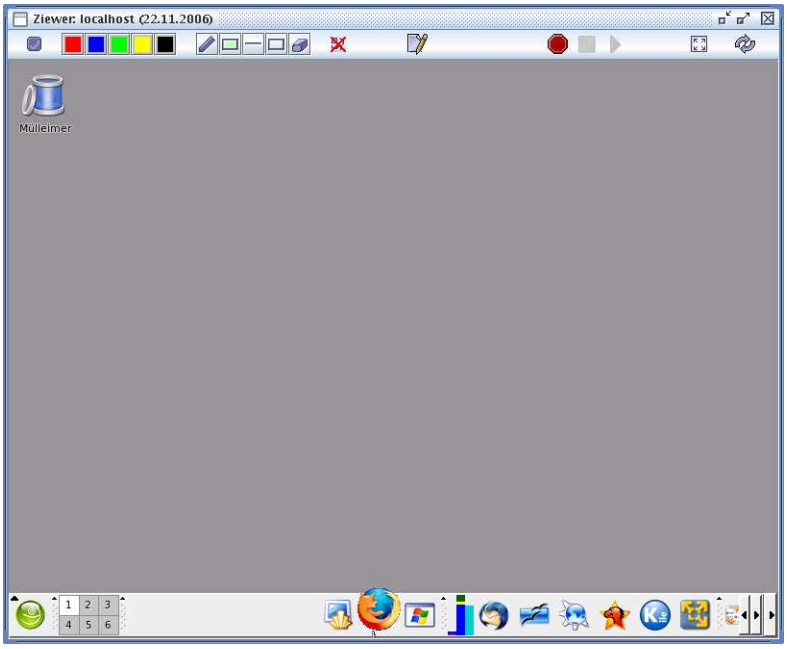


Fig. 9.8. Presenter GUI

Typically, the presented VNC desktop is controlled via keyboard and a pointing device like any other desktop. Furthermore, the pointing device can be used to annotate the desktop, which can show, for instance, a slide presentation. Hence, we need to switch betwen the *interaction* and the *annotation mode*, which can be done by the outmost left button (always showing the current selection). Note that any key presses are always forwarded to the presented desktop. Hence, the desktop can be controlled via keyboard while the *annotation mode* is enabled. This is very useful as, for instance, the pointing device can be used to annotate a presentation while the keyboard can still be used to switch to the next slide without disabling the *annotation mode*.

The other *annotation controls* are structured in *color buttons* and *mode buttons*. Clicking on either of these buttons will automatically switch to the *annotation mode* (as typically the selection will be followed by annotating). The available *paint modes* are (from left to right) *freehand*, *highlight*, *line*, *rectangle* and *delete*. The *highlighting mode* is used to focus the attention of the audience and the other modes are intended to add comments or sketches. Annotations are always applied in the currently selected *color mode*. While in the *delete mode* any previously made annotation can be deleted by selecting it with the pointing device. While in annotation mode, clicking at any point that corresponds to an annotation will remove the complete annotation, because each annotation is applied as one object and thus no partial annotations can be removed. The button left of the *mode buttons* is the *clear button* that can be used to remove all currently visible annotations from the screen. Furthermore, all keys that are typically used to switch to another slide (the page-up/down and arrow key) *automatically initiate* the removal of all current annotations since the

annotations are typically meaningless to another slide. Annotations are recorded and dynamically replayed in the same order and time scale as they were presented or deleted during the live lecture.

The *whiteboard button* in the center of the *control bar* switches between the *desktop* and the *whiteboard*, which is a blank white page that can be used for additional annotations. Note that there are different sets of annotations for the desktop and each whiteboard page. Hence, the *whiteboard* can be annotated independently of the presentation (or any other application), which enables providing additional comments on-demand.

As typically all key presses are forwarded to the presented desktop, the *TeleTeachingTool* defines only a few *key shortcuts*. The keys and functions are:

F9 switches to the next *color mode*;
F10 switches to the next *paint mode*;
F11 switches to the next "*task*" at the presented desktop;
F12 switches between *interaction* and *annotation mode*.

Note that "F9" and "F10" will implicitly activate the *annotation mode* (if not activated yet). "F11" simulates a so called "*task switch*", which (for most desktop systems) is typically caused by the key shortcut "ALT+TAB" and enables the *active window* to be selected without using the pointing device (which otherwise is used to activate an application by clicking on the corresponding window or task bar entry). The selection is typically performed by pressing the "TAB" key multiple times (in order to select the next window) while the "ALT" key is held down and selecting an entry by releasing the "ALT" finally. However, performing this procedure while controlling a remote desktop within the *TeleTeachingTool* results in switching between the *local* applications and not between the *remote* ones. Nevertheless this feature would be very useful to switch between applications very fast (for instance the presentation and a simulator or a programming editor) Therefore, the *TeleTeachingTool* simulates a hold down "ALT" key when pressing the "F11" key. The presented remote desktop will typically show a *task switch menu* then. By pressing the "F11" key multiple times, the teacher can select the desired entry. The simulated "ALT" will be released whenever any key other than "F11" or a mouse button is pressed. Hence, switching between two applications can be performed by two key presses.

9.3 Post Processing and Publishing

Most *post processing* aspects are performed automatically by the *TeleTeachingTool* or are automated, i.e. will by performed on request with little manual input. Whenever a *recorded lecture* is loaded which has no *indexing structure* (i.e. slide indices) and/or does not contain *slide preview images* for the *thumbnail overview*, the TTT will automatically analyze the recording and compute the indices and thumbnails (as described in Sections 7.3, 7.4 and 7.5). The computed *indexing structure* and the *thumbnails* can be stored permanently as part of the *electronic lecture* then.

Additionally, the TTT will automatically load, check for compability and integrate any *search base files* (if available). A *search base file* is an ASCII text or an XML file of the same name as the recorded lecture but with the file ending "`.txt`" or "`.xml`" (see Section 8.1.1 for supported *search base formats*). The TTT will test if the *number of text pages* that are stored in the search base matches the number of *slide indices* and, if loading an XML search base, if the XML structure matches the expected scheme. Provided it matches, the *search base* is integrated and can be stored permanently as part of the *electronic lecture* within to *desktop file* (file ending "`.ttt`").

Note that loading the *search base file* and the computation of *indices* and *thumbnails* are not only available during explicit *post processing*, but will rather be performed whenever necessary, for instance, whenever a student replays a *recorded lecture* that does not contain indices and/or a search base (see Section 9.1.1). However, it is preferable to distribute *lecture files* with integrated *indexing structures*, *thumbnails* and (optionally) *search base* for students' convenience.

9.3.1 Automated Post Processing

The *TeleTeachingTool* offers explicit *post processing* of recorded lectures, which is typically performed once for each recorded lecture before publishing it and typically takes no more than a few minutes. A recorded lecture is opened for post processing by selecting the menu entry "Post Processing→open.." (Figure 9.9) and choosing the appropriate lecture by use of a *file request dialog*.

Fig. 9.9. Opening a file for post processing

Then the lecture will be loaded and the *indexing structure* and the *thumbnails* are computed automatically. When the computation is finished, the *post processing dialog* will open. If loading an unmodified lecture, i.e. one that has not been post processed before, the *post processing dialog* will commonly look as given in Figure 9.10. The "*Info*" *tab* displays some metadata that reveals the *title*, the *date*, the *duration*, the *number of indices* and the *resolution* of the lecture. The *title* is read from the lecture file and typically is preset by the *lecture profile*, which was used for recording (Section 9.2). Without applying a *lecture profile*, commonly the *title* is set to the name of the recorded *VNC server*. If desired or necessary, the user can edit the *title*.

The "*Thumbnails*" *tab* shows the status of the thumbnails, i.e. whether the lecture contains thumbnails or, as presented in Figure 9.10, if the thumbnails have been

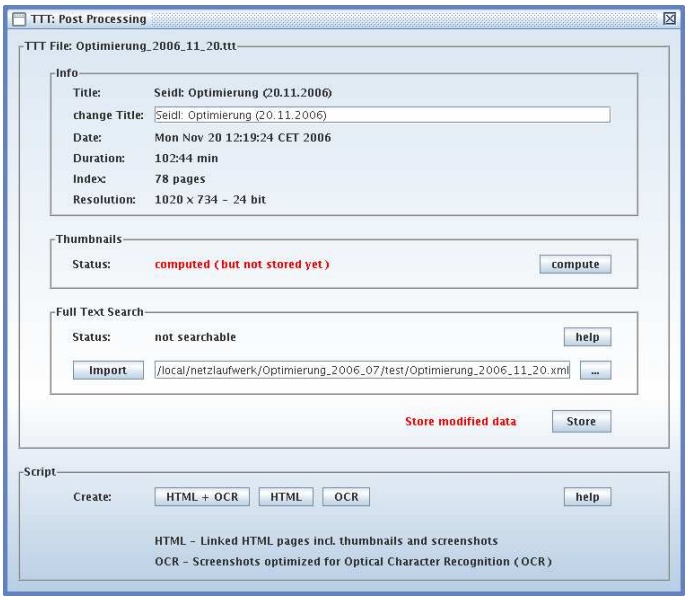


Fig. 9.10. Post Processing Dialog

computed but are not permanently stored to the lecture file, which is emphasized by the *red color*. In fact, any modified but not stored elements are colored in red (i.e. title, thumbnails and search base status). A (re)computation of the thumbnails can be initiated by pressing the *compute button*.

The last tab that relates to the *lecture file* is the "*Full Text Search*" *tab*. It reveals the current *status of the search base*, i.e. whether a search base is available and of what kind (ASCII text only or XML with coordinates, see Section 8.1.1 for formats). As mentioned above, matching *search bases files* are automatically loaded if available. Furthermore, the name of a *search base file* can be specified in the *text field* or via a *file request dialog* and than imported by pressing the *import button*. Any modified elements will be permanently stored within the *lecture file* by pressing the *store button*.

An additional feature of the TTT is the possibility of creating an *html script with annotations* for each recorded lecture. An *html script* consists of a thumbnail overview and linked slide screenshots (Section 7.5.1). The screenshots can be used as input for an external *optical character recognition* (OCR) application. However, the TTT also can store screenshots that are optimized for *optical character recognition* issues (Section 8.1.3). The user can select whether to compute and store an *html script*, *OCR input* or both by pressing the corresponding button in the "*Script*" *tab* (or storing nothing by not pressing any button). The output will be written to subdirectories in the same directory in which the lecture file is located. Moreover, the directories and any image and html files will automatically be named according to a *consistent naming scheme*, which will respect the *file name base* of the *lecture file*, i.e. same name but different file endings and possibly additional sequence numbers (referring to indices). Afterwards, the html script can be published (e.g. copied to a

web server) and the OCR input can be read by the OCR software and the resulting name can then be imported as the *search base* for the lecture as described above.

9.3.2 Post Processing Workflow

Although *recorded lectures* can be distributed and replayed without *post processing*, we suggest a few *post processing steps* in order to create *full-fledged electronic lectures* for the students' convenience. Quick *post processing* of recorded lectures is typically performed as follows.

1. **open lecture via the "Post Processing→open.." menu entry**
   (*indexing structure* and *thumbnails* will be computed automatically)
2. **press the "*HTML+OCR*" *button***
   (creates *html script* and *OCR input*)
3. **create *search base***:
   a) open *OCR application* and initiate reading and recognition of the *OCR input*
   b) store the output of the OCR application
      (using the same name as the *lecture file* but ending ".txt" or ".xml")
4. **press the "*Import*" *button***
   (reads a *search base file* with the preset naming; specify *file name* otherwise)
5. **press the "*Store*" *button***
   (creates a *full-fledged electronic lecture*)

Note that *step four* only *imports* the search base, i.e. reads and parses the *search base file* and interlinks the search base parts with *indices*. *Step five* is necessary in order to integrate the read *search base* as well as the computed *indexing structure* and *thumbnails* into the *desktop file*, so that these features will be available to students (without additional computation or reading a seperate *search base file* on startup).

Details concerning the supported *search base file formats* (ASCII text only or XML with coordinates) and *OCR applications* are discussed Section 8.1.1.

These five steps can typically be performed in less than five minutes for automated character recognition and in about 10–15 minutes if making some manual corrections during or after the recognition process. If omitting the creation of a *search base*, *post processing* can even be achieved in about one or two minutes, but note that a *full text search* can only be performed if a *search base* is available (either as a separate *search base file* or integrated within the *desktop file*)

9.3.3 Publishing

Afterwards the recorded lecture can be published. The TTT does not support the publishing of lectures as an integrated feature yet. Currently we use a script that packs the *audio*, *video* and *desktop file* as one *ZIP archive* and copies the *ZIP archive* to our web server. Furthermore, the script copies the *html script* and the *search base file* to the appropriate directories of our *web server*, so that they are respected by the *online full text search*. Besides packing and copying, the script checks the *consistent naming* and lists the information needed to create the appropriate entries at the download page. The script is listed in Appendix C.

In future, publishing should also be addressed by the TTT in order to provide a more complete environment. The packing can easily be integrated. Copying files to a web server requires some configuration possibilities in order to specify the server and the appropriate directories. We suggest adding these configuration parameters to the corresponding *lecture profile* or storing them as an additional *publishing profile*. The automated listing within the download page can be realized by reading the current page (i.e. the corresponding html file) and adding the new entry to a certain position within the file, for instance by storing a special mark hidden within an html comment. Afterwards the document can be copied back to the web server. Another possibility would be to implement a *dynamic download page* analogous to the dynamic *online full text search*. In fact, our web server already offers a dynamic listing of all available html scripts. As a download page should reveal additional information such as file sizes and duration or should link course related materials or the teacher's web page, the script must be extended correspondingly.

Instead of providing an own web server with proprietary services, an interface to support an *Learning Management System* such as *CLIX Campus* [CLIX, 2006] is worthwhile in order to provide a single address for students where they can find all course related information and materials, including the recorded lectures.

Furthermore, a course can be distributed as a self-contained DVD, which does not only contain the edited lecture files, but also additional course material, the automatically generated html scripts and the *TeleTeachingTool* software. Additionally, these DVDs are bootable and contain a self-starting linux operating system (KNOPPIX [Knopper, 2006]), which commonly supports (almost) any hardware. Any necessary software including the *TeleTeachingTool* is preinstalled and preconfigured on the DVD's operating system and on system startup a course overview will be presented within a browser with direct links to replay any of the recorded lectures.

In order to create such a course DVD rather quickly, we have produced a *DVD template*, which contains the operating system including any software and provides a special folder for the course related data. In order to produce a self-contained lecture DVD, it is sufficient to use the *DVD template* and copy the recorded lectures and any additional materials to the special course folder. Afterwards an *ISO image file* of the DVD is created, which can be burned to DVDs.

9.3.4 TTT to Flash Converter

Distributing electronic lectures in a *standard file or streaming format* is preferable as no software installation is necessary [Lauer and Ottmann, 2002]. However, most standard video formats and compression codecs are designed to compress "*real world video content*" and are not suitable in order to compress "*computer-based content*" such as slide presentations [Hogrefe et al., 2003, Lauer and Ottmann, 2002] and furthermore it is very difficult or even impossible to integrate enhanced *navigational* and *retrieval features* to *standard video formats*.

One wide-spread video and animation format that is especially designed to compress computer-based content is *Adobe Flash* (formerly *Macromedia Flash*) [Flash, 2006]. In a diploma thesis one of our students developed and implemented a prototype of a converter that transfers recorded lectures from the *TTT format* to *Flash* [Nopoudem, 2006]. The converted lecture will look like the example shown in Figure 9.11.

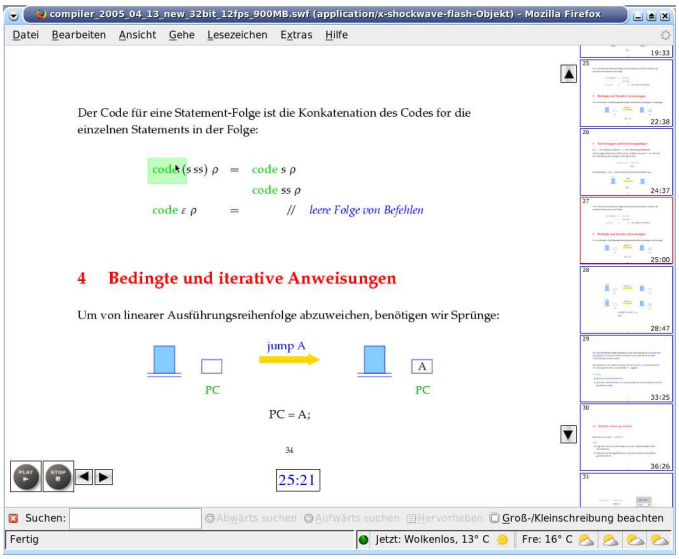


Fig. 9.11. TTT Flash Movie

The converter implementation extends the infrastructure of the *TeleTeachingTool* in a way that each *message object* is extended by an additional writeToFlash() method and therefore it should be an relative easy task to integrate the converter into the TTT. In fact, converting is performed in almost the same way as creating the *html scripts* and thus the *Flash converter functionality* might be added by placing an additional *create flash button* to the *post processing dialog* (Figure 9.10).

The current implementation of the *Flash converter* for TTT recordings supports the basic playback functions *start*, *stop* and *pause* and furthermore *slide-based navigation* via a *thumbnail overview*. A slider for *timeline navigation* and *visible scrolling* is currently not available within the resulting *Flash movies*. Furthermore, *full text*

search is not supported. Nevertheless, by implementing a conversion of TTT/RFB messages and implementing the basic navigation functions, the diploma thesis proved that *TTT lectures* can be transformed to a wide-spread (quasi) standard format. Extending the converter to create full-featured electronic recording is suggested as future work.

9.3.5 TTT Editor

Although the *lightweight lecture recording approach* discourages extensive manual post processing and editing, it is sometimes useful to have at least some rudimentary editing features. For instance, cutting the beginning or end of recordings as sometimes teachers will start the recording too early or stop them several minutes after the actual end of the lecture. Furthermore, lectures may be cut and concatenated so that the resulting pieces will relate to certain topics instead of reflecting a 90 minute lecture period.

An exchange student from the School of Computer Science, The Queen's University of Belfast, has developed and implemented an *editor for TTT recordings* as a master thesis [Bankhead, 2005]. Due to the requirements of the university, the *TTT Editor* was implemented as a stand alone application. In contrast to the TTT implementation, which processes lectures as *message streams* plus additional *structuring data* such as the timestamps that correspond to slide indices, the editor implementation uses indices as the major data structure and each *index object* contains a *set of messages*.

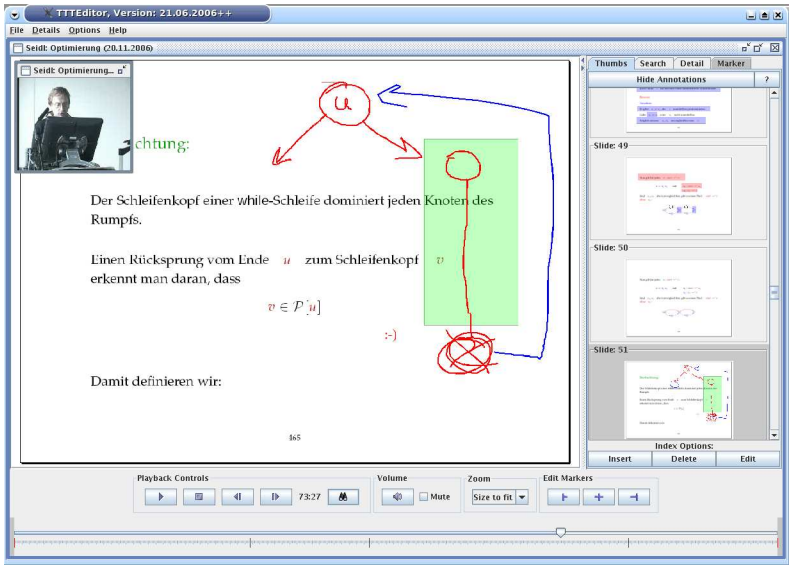


Fig. 9.12. TTT Editor

A screenshot of the *TTT Editor* is shown in Figure 9.12. The *TTT Editor* offers a *replay engine* that supports most replay features of the *TeleTeachingTool* (see Section 9.1.1) except *visible scrolling*. The *main window* displays the recorded lecture and the *control bar* at the bottom offers both the control elements (such as the *timeline slider* and other *replay controls*) and also additional controls to set *markers* for editing purposes.

The area on the right hand side of the GUI displays one of four *function tabs*, which can be selected by the appropriate tabs at the top of the area. Besides the two tabs that are known from the TTT replay GUI, the *thumbnail overview* (Figure 9.12 to the right) and a representation of *search results* (also via thumbnails), the editor offers two more tabs, the "*Detail*" and the "*Marker*" *tabs* (Figure 9.13).

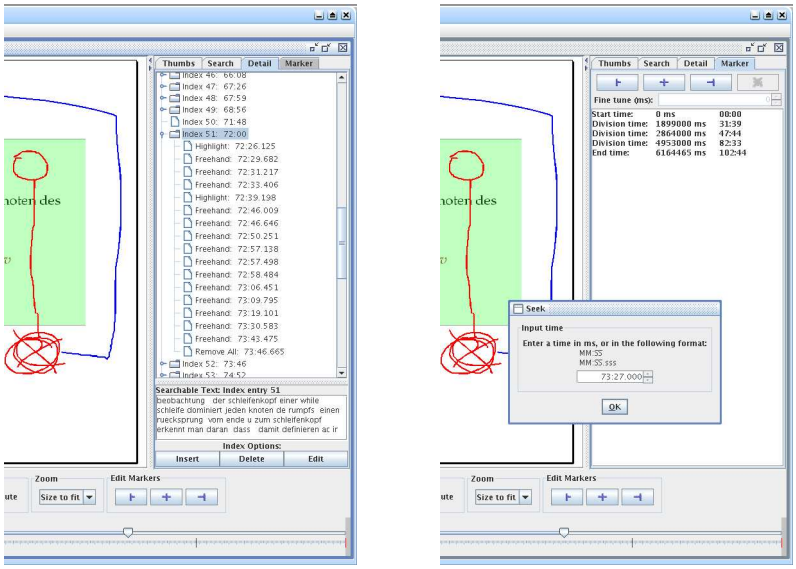


Fig. 9.13. TTT Editor *detail tab* (left) and *marker tab* (right)

The "*Detail*" *tab* represents the *list of indices* and for each index a list of the associated *annotation events*. Individual *indices* can be deleted or modified. The *index timestamp* can be adjusted, i.e. set to another position, the *index title* can be set and the *search base* can be edited manually (currently only ASCII search bases are supported; see Section 8.1.1 on page 162). Note that the TTT currently does not display *index titles*, which might be used to create a *table of contents*. Furthermore, new indices can be generated by choosing the appropriate timestamp and pressing the "*Insert*" *button* at the bottom of the tab. Additionally, the listed *annotations* can be deleted and, in future, possibly the editor will also support editing and adding annotations. In this regard, the *annotating tools* of the TTT can simply be added to the editor's GUI. However, one has to consider whether additional annotating should be done *while replaying* the lecture or during *pause mode*. Annotations can be placed exactly timed in *pause mode*, but since all annotations will refer to the *same timestamp*, placing many annotations in *pause mode* will result in a simultane-

ous appearance of all annotations during later replay, which might irritate students. [Lienhard and Zupancic, 2003] suggest limiting annotating features during replay to setting *static textual annotations* ("*PostIts*") instead of placing dynamic annotations.

The *TTT Editor* supports *markers*, which can be set in order to specify certain points for splitting lectures or cutting parts out of it or to mark positions, which could be used to place additional indices. The "*Marker*" *tab*, which is shown on the right hand side of Figure 9.13, lists all currently set *markers* including the *start* and *end marker*. Markers are also visualized by small vertical marks in the timeline at the bottom of the GUI. *Markers* are placed by setting the current playback time to the desired position and pressing one of the three *marker buttons*, which resets the *start* or *end marker* or places a new intermediate one, respectively. The buttons are avaliable within the "*Marker*" *tab* (at the top), but are also available in the *control bar* at the bottom, which is useful to place markers whenever one of the other tabs is displayed. Setting a certain time can be done by *timeline* and *slide-based navigation*, by pressing the *pause button* during replay or by explicitly setting the desired time in minutes, seconds and milliseconds by use of a special *time seek dialog*, which is also shown in Figure 9.13 (in the middle of the right screenshot).

Fig. 9.14. TTT Editor: File Menu

After manipulating indices and setting appropriate markers, the recorded lecture can be saved. If the *start* and *end markers* have been adjusted, i.e. if timestamps other than zero and the duration of the lecture have been assigned, the lecture will be correspondingly cut at the beginning and/or the end. Furthermore, a lecture can be split into several pieces by placing appropriate markers and selecting the "Save & sub divide" entry in the "File" menu, which is shown in Figure 9.14. The beginning and end of the lecture will be cut and each sub division will result in an own *lecture file*. Concatenating two or more files is done by selecting the "File→Concatenate" menu entry and choosing the appropriate files in the *concatenate dialog* as shown in Figure 9.15. Note that any splitting, cutting and concatenating operations will not only affect the *desktop recording* but the *audio* and *video files* as well.

Normally we do not edit our recorded lectures but rather use the editor only to cut a lecture whenever the beginning or end of the recording significantly differs from that of the lecture or to combine two parts of a lecture if the lecture was interrupted. Furthermore, we edited the course "Compilerbau" of Prof. Dr. Helmut Seidl, which was recorded during the summer semester 2005, in order to demonstrate how to split

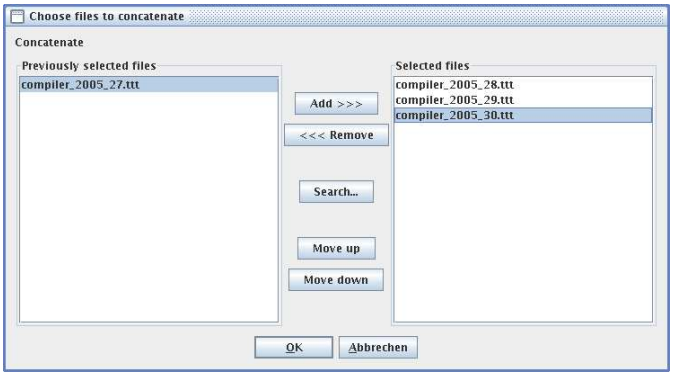


Fig. 9.15. TTT Editor: Concatenating Dialog

a lecture into meaningful chunks that relate to the course structure, i.e. chapters and topics, rather than to the presentation schedule, i.e. two 90 minute lectures a week. The edited course is available to students in the form of a self-contained DVD (as described in Section 9.3.3).

9.4 Transmitting Live Lectures

The *TeleTeachingTool* provides *synchronous live transmission* of VNC sessions as suggested and described in Chapter 4. To achive this, a *TTT Server* transforms the *RFB message stream*, which is received from a *VNC Server*, to *TTT messages* which are then sent to the students' *TTT Clients*. The *TTT Server* parses (to process distinct messages; Section 4.2.6), splits (to respect packet limits; Sections 4.2.4 and 4.2.5) and packetizes (as UDP datagrams; Section 4.2.7) the incoming messages. Additionally, the *TTT Server* will sent an *audio* and (optionally) a *video stream*.

Starting a *TTT Server* is initiated by selecting the "Server→Start TTT-Server" menu entry as shown in Figure 9.16. Note that this and any further screenshots in this section show an older version of the *TeleTeachingTool*, because the server functionality is currently not fully implemented in the redesigned version of the TTT (but will be in future).

Fig. 9.16. TTT Server

Selecting the menu entry will open the *server options dialog*, which is displayed in Figure 9.17. The *simple options dialog* is intended to enable a *quick start* of a *TTT Server* and hides additional parameters, which are automatically set to meaningful default values. In order to run a *TTT Server* it is sufficient to specify the *host name* and *port* of the *VNC Server* that delivers the desktop to be transmitted (and recorded), and furthermore to enable transmitting and optionally recording by selecting the appropriate *check boxes* at the bottom of the *simple options tab*. The default *audio* and *video devices* will be used and the server will be accessable by the students under the *host name* (of the machine) and a preassigned *port* (commonly port 33229) as listed at the top of the dialog box. The *TTT Server* is started by the "*ok*" *button*.

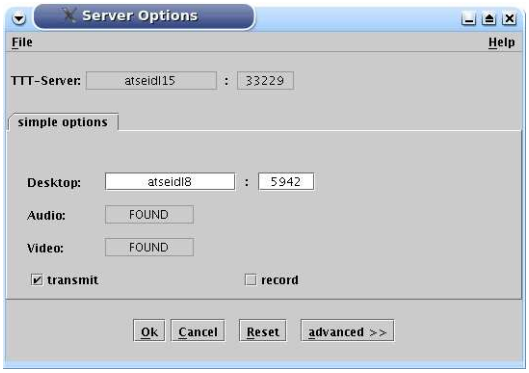


Fig. 9.17. TTT Server: simple options dialog

The *advanced options dialog* provides tweaking of the preassigned default parameters and is accessed from the *simple options dialog* by pressing the "*advanced*>>" *button*. The *advanced options dialog* consists of three different tabs which relate to the *source*, *transmission* and *recording parameters*, respectively.

In the *source tab* (Figure 9.18, left) the input sources are assigned, which are the desktop of a *VNC Server* (analogous to the simple options) and the *audio* and *video devices*, which can be selected from the list of all determined devices. Additionally the color depth can be assigned, which will be used for transmission and recording the desktop.

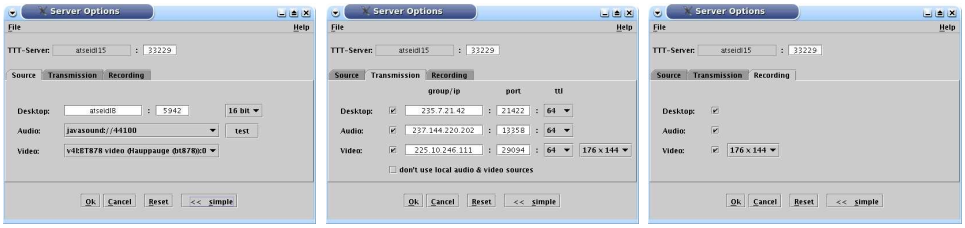


Fig. 9.18. TTT Server: advanced options dialog

The screenshot in the middle of Figure 9.18 shows the *transmission tab* with preassigned randomly generated *multicast groups* and *ports* for desktop, audio and video transmission. Each value can be manually edited. Furthermore, this tab enables the size of the video that shows the teacher to be set and the *time-to-live*[1] (ttl) in order to specify the range of the transmission. Furthermore, the *port* of the *TTT Server* can be manually adjusted at the top of the dialog. Note that the *host name* cannot be edited, because it is the *host name* of the machine on which the *TTT Server* is executed. Changing the name rather requires a modification of the network properties on a lower level (network and operating system).

Finally, the *recording tab* (right hand side of the figure) offers the possibility to enable or disable the recording of the three data streams (desktop, audio and video) as well as to adjust the size for the recorded video (of the teacher).

The user can switch between the *simple* and *advanced options dialog* as required. Any value will be preserved until edited, even after terminating and restarting the *TeleTeachingTool*. Furthermore, any options can be explicitly saved to a file and loaded again by selecting the appropriate function within the 'File" menu of the dialog. In future, the server parameters should be integrated into the *lecture profiling concept* (Section 9.2).

[1] The *time-to-live* (ttl) specifies the maximum number of transmissions for each network packet (datagram) while traversing the network, i.e. the ttl of a datagram is reduced by every host on the route to its destination and if it reaches zero before the datagram arrives at its destination, then the datagram is discarded. In the case of *multicast* delivery, the ttl defines the range of the transmission from local networks (ttl of 0–2) to continental or world wide delivery

9.5 Usage Scenarios

We are using (variations of) the *TeleTeachingTool* since the winter semester 2001/02 and have recorded about 400 lectures over recent years at the Universität Trier and the Technische Universität München. We have used the TTT in different set-ups which will be described here.

A absolutely transparent recording environment was used while recording the Lecture "*Informatik III*" by Prof. Dr. Johann Schlichter. He use a tablet PC to present his slides by use of a standard web browser. Slides were annotated by use of an electronic pen and an Java applet. During the lecture the desktop was exported to a *TTT Server* running on another machine, which was also connected to the audio and video equipment of the lecture hall. The tablet PC was also connected to the beamer in order to present the slides to the audience. This scenario enabled Prof. Schlichter to give his lecture in the same way as would have been the case without recording. Since we achieved the highest degree of transparency and flexibility.

A very similar set-up is used at the Universität Trier in order to record the lectures given by Prof. Dr. Hans-Jürgen Bucher. Instead of a tablet PC, Prof. Bucher uses an ordinary labtop. Again the PC of the teacher was accessed remotely by a *TTT Server* in order to be recorded. Since the used laptop does not provide appropriate input devices for annotating purposes, we also placed an electronic tablet into the lecture hall, which presented the same desktop but within a *TTT Teacher Client*. As this client provides the built-in annotation tools of the TTT, Prof. Bucher can annotate his slides. However, he preferred to control is desktop directly instead of remotely, i.e. generally he uses his own familiar laptop but whenever he required an annotation feature, he used the electronic tablet.

In order to record the lecture of Prof. Dr. Helmut Seidl we use a rather different solution. Instead of using a laptop as source of the desktop recording, we rather use an office PC, which is remotely controlled. In fact, this office PC is placed in Prof.Seidl's office and is the machine he uses during his working hours. Other than the previously described scenarios, this machine does not export the standard desktop, but a *VNC Server* is started in the background and exports an own virtual VNC desktop, which is accessed by a TTT Server which is executed on a machine placed in the lecture hall. Again we provide an electronic tablet and the TTT's annotation features for annotating the slides and other presented content. The benefit of this set-up is that the teacher need not to carry his laptop to the lecture hall. Furthermore, he can use any application which are available on his office PC and he can prepare a lecture, i.e. he can select the slide he intents to present first. In fact, the VNC server is never terminated in our set-up and therefore the presentation can be initiated the day before the lecture is given. Consider an lecture which takes place early in the morning, but everything is already prepared the day before. The lecture can be started be just connecting to the office PC.

This three scenarios show the different possibilities our lecturing environment offers. Which of these scenarios should be used depends on the available hardware and the preferences of the teacher.

9.6 Java Media Framework (JMF)

The *audio stream* (verbal narration) and *video stream* (showing the teacher) are not encoded within the *message stream/desktop stream* (encoding the desktop presentation), but are rather delivered and recorded by use of suitable *standard media formats*, such as the *Real-time Transport Protocol* (RTP), which defines a standardized packet format for delivering audio and video over the Internet [Schulzrinne et al., 2003]. During the initialization phase (i.e. whenever a client connects to the *TTT Server*) the *IP addresses* and *ports* are specified, describing where to receive *additional media streams*. For *asynchronous replay* (of previously recorded lectures) the *additional media streams* are determined by appropriate filenames (same as the desktop file but different endings). Since this thesis focuses on the recording, transmission and automated post-processing of computer-based presentations, i.e. the presented virtual desktop, and not on capturing, encoding and replaying of audio and video streams, we prefer to use an existing environment instead of implementing the appropriate functionalities.

The *Java Media Framework* (JMF) [JMF, 2006] is an optional package for the *Java platform*, which can capture, playback, stream and transcode multiple (standard) media formats, i.e audio, video and other time-based media. Hence, JMF is a full-featured framework that can be integrated into Java applications by use of the *Java Media Framework API* (*application programming interface*). Nevertheless, the framework might be replaced by any other appropriate package that is accessible from within Java applications or, as the implementation of the ideas that are described in this thesis does not necessarily need to be implemented in Java, any other language as well.

In order to provide a full-featured recording environment, audio and video streams must be captured, recorded, transmitted and replayed in a synchronously and asynchronously manner. The capturing of media streams is not directly integrated in the JMF core package. Instead JMF accesses certain (maybe platform dependent) capturing packages such as *JavaSound*, *video4linux* (v4l) or *video for windows* (VFW), which commonly support typically used hardware. Platform dependent libraries can only be accessed by the JMF in the appropriate *performance pack versions*, which are available for several operating systems. Note that replaying of almost any supported data formats and codecs can be handled by the platform independent *cross platform version*.

Regarding the formats and codecs, we prefer standard formats (Criterion: C9d) and, since we do not intend to implement any codecs (which can be integrated into the JMF), we will use only those codecs and formats that are supported by and available for the JMF. From the available video formats[2] only the *h.263* codec is capable of producing acceptable data transfer rates and file sizes (about 70–90 Mbyte for a video of 90 minutes at a resulotion of 176×144 pixels). All other supported formats produce files of several hundred Mbytes. Unfortunately, more "up-to-date" codecs like *MPEG-4/DivX* are currently not supported by the JMF.

[2] The complete set of formats is listed under http://java.sun.com/products/java-media/jmf/2.1.1/formats.html

The variety of available audio formats and codecs is higher and most codecs achieve suitable compression rates to produce acceptable file sizes. As with video recording and transmission, the bandwidth consumption and file sizes for audio streams should be as low as possible. However, a reduced sound quality cannot be tolerated (Criterion: C1 Verbal Narration). Note that "CD quality" and "stereo sound" is not necessary to capture the verbal narration of a teacher, but a distorted speech with dropouts or nasal reproduction must be avoided. By analyzing the produced data rate (and thus the file sizes and bandwidth consumption) and the sound quality of the recorded speech, we have chosen *MPEG-1 Audio Layer 3*, commonly known as *mp3*, at 22,050 Hz, 16-bit, Mono as a suitable codec. Lowering the rate or recording to 8-bit noticeably reduces the sound quality and thus is not acceptable. Recording the teacher's speech in stereo is not necessary. As JMF does not support *mp3* for transmission purposes, we must use another codec for synchronous electronic lectures. We have found μlaw at 8000 Hz, 8-bit, Mono to be suitable. The other available codecs result either in a lower quality or higher bandwidth consumption.

Asynchronous electronic lectures are recorded to files. The supported video file formats are *QuickTime* (mov) and *AVI* (*audio video interleaved*). Due to a bug in the then available version of JMF, AVI replay was only possibly up to 35 minutes and 47 seconds (caused by an *integer overflow*). Hence, we used *QuickTime*. As the JMF cannot record *mp3 audio* within a *QuickTime video* we record one file for each stream.

For *synchronous transmission* the JMF supports the *real-time transport protocol* (RTP), which defines a standardized packet format for delivering audio and video over the Internet [Schulzrinne et al., 2003] and is commonly used by many video conferencing systems. If compared to other streaming media formats, the benefit of rtp is that no noticeable buffering is necessary and therefore rtp transmissions achieve rather short round-trip times and therefore are suitable for a two-sided communication.

9.6.1 TTT/JMF Interface

The interface between the JMF and the TTT can be limited to a few functions in order to synchronize the desktop handling with the media streams, which are processed by the JMF. In particular the following functionality is required for the server and recording component of the TTT:

- accessing audio/video capture devices,
- initializing, starting and terminating the recording of audio/video streams and
- initializing, starting and terminating the transmission of audio/video streams.

The student component requires:

- receiving and replaying of transmitted audio/video stream,

- reading and replaying (including start, stop and pause) of recorded streams,
- requesting the current replay time (to synchronize desktop replay) and
- setting the replay time (to enable random access).

JMF provides the class `javax.media.Manager` as the *access point* for obtaining system dependent resources, which is used to create instances of (implementing classes of):

`javax.media.protocol.DataSource`	the origin of a media stream;
`javax.media.Player`	replays a media stream;
`javax.media.Processor`	transforms a media stream (e.g. encoding and decoding) and
`javax.media.DataSink`	sepcifies the output for a media stream.

The TTT recording and transmitting components create a `DataSource` for the selected *audio and video capture devices*. In order to support different outputs (e.g. for recording and transmission), the media stream of the `DataSource` is multiplexed and handed to appropriate `Processors`. Each `Processor` encodes the media stream according to the given format specifications (e.g. μlaw/rtp, 8000 Hz, 8-bit, Mono for our audio transmission) and delivers the encoded stream to an appropriate `DataSink`, which is either related to an *output file* or an *output socket*.

Replaying is done by creating an appropriate `DataSource`, which is either associated with a recorded file or an *input socket* (which is connected to the *output socket* of the *TTT server*). A `Player` is created for such a `DataSource`. This `Player` offers certain controlling methods and, in the case of a video stream, a *visual component*, which can be displayed by the TTT. A `DataSink` is not required during replay.

The synchronization between the *audio* and the *video stream* is accomplished by the JMF. The *desktop stream* is synchronized to the *audio stream* by comparing the *timestamp* of the next *message* (of the *desktop stream*) against the *current replay time* of the *audio stream*. Typically the *timestamp* will be larger and the *message* will be delayed accordingly. If the audio replay has already passed the *timestamp*, the *message* will be replayed instantaneously. *Random access* is performed by setting the *replay time* of the *audio* and *video streams* according to the specified *timestamp*, computing the *current state* of the *desktop framebuffer* (as described in Section 5.3) and setting the first message that exceeds the given *timestamp* as *next message* to be processed. No synchronization is required for *synchronous transmissions*. In fact the *audio*, *video* and *desktop streams* are replayed as they arrive.

9.6.2 Limitations and Problems

Although the *Java Media Framework* provides a meaningful infrastructure in order to capture, playback and stream standard audio and video formats, there are some drawbacks. The supported codecs, especially the video codecs, are rather outdated. Support of newer, more advanced and better compressing formats such as

ogg, *h.323*, *h.264* or *MPEG-4*, is missing. Furthermore, recording an audio stream is rather problematic regarding the synchronization. The recorded audio streams are sometimes too short. The reason is that small pauses may occur during which no audio is recorded and which are mostly generated by high system loads on systems that use onboard sound devices. Although these pauses are no longer than a few milliseconds, they add up over a lecture of about 90 minutes to several seconds or, in the worst case, even minutes. During replay the synchronization between the audio and the desktop stream (video as well) will be fine at the beginning but will get worse towards the end. Hence, in order to produce synchronous recordings, hardware that will not be affected by this problem must be used, which can be troublesome due to the high number of possible hardware combinations. Typically stand alone sound adapters are better suited than today's commonly used onboard solutions.

In order to provide a workaround, the *TeleTeachingTool* checks during replay whether the data streams are of equal lengths. If not, the desktop replay rate will be slightly increased so that the replay duration matches the duration of the audio stream. However, the pauses are not necessarily equally distributed throughout the duration and therefore this is only a slight improvement but at least limits the asynchronism to a few seconds, which is an improvement while replaying progresses towards the end of the electronic lecture. Nevertheless, the replay of the video stream will still be asynchronous. The JMF API offers a method to set the replay rate of media streams, but unfortunately only a rate of 1.0 is implemented for the given formats. Another solution was suggested by Prof. Dr. Wolfgang Slany (Technische Universität Graz). He uses the *TeleTeachingTool* for recording his lectures, but additionally records the audio stream with another audio recording tool. Afterwards he edits (cuts) the beginning of the audio stream to provide synchronous replay. However, manual post-processing is necessary.

Since the developer of the *Java Media Framework*, which is *Sun Microsystems, Inc.*, does not focus much effort on improving this framework, for instance by supporting newer, more meaningful codecs and formats, we might watch out for an alternative. In fact the latest update on the JMF web pages is dated "November 2004". *QuickTime for Java* [QTJava, 2006] offers access to the *QuickTime API* for Java applications, but does not support the linux operating system. The open source project *Freedom for Media in Java* (FMJ) [FMJ, 2006] aims to reimplement and improve JMF while preserving compatibility with the JMF API, but is still in progress.

9.6.3 Summary

In summary, the *Java Media Framework* offers (almost) all of the functionality that is required to build up an environment for recording and replaying asynchronous electronic lectures as well as to transmit and receive synchronous ones and therefore is a suitable basis to implement the *TeleTeachingTool* in order to approve the ideas and concepts, which we suggested in this theses. Currently, the *TeleTeachingTool* uses JMF to record *mp3 audio* and *h.263/QuickTime video files* and transmits *μlaw/rtp audio* and *h.263/rtp video streams*. As the available encodings are rather limited and somehow "outdated" and furthermore the current implementation of the *Java Media Framework* has some drawbacks, another possibility to integrate audio

and video streams should be preferred to provide a more stable and satisfying lecture recording environment in the future.

9.7 File Format Specification

The *desktop recording* (file ending ".ttt") produced by the *TeleTeachingTool* consists of three parts: a ***header***, which specifies certain *parameters* (analogous to the *initialization* provided by a *VNC Server*), optional ***extensions*** that may contain *indexing structures*, *thumbnails* and *search bases*, and the main ***body*** that stores the *logged messages*.

Since the TTT is derived from the VNC, the *TTT format* shows many analogies to the *RFB protocol specification* [Richardson, 2005]. Unlike the *RFB protocol*, the *TTT protocol* enables reading (and skipping) of (potentially unknown) *message types* without parsing them and thus enables *faster processing* of recorded lectures.

9.7.1 Header

The *header* is derived from the *initialization phase* of the RFB protocol [Richardson, 2005] and analogous starts with the *protocol version*:

No. of bytes	Type	Description
12	`byte` array	*protocol version* [`TTT 001.001\n`]

Unlike the *RFB protocol*, the TTT encodes/compresses **any subsequent data** (including *extensions* and *body*) that follows the *protocol version* by use of a *zlib deflate stream* in order to reduce the file size.

The *initialization* is the same as the *ServerIntialization* of the RFB protocol (for details and the meaning of the parameters see *RFB protocol specification* [Richardson, 2005]):

No. of bytes	Type	Description
2	`short`	*framebuffer-width*
2	`short`	*framebuffer-height*
1	`byte`	*bits-per-pixel*
1	`byte`	*color-depth*
1	`byte`	*big-endian-flag*
1	`byte`	*true-color-flag*
2	`short`	*red-max*
2	`short`	*green-max*
2	`short`	*blue-max*
1	`byte`	*red-shift*
1	`byte`	*green-shift*
1	`byte`	*blue-shift*
3		*padding*
4	`int`	*name-length*
name-length	`byte` array	*name-string*

Now a possibly empty list of *extensions* follows, where an *extension length* of zero bytes indicates the end of the list:

No. of bytes	Type	Description
4	`int`	*length-of-extension*
length-of-extension	`byte` array	*extension*

The *header* ends with the time/date when the recording was started (in milliseconds since midnight, January 1, 1970 UTC):

No. of bytes	Type	Description
8	`long`	*startime*

9.7.2 Extensions

The *TTT protocol* provides an *extensibility* by offering the *concept of extensions*, which enable the integration of additional data, such as *indexing structures*, without modifying the *protocol header* or the *message type specifications*.

Currently the TTT suports the following two extension types:

Type 1: ***index table extension***
Type 2: ***searchbase extension***

Each *extension* must start with a `byte` specifying the *extension-type* and is followed by *type specific data*. Since each *extension* is preceded by its *size* (see *header*), any unknown extension can be skipped (ignored).

Index Table Extension

The *index table extension* stores the *indexing structure* of a recording and optionally contains *thumbnails* (preview images) and *ASCII search bases* for the indices.

The *index table extension* starts with:

No. of bytes	Type	Description
1	`byte`	*extension-type* [`1`]
2	`short`	*number-of-indices*

followed by *number-of-indices* "*index entries*":

No. of bytes	Type	Description
4	`int`	*timestamp* (milliseconds since beginning)
1	`byte`	*title-length*
title-length	`byte` array	*title-string*
4	`int`	*searchbase-length* (no searchbase if zero)
searchbase-length	`byte` array	*searchbase-string* (ASCII)
4	`int`	*thumbnail-size* (no thumbnail if zero)
thumbnail-size	`byte` array	*thumbnail-image* (png image)

Search Base Table with Coordinates Extension

The *search base table extension* stores a *search base* with *coordinates* (in order to emphasize search results) for each *index entry* of a recording.

The *search base table extension* starts with:

No. of bytes	Type	Description
1	`byte`	*extension-type* [2]
2	`short`	*number-of-indices*
8	`double`	*coordinate-ratio*

Note that the number of indices of the *search base table extension* must match the number of indices listed by the *index table extensions*. The *coordinate-ratio* is required to translate the given coordinates to *pixel coordinates* (by multiplying the *coordinate* by the *ratio*), because *ScanSoft's Omnipage*, which we use to generate the *search bases* (see Section 8.1.1), produces coordinates that relate to a *dpi measurement* (*dots per inch*) instead of *pixels*.

The *extension header* is followed by *number-of-indices* "*search base entries*":

No. of bytes	Type	Description
2	`short`	*number-of-words*

followed by *number-of-words* "*word entries*":

No. of bytes	Type	Description
2	`short`	*word-length*
word-length	`byte` array	*word-string* (ASCII)
2	`short`	*x-position*
2	`short`	*y-position*
2	`short`	*width*
2	`short`	*height*

9.7.3 Body

The *body* of a *TTT file* contains a *sequence of messages*, which is terminated by the *end of file* ("`EOF`"). Each *message* starts as follows:

No. of bytes	Type	Description
4	`int`	*message-length*
1	`byte`	*encoding* (message type)

where the *encoding* is one of the following:

value	Description
0	*EncodingRaw*
1	*EncodingCopyRect*
5	*EncodingHextile*
17	*EncodingTTTCursorPosition*
18	*EncodingTTTXCursor*
19	*EncodingTTTRichCursor*
20	*AnnotationRectangle*
21	*AnnotationLine*
22	*AnnotationFreehand*
23	*AnnotationHighlight*
24	*AnnotationDelete*
25	*AnnotationDeleteAll*
33	*EncodingWhiteboard*
64	*EncodingFlagUpdate*
128	*EncodingFlagTimestamp*

The *EncodingFlagUpdate* and the *EncodingFlagTimestamp* can be combined ("`OR`" operator) with any other encoding. The *EncodingFlagUpdate* is only relevant for *transmitted* lectures and reveals that a message contains *potentially outdated updates*, which will only be displayed by clients with an uninitialized framebuffer as suggested in Section 4.2.12 (reusing non-incremental keyframe stripes).

If the *EncodingFlagTimestamp* is set, the *encoding* is succeeded by the *timestamp* of that message (otherwise the *timestamp* will be carried over from the previous message):

No. of bytes	Type	Description
4	`int`	*timestamp* (if *TimestampFlag* is set)

Framebuffer Update Encodings

In order to provide *direct access* to the *rectangle headers*, we store *individual rectangles* instead of *messages* that contain *sequences of rectangles* (as suggested in Section 5.1.2). Furthermore, we have placed the *encoding field* in front of the coordinates and dimensions (unlike the RFB protocol). The *RFB rectangle encodings* are used as *TTT encodings* unless they do not fit in our 1 byte *encoding-field* (since the RFB protocol uses 4 bytes), which is the case for the *cursor pseudo encodings*: (*EncodingXCursor* [`-240`] and *EncodingRichCursor* [`-239`]).

Any *framebuffer update rectangle* or *cursor encoding* starts as follows:

No. of bytes	Type	Description
2	`short`	*x-position*
2	`short`	*y-position*
2	`short`	*width*
2	`short`	*height*

and will be succeeded by *encoding related data* according to the corresponding *RFB encodings* as specified in the *RFB protocol specification* [Richardson, 2005].

Note that any *encoding related data* can be read without parsing due to the size tag that precedes each message. The length of the *encoding related data* is the *length of the message* minus the *length of the header*. Reading rectangles without parsing their content is meaningful, because many rectangle processing algorithms (for instance the slide detection) can be performed by use of the rectangle headers only. Furthermore, the *size tags* enable skipping of messages of unknown encoding type.

EncodingTTTCursorPosition

In order to specify the position of the cursor (shape), the *width* and *height fields* are not necessary:

No. of bytes	Type	Description
2	`short`	*x-position*
2	`short`	*y-position*

AnnotationRectangle, AnnotationLine and AnnotationHighlight

These message types only differ in their graphical representation (and obviously the *encoding field*). The *format* of all three types is:

No. of bytes	Type	Description
1	`byte`	*color* (of a predefined color table)
2	`short`	*start-x-position*
2	`short`	*start-y-position*
2	`short`	*end-x-position*
2	`short`	*end-y-position*

where either a *line* or a *rectangle*, respectively, will be drawn (in the given *color*) from the *start position* to the *end position*. A *line* is a straight line between the two points and a *rectangle* contains the two specified points as corners and will be drawn parallel to the axes. In case of a *AnnotationHighlight*, the rectangle will be *filled* in a translucent color, so that the underlying desktop framebuffer is emphasized but still visible.

AnnotationFreehand

A *freehand annotation* consists of a sequence of points, which are connected by lines, and starts as follows:

No. of bytes	Type	Description
1	`byte`	*color* (of a predefined color table)
2	`short`	*number-of-points*

followed by *number-of-points* "*point entries*", where each *point* consists of:

No. of bytes	Type	Description
2	`short`	*x-position*
2	`short`	*y-position*

AnnotationDelete

No. of bytes	Type	Description
2	`short`	*x-position*
2	`short`	*y-position*

Note that not the single point will be deleted but rather any annotations that contain the point of deletion, i.e. that affect/paint the pixel at the corresponding coordinate.

AnnotationDeleteAll

This message initiates the deletion of any currently shown annotations and needs no further message specific data.

EncodingWhiteboard

Whiteboard messages are used to switch between *blank page(s)*, the whiteboard, and the desktop and consists of the following field only:

No. of bytes	Type	Description
1	`byte`	*page-number*

where `0` relates to the *desktop* (*whiteboard disabled*) and larger values correspond to *whiteboard pages* (*whiteboard enabled*).

10

Conclusion

The *lightweight* approach of *recording* and *transmitting live lectures* enables the *cost-efficient production* of multimedia-based learning materials for distance education as well as for local students in order to study independent of time and place.

In general, there are two conflicting recording approaches: *recording symbolic representation* and *pixel-based screen recording*. *Symbolically represented documents* typically offer *structured data* and thus enhanced *navigational features*, editing and searchable content, but restrict the teaching and recording process because only certain document formats and presentation applications can be supported. In contrast, *screen recording* offers a very *flexible technique* for the grabbing/recording process as it allows *virtually any material* to be captured (on a pixel basis) displayed during a presentation, but commonly supports only *sequential replay* and *timeline navigation*, and *lacks any retrievability*.

The first *main goal* of this thesis was the design and development of a *flexible, easy-to-use recording and transmission environment*, which does not restrict teachers in their content production and presentation process, but rather can be seamlessly integrated into an existing teaching environment in a *transparent* manner, so that the teacher is not aware of the recording process.

Virtual Network Computing (VNC) was chosen as a suitable basis to create a *flexible environment*. By remotely accessing/grabbing the presented desktop via VNC, lecture recording and transmission can be *seamlessly integrated* into almost any existing learning environment as long as the teacher uses *computer-based presentations*. Moreover, this approach does not restrict the teachers in their choice of presented documents and applications.

In order to support a *scalable desktop transmission*, i.e. to support a large number of simultaneously connected students, we have additionally modified the *VNC infrastructure* and its *Remote Framebuffer* (RFB) protocol by reducing the unnecessarily huge number of different pixel formats and individual handling of clients. Moreover, the transmission was transferred from connection-oriented TCP communication to connection-less UDP data transmission in order to make use of scalable multicast data transfer, which requires message splitting and regarding the effects of packet

loss. By adding audio and video streams, which are captured and transmitted by use of the *Java Media Framework* (JMF), this now *scalable VNC environment* can be used to deliver *synchronous electronic lectures* for *distance learning*.

In order to support the special requirements of an all-digital recording environment, we enriched the presentation capabilities by adding an (optional) easy to use *annotation system*, which can be used to emphasize certain presented elements during a lecture in order to focus the attention of the audience. Such an *annotation system* also enables on demand commenting and drawing of sketches, annotating presented slides and an additional electronic whiteboard, which are possibilities to replace the tradionally used chalk and blackboards.

The second *main goal* was the *automated production* of *full-fledged asynchronous electronic lectures*. The VNC environment can easily be extended to record VNC sessions by logging timestamped messages. We have adapted the *recording format* to enable *direct access* to message and rectangle headers, which improves the usability and performance during replay. As we have proved that *random access* can be performed efficiently for our recording, we have given the basis for other navigational features. The main drawback of the screen recording approach is the *missing structure* of the produced electronic lectures. We have shown how *analyzing the pixel-based recordings* makes it possible to *regain a structure*. Furthermore, the structuring process is *automated* by providing *empirically determined thresholds* for *slide and animation detection*. At first potential indices are derived and then classified.

The *index structure* is used to provide *slide-based navigation* via a comfortable *thumbnail overview* giving a meaningful *visual representation of indices*. The *indexing structure* also enables the *automated generation* of a *lecture related script* that will *include the annotations*, which were made during the lecture but are typically not included in published slides.

Moreover, we have discussed how the *indexing process* can be performed *on the fly* during the live presentation, which enables the use os slide indices in order to access previously presented slides including the corresponding annotations. By performing an efficient comparison of framebuffer states even an interlinkage of annotations and (pixel-based) content can be accomplished.

Retrievability of content is an important issue for *electronic lectures* since it enables students to locate and access certain topics of their interest. Textual (and any other) content is *stored pixel-based*. Nevertheless we suggested a possibility to create a *textual search base* which allows to perform *full text search* for *screen recorded lectures*: The *indexing structure* makes it possible to address a pixel-based representation of recorded slides. Applying *optical character recognition* to automatically generated *slide images* delivers the *textual content* of each slide, which then is interlinked with the *indexing structure* of the recording. Hence, we can perform *full text search* and receive accessable search results in the form of *slide indices*. If the character recognition delivers the coordinates of the recognized elements, we can even emphasize the *search results*.

Furthermore, we have suggested and implemented several solutions to enable *cross lecture searches* in order to handle the huge number of electronic lectures within a

database of lectures. Our own lecture recording archive currently contains about 600 hours of recorded lectures. The *online full text search* regards all individual lecture search bases and delivers the name and the indices of matching lectures for any given search pattern. With the *Search&BrowsingTool* we offer a more comfortable *cross lecture search* by classifying recordings according to metadata, such as name and date. By providing various *views*, a suitable representation of search results can be achieved.

Hence, we have overcome the limited *navigational* and missing *retrieval features* of the *screen recording approach*. Another drawback is the missing *scalability* of *pixel-based content*. *Electronic lectures* which are produced by use of *symbolic representations* including *vector fonts* and *vector graphics*, can be *scaled to any screen resolution* without *loss in quality*. Scaling *pixel-based recordings* will result in a loss of quality. Currently, we do not see how to eliminate this drawback. However, as many lecture recorders either use *pixel-based input formats* (to achieve more flexibility) or produce *electronic lectures* in a *standard video format*, which commonly are also *pixel-based*, this drawback should not be that important.

In summary, the *TeleTeachingTool* offers a *flexible* and *easy to use* solution to *lecture recording* and *scalable transmission*, which nevertheless produces *electronic lectures* of manageable file sizes and necessary *navigational* and *retrieval features*. The *annotation system* and the *profiling concept* increases the usability and comfort of the system. Since almost all aspects of the *content production* and *post-processing* are *automated*, the *TeleTeachingTool* produces *electronic lectures* in a fast and efficient manner.

10.1 Future Work

Currently the *TeleTeachingTool* offers already *flexible recording* and *automated post-processing* but the *publishing process* should also be automated. This requires an appropriate interface to a lecture archive, i.e. a web server, a learning management system or any other database. The *cross lecture search* may be extended to include *additional learning materials* or the *searchability of electronic lectures* should be enabled from within *conventional search engines* (perhaps within a learning management system). It might be useful to improve the *content analysis* of pixel-based recordings (for instance by use of color histograms) in order to enable the searching of slides with figures or images. Retrievability can be further improved by integrating support for *audio retrieval* by use of *speech recognition*.

Another meaningful approach of multimedia-based learning material is the integration of *digital student annotations* as suggested by [Schütz, 2002, Lienhard and Lauer, 2002, Lienhard and Zupancic, 2003]. Since TTT annotations are handled on a *separate layer* anyway, the concept of *annotation layering* can easily be added to the TTT protocol but an infrastructure for live annotating and exchanging annotations must be added. Currently the TTT does not support *textual annotations*, because typically the *freehand drawing feature* is preferred over typing textual notes during a live presentation. However, students probably will not have

suitable input devices such as an electronic pen. Hence, the TTT's annotating system should be extended to support *textual annotations*. Possibly other kinds of annotations such as links to other materials (e.g. web pages) will also be meaningful. The TTT protocol can be easily extended by adding additional *message types*. Unlike the original RFB protocol, which cannot handle unknown message types or encodings, the TTT protocol will simply skip any unknown messages, which is possible due to the *size tag* that precedes each message.

Regarding the online transmission, a protocol that respects a given bandwidth would be preferable instead of the currently used protocol, which generates high peaks while switching to another slide. Transmitting interlaced framebuffers can be useful to reduce the peaks and use the idle times of the transmission. Furthermore, *cooperative work* should be addressed by switching between multiple simultaneously connected desktops and supporting verbal communication between several participants and thus provide an environment for an *electronic classroom* or *electronic seminars*.

A

File sizes of recorded VNC Sessions

This appendix lists the *file sizes*, the *average per minute sizes* and the *average pixel densities*, i.e. the average number of pixels that are modified (updated) within a minute, for real live lectures that were recorded throughout the last years by use of the *TeleTeachingTool*. Note that some erroneous recordings (e.g. split due to network failure) as well as the few lectures with other resolutions are not listed here.

A.1 8 bit recordings with update stripes

The following courses were recorded with the first prototype of the *TeleTeachingTool*. The recordings contain *Hextile* encoded VNC sessions with *non-incremental update stripes* (12 stripes at a rate of $\frac{1}{10}$Hz and thus a 2 min period). No additional file compression was used. The resolution is 1024×768 pixels (or slightly less to make space for control elements) at 8 bit per pixel.

The sessions include mainly slide presentations and sometimes dynamic content (e.g. animated simulations and programming examples). Annotations (if applied) and pointer movements are generally stored pixel-based.

Course: "*Abstract Machines*" (Seidl/Wilhelm, 2002):

name	duration	size	density
abstrakt_2002_04_16_tr.vnc	92 min	9.3 Mbytes	103 kbytes/min
abstrakt_2002_04_23_tr.vnc	85 min	10.0 Mbytes	121 kbytes/min
abstrakt_2002_04_30_tr.vnc	91 min	11.5 Mbytes	130 kbytes/min
abstrakt_2002_05_07_tr.vnc	97 min	13.9 Mbytes	147 kbytes/min
abstrakt_2002_05_14_sb.vnc	65 min	12.4 Mbytes	195 kbytes/min
abstrakt_2002_05_28_sb.vnc	71 min	33.5 Mbytes	483 kbytes/min
abstrakt_2002_06_04_sb.vnc	47 min	21.3 Mbytes	465 kbytes/min
abstrakt_2002_06_11_tr.vnc	86 min	11.3 Mbytes	134 kbytes/min
abstrakt_2002_06_18_tr.vnc	86 min	12.1 Mbytes	144 kbytes/min
abstrakt_2002_06_25_tr.vnc	84 min	10.3 Mbytes	125 kbytes/min
abstrakt_2002_07_02_tr.vnc	90 min	10.3 Mbytes	118 kbytes/min

```
abstrakt_2002_07_09_tr.vnc  17 min   3.4 Mbytes  207 kbytes/min
-------------------------------------------------------------
                                    average:  197 kbytes/min
```

Course: "*Informatik I*" (Seidl, 2001/02):

```
name                  duration     size        density
-----------------------------------------------------------
info1_2001_10_30.vnc  78 min   3.4 Mbytes   45 kbytes/min
info1_2001_11_02.vnc  87 min   3.5 Mbytes   41 kbytes/min
info1_2001_11_06.vnc  86 min   3.4 Mbytes   41 kbytes/min
info1_2001_11_09.vnc  86 min   3.4 Mbytes   40 kbytes/min
info1_2001_11_13.vnc  81 min   2.4 Mbytes   31 kbytes/min
info1_2001_11_16.vnc  84 min   4.5 Mbytes   55 kbytes/min
info1_2001_11_20.vnc  87 min  14.2 Mbytes  168 kbytes/min
info1_2001_11_23.vnc  77 min   9.6 Mbytes  128 kbytes/min
info1_2001_11_27.vnc  88 min  16.8 Mbytes  195 kbytes/min
info1_2001_11_30.vnc  72 min   7.2 Mbytes  103 kbytes/min
info1_2001_12_04.vnc  88 min  15.0 Mbytes  175 kbytes/min
info1_2001_12_07.vnc  89 min  10.6 Mbytes  122 kbytes/min
info1_2001_12_11.vnc  85 min  17.7 Mbytes  214 kbytes/min
info1_2001_12_14.vnc  88 min  17.3 Mbytes  201 kbytes/min
info1_2001_12_18.vnc  88 min  18.3 Mbytes  213 kbytes/min
info1_2001_12_21.vnc  89 min  17.9 Mbytes  206 kbytes/min
info1_2002_01_08.vnc  57 min   0.8 Mbytes   15 kbytes/min
info1_2002_01_11.vnc  92 min  13.6 Mbytes  152 kbytes/min
info1_2002_01_15.vnc  90 min  17.9 Mbytes  204 kbytes/min
info1_2002_01_18.vnc  89 min  16.8 Mbytes  193 kbytes/min
info1_2002_01_22.vnc  89 min  12.7 Mbytes  146 kbytes/min
info1_2002_01_25.vnc  87 min  10.0 Mbytes  117 kbytes/min
info1_2002_01_29.vnc  88 min  17.1 Mbytes  199 kbytes/min
info1_2002_02_01.vnc  90 min  15.8 Mbytes  179 kbytes/min
info1_2002_02_05.vnc  89 min  14.5 Mbytes  167 kbytes/min
info1_2002_02_08.vnc  89 min  11.4 Mbytes  132 kbytes/min
info1_2002_02_12.vnc  86 min  14.5 Mbytes  173 kbytes/min
info1_2002_02_19.vnc  61 min   7.3 Mbytes  123 kbytes/min
-----------------------------------------------------------
                                  average:  134 kbytes/min
```

Course: "*Medienwissenschaft I*" (Bucher, 2002):

```
name                    duration     size        density
-------------------------------------------------------------
medien1_2002_04_16.vnc  87 min   9.9 Mbytes  117 kbytes/min
medien1_2002_04_23.vnc  88 min  13.7 Mbytes  159 kbytes/min
medien1_2002_04_30.vnc  87 min  16.9 Mbytes  198 kbytes/min
medien1_2002_05_07.vnc  86 min  12.8 Mbytes  152 kbytes/min
medien1_2002_05_14.vnc  91 min  15.8 Mbytes  178 kbytes/min
medien1_2002_05_28.vnc  78 min   6.6 Mbytes   86 kbytes/min
medien1_2002_06_11.vnc  84 min  16.8 Mbytes  205 kbytes/min
medien1_2002_06_18.vnc  94 min  17.1 Mbytes  186 kbytes/min
medien1_2002_06_25.vnc  95 min  12.7 Mbytes  137 kbytes/min
```

```
medien1_2002_07_09.vnc  86 min  26.7 Mbytes  317 kbytes/min
medien1_2002_07_16.vnc  77 min   6.9 Mbytes   92 kbytes/min
-----------------------------------------------------------
                                  average:  166 kbytes/min
```

Course: "*Technische Grundlagen des Elektronischen Publizierens im WWW*" (Meinel, 2001/02):

```
name                duration     size        density
-----------------------------------------------------------
tgep_2001_10_30.vnc  66 min   7.3 Mbytes  114 kbytes/min
tgep_2001_11_06.vnc  80 min  12.8 Mbytes  164 kbytes/min
tgep_2001_11_08.vnc  74 min   2.8 Mbytes   39 kbytes/min
tgep_2001_11_13.vnc  72 min   6.8 Mbytes   97 kbytes/min
tgep_2001_11_15.vnc  87 min   4.3 Mbytes   50 kbytes/min
tgep_2001_11_20.vnc  90 min   6.4 Mbytes   73 kbytes/min
tgep_2001_11_27.vnc  70 min   3.1 Mbytes   46 kbytes/min
tgep_2001_11_29.vnc  78 min   4.3 Mbytes   57 kbytes/min
tgep_2001_12_04.vnc  50 min   3.9 Mbytes   80 kbytes/min
tgep_2001_12_06.vnc  82 min   4.6 Mbytes   58 kbytes/min
tgep_2001_12_11.vnc  70 min   3.9 Mbytes   58 kbytes/min
tgep_2001_12_13.vnc  78 min   3.7 Mbytes   49 kbytes/min
tgep_2001_12_18.vnc  74 min   2.9 Mbytes   40 kbytes/min
tgep_2002_01_08.vnc  81 min  12.7 Mbytes  161 kbytes/min
tgep_2002_01_10.vnc  83 min   6.4 Mbytes   80 kbytes/min
tgep_2002_01_15.vnc  69 min   3.3 Mbytes   49 kbytes/min
tgep_2002_01_17.vnc  77 min   4.4 Mbytes   59 kbytes/min
tgep_2002_01_22.vnc  76 min   4.2 Mbytes   56 kbytes/min
tgep_2002_01_29.vnc  87 min   9.3 Mbytes  110 kbytes/min
tgep_2002_01_31.vnc  77 min   5.7 Mbytes   76 kbytes/min
tgep_2002_02_05.vnc  61 min   4.8 Mbytes   81 kbytes/min
tgep_2002_02_07.vnc  76 min   6.0 Mbytes   82 kbytes/min
tgep_2002_02_14.vnc  79 min   6.7 Mbytes   87 kbytes/min
tgep_2002_02_19.vnc  78 min   6.6 Mbytes   87 kbytes/min
tgep_2002_02_21.vnc  70 min   8.2 Mbytes  121 kbytes/min
-----------------------------------------------------------
                                 average:   78 kbytes/min
```

A.2 16 bit recordings with stripes and file compression

The following courses were recorded with the *TeleTeachingTool* with additional *zlib deflate* compression (applied to the file body). The recordings contain *Hextile* encoded VNC sessions with *non-incremental update stripes* (24 stripes at a rate of $\frac{1}{5}$Hz and thus a 2 min period). The resulution is 1024×768 pixels (or slightly less to make space for control elements) at 16 bit per pixel.

The following sessions include mainly slide presentations and some dynamic content (e.g. animated simulations and programming examples). Annotations and pointer movements are generally stored symbolically.

Course: "*Abstrakte Maschinen*" (Seidl, 2003):

name	duration	size	density	pixel density
abstrakt_2003_04_29.ttt	91 min	2.3 Mbytes	26 kbytes/min	1255 kpixel/min
abstrakt_2003_05_06.ttt	105 min	2.8 Mbytes	28 kbytes/min	2069 kpixel/min
abstrakt_2003_05_13.ttt	77 min	1.7 Mbytes	23 kbytes/min	1173 kpixel/min
abstrakt_2003_05_20.ttt	95 min	2.3 Mbytes	25 kbytes/min	1407 kpixel/min
abstrakt_2003_05_27.ttt	88 min	2.3 Mbytes	27 kbytes/min	1826 kpixel/min
abstrakt_2003_06_03.ttt	91 min	2.3 Mbytes	27 kbytes/min	2087 kpixel/min
abstrakt_2003_06_17.ttt	92 min	2.9 Mbytes	32 kbytes/min	2743 kpixel/min
abstrakt_2003_06_24.ttt	65 min	2.8 Mbytes	44 kbytes/min	2846 kpixel/min
abstrakt_2003_06_25.ttt	92 min	5.9 Mbytes	65 kbytes/min	7028 kpixel/min
abstrakt_2003_07_08.ttt	91 min	2.5 Mbytes	28 kbytes/min	1785 kpixel/min
abstrakt_2003_07_15.ttt	89 min	2.6 Mbytes	30 kbytes/min	1096 kpixel/min
abstrakt_2003_07_16.ttt	61 min	1.9 Mbytes	31 kbytes/min	1061 kpixel/min
abstrakt_2003_07_29.ttt	81 min	2.8 Mbytes	35 kbytes/min	2211 kpixel/min
		average:	32 kbytes/min	2199 kpixel/min

Course: "*Abstrakte Maschinen*" (Seidl, 2004):

name	duration	size	density	pixel density
abstrakt_2004_04_19.ttt	73 min	1.9 Mbytes	27 kbytes/min	869 kpixel/min
abstrakt_2004_04_21.ttt	85 min	2.1 Mbytes	26 kbytes/min	1277 kpixel/min
abstrakt_2004_04_26.ttt	86 min	2.6 Mbytes	31 kbytes/min	1306 kpixel/min
abstrakt_2004_04_28.ttt	89 min	2.1 Mbytes	24 kbytes/min	814 kpixel/min
abstrakt_2004_05_03.ttt	90 min	6.2 Mbytes	71 kbytes/min	1658 kpixel/min
abstrakt_2004_05_05.ttt	88 min	2.4 Mbytes	28 kbytes/min	865 kpixel/min
abstrakt_2004_05_12.ttt	88 min	2.3 Mbytes	27 kbytes/min	1540 kpixel/min
abstrakt_2004_05_19.ttt	77 min	3.5 Mbytes	47 kbytes/min	10354 kpixel/min
abstrakt_2004_05_26.ttt	87 min	2.2 Mbytes	26 kbytes/min	1250 kpixel/min
abstrakt_2004_06_09.ttt	85 min	3.1 Mbytes	37 kbytes/min	2417 kpixel/min
abstrakt_2004_06_16.ttt	88 min	3.5 Mbytes	40 kbytes/min	2100 kpixel/min
abstrakt_2004_06_23.ttt	77 min	4.6 Mbytes	61 kbytes/min	5115 kpixel/min
abstrakt_2004_06_30.ttt	89 min	2.9 Mbytes	33 kbytes/min	1389 kpixel/min
abstrakt_2004_07_07.ttt	85 min	2.3 Mbytes	27 kbytes/min	1199 kpixel/min
abstrakt_2004_07_14.ttt	40 min	1.0 Mbytes	25 kbytes/min	1083 kpixel/min
abstrakt_2004_07_21.ttt	74 min	1.5 Mbytes	21 kbytes/min	1052 kpixel/min
		average:	34 kbytes/min	2143 kpixel/min
ignoring abstrakt_2004_05_19.ttt		average:	31 kbytes/min	1344 kpixel/min

Course: "*Programmiersprachen*" (Berlea, 2005/06):

name	duration	size	density	pixel density
prgsprachen_2005_10_19.ttt	92 min	5.6 Mbytes	62 kbytes/min	1847 kpixel/min
prgsprachen_2005_10_26.ttt	87 min	5.7 Mbytes	67 kbytes/min	1747 kpixel/min

prgsprachen_2005_11_02.ttt	89 min	5.1 Mbytes	58 kbytes/min	1895 kpixel/min
prgsprachen_2005_11_09.ttt	86 min	3.8 Mbytes	45 kbytes/min	1072 kpixel/min
prgsprachen_2005_11_23.ttt	87 min	6.8 Mbytes	80 kbytes/min	2072 kpixel/min
prgsprachen_2005_11_30.ttt	79 min	4.4 Mbytes	58 kbytes/min	1016 kpixel/min
prgsprachen_2005_12_07.ttt	84 min	4.4 Mbytes	54 kbytes/min	1410 kpixel/min
prgsprachen_2005_12_14.ttt	83 min	5.0 Mbytes	62 kbytes/min	1781 kpixel/min
prgsprachen_2006_01_11.ttt	83 min	4.1 Mbytes	51 kbytes/min	1210 kpixel/min
prgsprachen_2006_01_18.ttt	79 min	3.7 Mbytes	48 kbytes/min	1035 kpixel/min
prgsprachen_2006_01_25.ttt	81 min	4.8 Mbytes	61 kbytes/min	1357 kpixel/min
prgsprachen_2006_02_08.ttt	78 min	3.2 Mbytes	42 kbytes/min	799 kpixel/min
		average:	57 kbytes/min	1436 kpixel/min

Course: "*Informatik I*" (Seidl, 2002/03):

name	duration	size	density	pixel density
info1_2002_10_29.ttt	85 min	2.4 Mbytes	29 kbytes/min	659 kpixel/min
info1_2002_11_05.ttt	89 min	4.1 Mbytes	48 kbytes/min	1030 kpixel/min
info1_2002_11_08.ttt	88 min	3.9 Mbytes	46 kbytes/min	1003 kpixel/min
info1_2002_11_12.ttt	88 min	5.0 Mbytes	58 kbytes/min	1604 kpixel/min
info1_2002_11_15.ttt	77 min	8.6 Mbytes	114 kbytes/min	1937 kpixel/min
info1_2002_11_19.ttt	90 min	3.9 Mbytes	44 kbytes/min	1623 kpixel/min
info1_2002_11_22.ttt	90 min	3.5 Mbytes	40 kbytes/min	1326 kpixel/min
info1_2002_11_26.ttt	90 min	4.8 Mbytes	55 kbytes/min	1912 kpixel/min
info1_2002_11_29.ttt	89 min	2.4 Mbytes	28 kbytes/min	1036 kpixel/min
info1_2002_12_03.ttt	81 min	1.7 Mbytes	22 kbytes/min	882 kpixel/min
info1_2002_12_06.ttt	89 min	1.8 Mbytes	21 kbytes/min	863 kpixel/min
info1_2002_12_10.ttt	89 min	3.0 Mbytes	35 kbytes/min	940 kpixel/min
info1_2002_12_13.ttt	90 min	4.2 Mbytes	47 kbytes/min	1115 kpixel/min
info1_2002_12_17.ttt	91 min	3.7 Mbytes	42 kbytes/min	978 kpixel/min
info1_2002_12_20.ttt	90 min	3.4 Mbytes	39 kbytes/min	764 kpixel/min
info1_2003_01_07.ttt	79 min	2.8 Mbytes	37 kbytes/min	1033 kpixel/min
info1_2003_01_10.ttt	87 min	2.9 Mbytes	35 kbytes/min	798 kpixel/min
info1_2003_01_14.ttt	88 min	2.9 Mbytes	34 kbytes/min	854 kpixel/min
info1_2003_01_17.ttt	88 min	2.4 Mbytes	28 kbytes/min	1139 kpixel/min
info1_2003_01_21.ttt	88 min	2.8 Mbytes	33 kbytes/min	962 kpixel/min
info1_2003_01_24.ttt	89 min	2.9 Mbytes	33 kbytes/min	680 kpixel/min
info1_2003_01_28.ttt	91 min	2.7 Mbytes	30 kbytes/min	811 kpixel/min
info1_2003_01_31.ttt	90 min	5.1 Mbytes	58 kbytes/min	1309 kpixel/min
info1_2003_02_04.ttt	90 min	3.9 Mbytes	45 kbytes/min	1165 kpixel/min
info1_2003_02_07.ttt	89 min	4.3 Mbytes	49 kbytes/min	1038 kpixel/min
info1_2003_02_11.ttt	89 min	3.6 Mbytes	41 kbytes/min	1177 kpixel/min
info1_2003_02_18.ttt	79 min	4.0 Mbytes	52 kbytes/min	1269 kpixel/min
		average:	42 kbytes/min	1107 kpixel/min

Course: "*Informatik I*" (Seidl, 2004/05):

name	duration	size	density	pixel density
info1_2004_10_21.ttt	48 min	1.8 Mbytes	39 kbytes/min	819 kpixel/min

info1_2004_10_22.ttt	83 min	3.5 Mbytes	44 kbytes/min	1078 kpixel/min
info1_2004_10_28.ttt	89 min	1.9 Mbytes	22 kbytes/min	585 kpixel/min
info1_2004_10_29.ttt	83 min	1.8 Mbytes	22 kbytes/min	532 kpixel/min
info1_2004_11_04.ttt	88 min	3.6 Mbytes	42 kbytes/min	1039 kpixel/min
info1_2004_11_11.ttt	89 min	2.3 Mbytes	26 kbytes/min	1024 kpixel/min
info1_2004_11_12.ttt	86 min	2.2 Mbytes	26 kbytes/min	1013 kpixel/min
info1_2004_11_18.ttt	87 min	2.3 Mbytes	27 kbytes/min	1477 kpixel/min
info1_2004_11_25.ttt	86 min	3.1 Mbytes	37 kbytes/min	1017 kpixel/min
info1_2004_11_26.ttt	86 min	2.1 Mbytes	25 kbytes/min	1064 kpixel/min
info1_2004_12_03.ttt	87 min	1.8 Mbytes	21 kbytes/min	801 kpixel/min
info1_2004_12_09.ttt	84 min	3.0 Mbytes	36 kbytes/min	995 kpixel/min
info1_2004_12_10.ttt	85 min	2.9 Mbytes	35 kbytes/min	754 kpixel/min
info1_2004_12_16.ttt	78 min	2.8 Mbytes	37 kbytes/min	788 kpixel/min
info1_2004_12_17.ttt	86 min	3.0 Mbytes	35 kbytes/min	656 kpixel/min
info1_2004_12_23.ttt	86 min	3.3 Mbytes	39 kbytes/min	1487 kpixel/min
info1_2005_01_07.ttt	92 min	2.8 Mbytes	31 kbytes/min	787 kpixel/min
info1_2005_01_13.ttt	82 min	2.0 Mbytes	25 kbytes/min	587 kpixel/min
info1_2005_01_14.ttt	85 min	2.2 Mbytes	26 kbytes/min	666 kpixel/min
info1_2005_01_20.ttt	85 min	2.7 Mbytes	32 kbytes/min	841 kpixel/min
info1_2005_01_21.ttt	86 min	2.6 Mbytes	32 kbytes/min	817 kpixel/min
info1_2005_01_27.ttt	83 min	2.6 Mbytes	32 kbytes/min	823 kpixel/min
info1_2005_01_28.ttt	85 min	3.5 Mbytes	42 kbytes/min	771 kpixel/min
info1_2005_02_03.ttt	84 min	3.3 Mbytes	40 kbytes/min	934 kpixel/min
info1_2005_02_04.ttt	83 min	7.9 Mbytes	98 kbytes/min	2320 kpixel/min
		average:	34 kbytes/min	947 kpixel/min

Course: "*Informatik II*" (Seidl, 2005):

name	duration	size	density	pixel density
info2_2005_04_12.ttt	86 min	3.7 Mbytes	44 kbytes/min	850 kpixel/min
info2_2005_04_15.ttt	84 min	5.3 Mbytes	65 kbytes/min	1067 kpixel/min
info2_2005_04_19.ttt	90 min	4.4 Mbytes	50 kbytes/min	1210 kpixel/min
info2_2005_04_22.ttt	88 min	4.0 Mbytes	46 kbytes/min	779 kpixel/min
info2_2005_04_26.ttt	88 min	3.3 Mbytes	38 kbytes/min	917 kpixel/min
info2_2005_04_29.ttt	88 min	2.3 Mbytes	27 kbytes/min	856 kpixel/min
info2_2005_05_03.ttt	90 min	2.3 Mbytes	26 kbytes/min	938 kpixel/min
info2_2005_05_06.ttt	88 min	2.6 Mbytes	30 kbytes/min	1007 kpixel/min
info2_2005_05_10.ttt	90 min	2.4 Mbytes	28 kbytes/min	918 kpixel/min
info2_2005_05_13.ttt	93 min	5.5 Mbytes	60 kbytes/min	1216 kpixel/min
info2_2005_05_20.ttt	85 min	3.0 Mbytes	36 kbytes/min	781 kpixel/min
info2_2005_05_24.ttt	89 min	4.0 Mbytes	46 kbytes/min	916 kpixel/min
info2_2005_05_27.ttt	82 min	2.9 Mbytes	37 kbytes/min	661 kpixel/min
info2_2005_05_31.ttt	90 min	3.4 Mbytes	39 kbytes/min	733 kpixel/min
info2_2005_06_03.ttt	88 min	3.5 Mbytes	41 kbytes/min	680 kpixel/min
info2_2005_06_07.ttt	85 min	3.0 Mbytes	36 kbytes/min	687 kpixel/min
info2_2005_06_10.ttt	83 min	2.8 Mbytes	35 kbytes/min	890 kpixel/min
info2_2005_06_14.ttt	88 min	4.6 Mbytes	54 kbytes/min	1055 kpixel/min
info2_2005_06_17.ttt	74 min	2.9 Mbytes	40 kbytes/min	710 kpixel/min
info2_2005_06_21.ttt	83 min	1.8 Mbytes	22 kbytes/min	585 kpixel/min
info2_2005_06_24.ttt	82 min	2.3 Mbytes	29 kbytes/min	651 kpixel/min

info2_2005_06_28.ttt	86 min	3.5 Mbytes	42 kbytes/min	731 kpixel/min
info2_2005_07_01.ttt	87 min	3.8 Mbytes	44 kbytes/min	724 kpixel/min
info2_2005_07_05.ttt	86 min	3.1 Mbytes	37 kbytes/min	615 kpixel/min
info2_2005_07_08.ttt	84 min	3.1 Mbytes	38 kbytes/min	731 kpixel/min
info2_2005_07_12.ttt	73 min	4.0 Mbytes	57 kbytes/min	1680 kpixel/min
		average:	40 kbytes/min	868 kpixel/min

The following sessions include ***pixel intensive*** slide presentations (e.g. animated content, pixel-based annotations, high colored pictures or scanned newspaper articles).

Course: "*Medienwissenschaft I*" (Bucher, 2003/04):

name	duration	size	density	pixel density
medien1_2003_10_28.ttt	101 min	31.8 Mbytes	323 kbytes/min	842 kpixel/min
medien1_2003_11_04.ttt	95 min	17.7 Mbytes	191 kbytes/min	897 kpixel/min
medien1_2003_11_11.ttt	84 min	9.1 Mbytes	111 kbytes/min	640 kpixel/min
medien1_2003_11_18.ttt	92 min	9.1 Mbytes	101 kbytes/min	626 kpixel/min
medien1_2003_11_25.ttt	87 min	10.0 Mbytes	118 kbytes/min	606 kpixel/min
medien1_2003_12_02.ttt	92 min	21.9 Mbytes	244 kbytes/min	752 kpixel/min
medien1_2003_12_09.ttt	65 min	8.2 Mbytes	129 kbytes/min	762 kpixel/min
medien1_2004_01_06.ttt	94 min	12.4 Mbytes	136 kbytes/min	662 kpixel/min
medien1_2004_01_13.ttt	95 min	24.4 Mbytes	263 kbytes/min	800 kpixel/min
medien1_2004_01_27.ttt	68 min	22.6 Mbytes	341 kbytes/min	1237 kpixel/min
medien1_2004_02_03.ttt	89 min	15.0 Mbytes	173 kbytes/min	830 kpixel/min
medien1_2004_02_10.ttt	86 min	7.1 Mbytes	84 kbytes/min	673 kpixel/min
medien1_2004_02_17.ttt	89 min	17.4 Mbytes	200 kbytes/min	765 kpixel/min
		average:	185 kbytes/min	776 kpixel/min

Course: "*Medienwissenschaft II*" (Bucher, 2002/03):

name	duration	size	density	pixel density
medien2_2002_11_05.ttt	65 min	10.8 Mbytes	170 kbytes/min	713 kpixel/min
medien2_2002_11_26.ttt	70 min	12.5 Mbytes	183 kbytes/min	1036 kpixel/min
medien2_2002_12_03.ttt	93 min	22.2 Mbytes	244 kbytes/min	854 kpixel/min
medien2_2002_12_17.ttt	90 min	19.5 Mbytes	222 kbytes/min	936 kpixel/min
medien2_2003_01_07.ttt	92 min	30.8 Mbytes	343 kbytes/min	1006 kpixel/min
medien2_2003_01_14.ttt	94 min	25.2 Mbytes	274 kbytes/min	815 kpixel/min
medien2_2003_01_21.ttt	98 min	21.8 Mbytes	227 kbytes/min	560 kpixel/min
medien2_2003_01_28.ttt	93 min	17.8 Mbytes	197 kbytes/min	862 kpixel/min
medien2_2003_02_11.ttt	92 min	3.5 Mbytes	39 kbytes/min	638 kpixel/min
medien2_2003_02_18.ttt	82 min	11.3 Mbytes	141 kbytes/min	764 kpixel/min
		average:	204 kbytes/min	818 kpixel/min

Course: "*Informatik III*" (Schlichter, 2005/06):

name	duration	size	density	pixel density

```
-------------------------------------------------------------------------
info3_2005_10_18.ttt  85 min  10.3 Mbytes  124 kbytes/min  1087 kpixel/min
info3_2005_10_24.ttt  73 min   7.5 Mbytes  106 kbytes/min  1132 kpixel/min
info3_2005_10_25.ttt  85 min  11.5 Mbytes  138 kbytes/min  1156 kpixel/min
info3_2005_10_31.ttt  87 min   9.4 Mbytes  110 kbytes/min  1013 kpixel/min
info3_2005_11_07.ttt  89 min   9.2 Mbytes  106 kbytes/min   922 kpixel/min
info3_2005_11_08.ttt  84 min  10.0 Mbytes  122 kbytes/min  1011 kpixel/min
info3_2005_11_14.ttt  89 min  13.0 Mbytes  149 kbytes/min  1347 kpixel/min
info3_2005_11_21.ttt  92 min  13.2 Mbytes  147 kbytes/min  1290 kpixel/min
info3_2005_11_22.ttt  89 min  14.2 Mbytes  164 kbytes/min  1292 kpixel/min
info3_2005_11_28.ttt  91 min  11.3 Mbytes  128 kbytes/min  1166 kpixel/min
info3_2005_11_29.ttt  88 min  15.8 Mbytes  184 kbytes/min  1550 kpixel/min
info3_2005_12_05.ttt  86 min  11.3 Mbytes  134 kbytes/min  1165 kpixel/min
info3_2005_12_06.ttt  90 min  10.8 Mbytes  123 kbytes/min  1174 kpixel/min
info3_2005_12_12.ttt  90 min  11.9 Mbytes  135 kbytes/min  1220 kpixel/min
info3_2005_12_13.ttt  90 min  18.6 Mbytes  212 kbytes/min  1582 kpixel/min
info3_2005_12_19.ttt  89 min  13.4 Mbytes  154 kbytes/min  1305 kpixel/min
info3_2006_01_09.ttt  86 min  10.8 Mbytes  128 kbytes/min  1134 kpixel/min
info3_2006_01_10.ttt  84 min  14.1 Mbytes  172 kbytes/min  1823 kpixel/min
info3_2006_01_16.ttt  80 min   9.3 Mbytes  120 kbytes/min   946 kpixel/min
info3_2006_01_17.ttt  87 min  10.1 Mbytes  119 kbytes/min  1108 kpixel/min
info3_2006_01_23.ttt  85 min  15.5 Mbytes  187 kbytes/min  1469 kpixel/min
info3_2006_01_24.ttt  89 min  13.8 Mbytes  159 kbytes/min  1318 kpixel/min
info3_2006_01_30.ttt  81 min  15.3 Mbytes  194 kbytes/min  1598 kpixel/min
info3_2006_01_31.ttt  86 min  12.1 Mbytes  145 kbytes/min  1190 kpixel/min
info3_2006_02_06.ttt  84 min  12.4 Mbytes  151 kbytes/min  1300 kpixel/min
-------------------------------------------------------------------------
                                 average:  144 kbytes/min  1251 kpixel/min
```

A.3 32 bit recordings with file compression but no stripes

The following courses were recorded with the *TeleTeachingTool* with additional *zlib deflate* compression (applied to the file body). The recordings contain *Hextile* encoded VNC sessions ***without*** *non-incremental update stripes* The resulution is 1024×768 pixels (or slightly less to make space for control elements) at color depth of 24 bit but pixel values are stored at 4 byte per pixel (32 bit).

The sessions include mainly slide presentations and some dynamic content (e.g. animated simulations and programming examples). Annotations and pointer movements are generally stored symbolically.

Course: "*Abstrakte Maschinen*" (Seidl, 2006):

```
name                    duration     size        density        pixel density
-------------------------------------------------------------------------------
abstrakt_2006_05_18.ttt  89 min   2.2 Mbytes   25 kbytes/min   709 kpixel/min
abstrakt_2006_06_01.ttt  87 min   0.8 Mbytes    9 kbytes/min   286 kpixel/min
abstrakt_2006_06_08.ttt  67 min  28.7 Mbytes  439 kbytes/min 30071 kpixel/min
abstrakt_2006_06_22.ttt  82 min   1.1 Mbytes   14 kbytes/min   483 kpixel/min
```

```
abstrakt_2006_06_29.ttt  85 min   1.9 Mbytes   23 kbytes/min   373 kpixel/min
abstrakt_2006_07_06.ttt  86 min   1.6 Mbytes   19 kbytes/min   648 kpixel/min
abstrakt_2006_07_13.ttt  91 min   1.2 Mbytes   14 kbytes/min   484 kpixel/min
abstrakt_2006_07_20.ttt  90 min   1.8 Mbytes   20 kbytes/min   546 kpixel/min
abstrakt_2006_07_27.ttt  29 min   0.4 Mbytes   16 kbytes/min   455 kpixel/min
-------------------------------------------------------------------------------
                                  average:   64 kbytes/min  3783 kpixel/min
-------------------------------------------------------------------------------
ignoring abstrakt_2006_06_08.ttt  average:   15 kbytes/min   442 kpixel/min
```

Without the pixel intensive lecture of the 2006/06/08 (which obviously is a massive outlier) an average of 17 kbytes/min and 498 kpixel/min are achieved.

Course: "*Compilerbau*" (Seidl, 2006):

```
name                     duration     size        density        pixel density
-------------------------------------------------------------------------------
compiler_2006_04_26.ttt  89 min   0.6 Mbytes    7 kbytes/min   224 kpixel/min
compiler_2006_05_03.ttt  87 min   2.5 Mbytes   29 kbytes/min  1021 kpixel/min
compiler_2006_05_08.ttt  88 min   2.3 Mbytes   27 kbytes/min   492 kpixel/min
compiler_2006_05_15.ttt  88 min   1.3 Mbytes   15 kbytes/min   451 kpixel/min
compiler_2006_05_17.ttt  86 min   2.4 Mbytes   29 kbytes/min   547 kpixel/min
compiler_2006_05_22.ttt  89 min   4.0 Mbytes   46 kbytes/min   685 kpixel/min
compiler_2006_05_24.ttt  90 min   4.1 Mbytes   47 kbytes/min   786 kpixel/min
compiler_2006_05_29.ttt  85 min  14.9 Mbytes  180 kbytes/min  2392 kpixel/min
compiler_2006_05_31.ttt  88 min   2.2 Mbytes   26 kbytes/min   506 kpixel/min
compiler_2006_06_07.ttt  69 min   1.9 Mbytes   28 kbytes/min   499 kpixel/min
compiler_2006_06_12.ttt  85 min   2.4 Mbytes   29 kbytes/min   427 kpixel/min
compiler_2006_06_14.ttt  80 min   1.2 Mbytes   15 kbytes/min   393 kpixel/min
compiler_2006_06_19.ttt  89 min   1.8 Mbytes   21 kbytes/min   493 kpixel/min
compiler_2006_06_21.ttt  87 min   1.6 Mbytes   19 kbytes/min   445 kpixel/min
compiler_2006_06_26.ttt  74 min   3.6 Mbytes   50 kbytes/min   751 kpixel/min
compiler_2006_06_28.ttt  89 min   4.9 Mbytes   56 kbytes/min   917 kpixel/min
compiler_2006_07_03.ttt  87 min   4.1 Mbytes   48 kbytes/min   837 kpixel/min
compiler_2006_07_05.ttt  78 min   1.1 Mbytes   15 kbytes/min   370 kpixel/min
compiler_2006_07_10.ttt  88 min   3.4 Mbytes   39 kbytes/min   567 kpixel/min
compiler_2006_07_12.ttt  89 min   1.0 Mbytes   12 kbytes/min   306 kpixel/min
compiler_2006_07_17.ttt  86 min   1.0 Mbytes   12 kbytes/min   282 kpixel/min
compiler_2006_07_19.ttt  89 min   1.6 Mbytes   18 kbytes/min   404 kpixel/min
compiler_2006_07_24.ttt  82 min   2.5 Mbytes   31 kbytes/min   546 kpixel/min
compiler_2006_07_26.ttt  55 min   1.7 Mbytes   33 kbytes/min   667 kpixel/min
-------------------------------------------------------------------------------
                                  average:   34 kbytes/min   625 kpixel/min
```

B

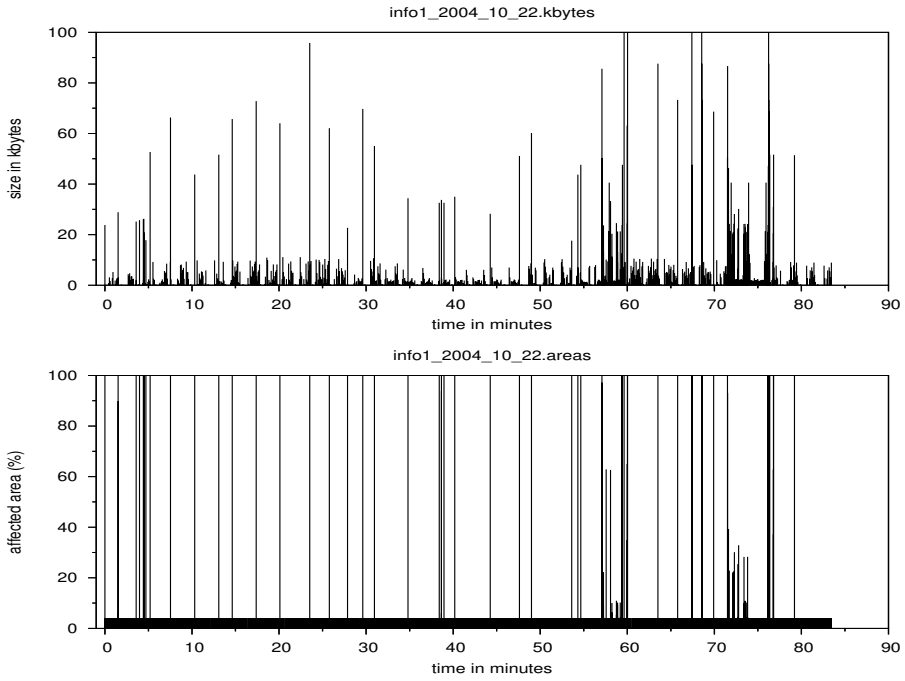
Message sizes *by-byte* and *by-area*

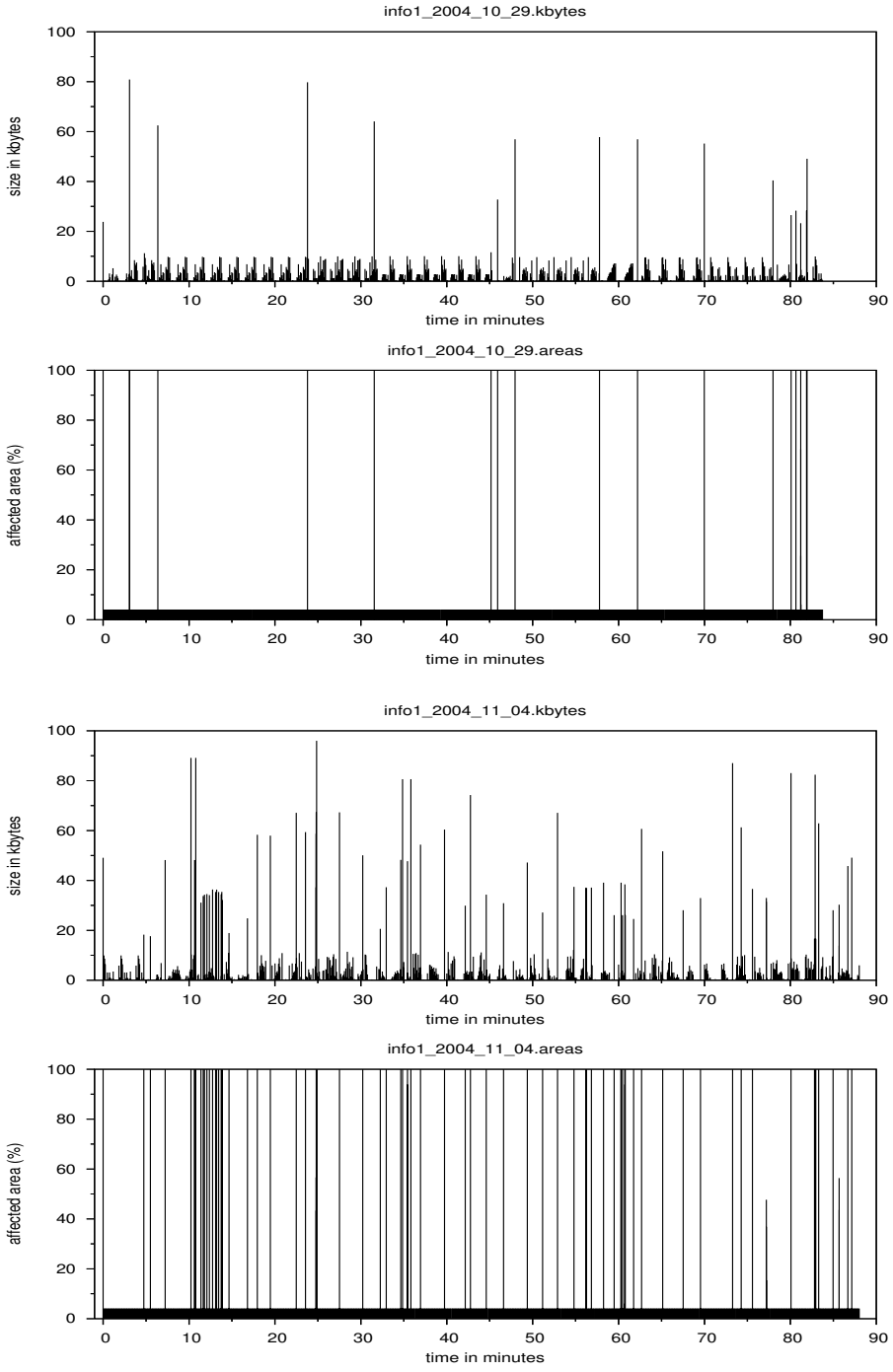
This Appendix lists the message size in bytes and in proportion to the resolution of the framebuffer of some recorded lectures.

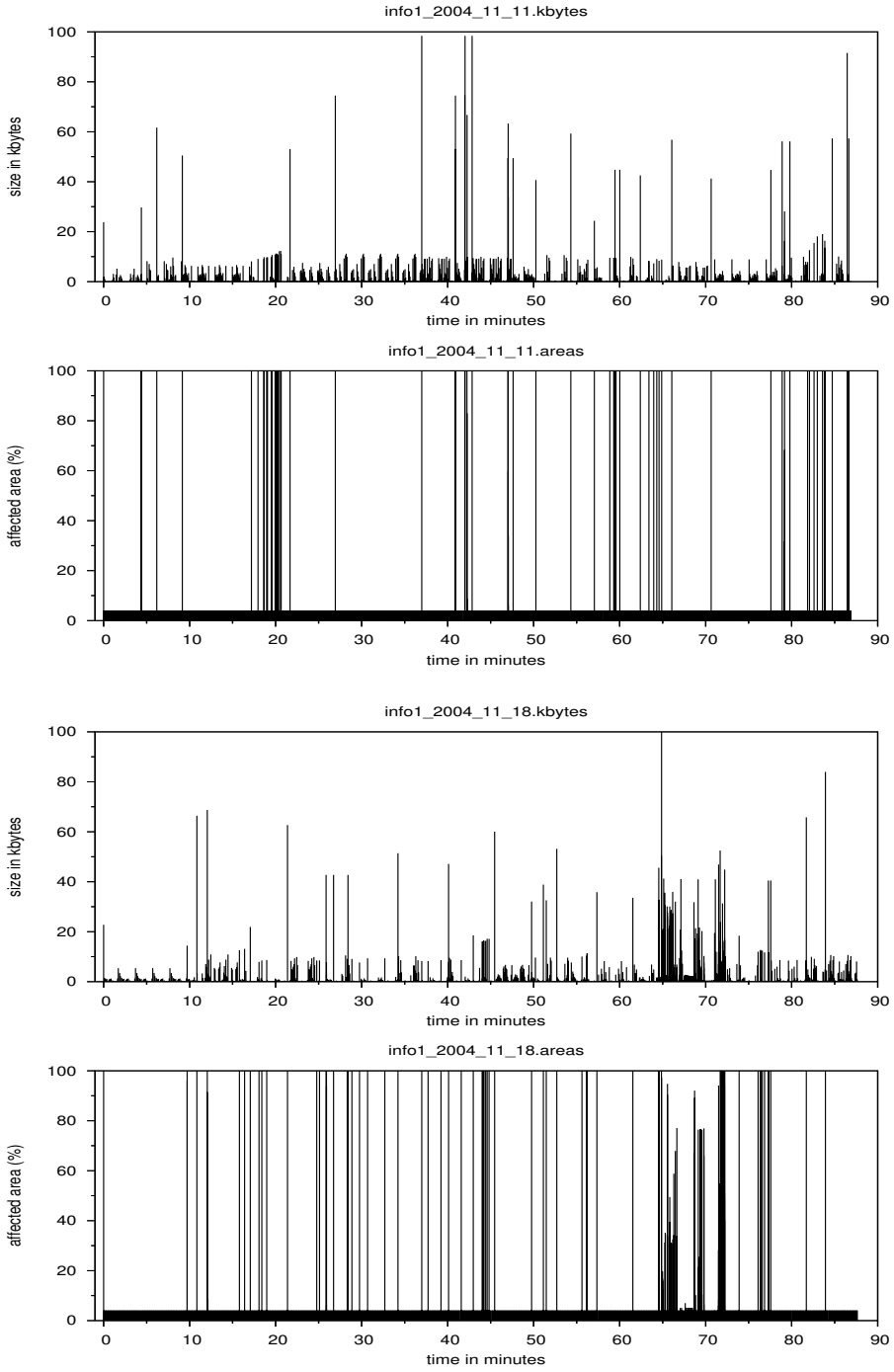
B.1 Einführung in die Informatik I [WS2004/05]

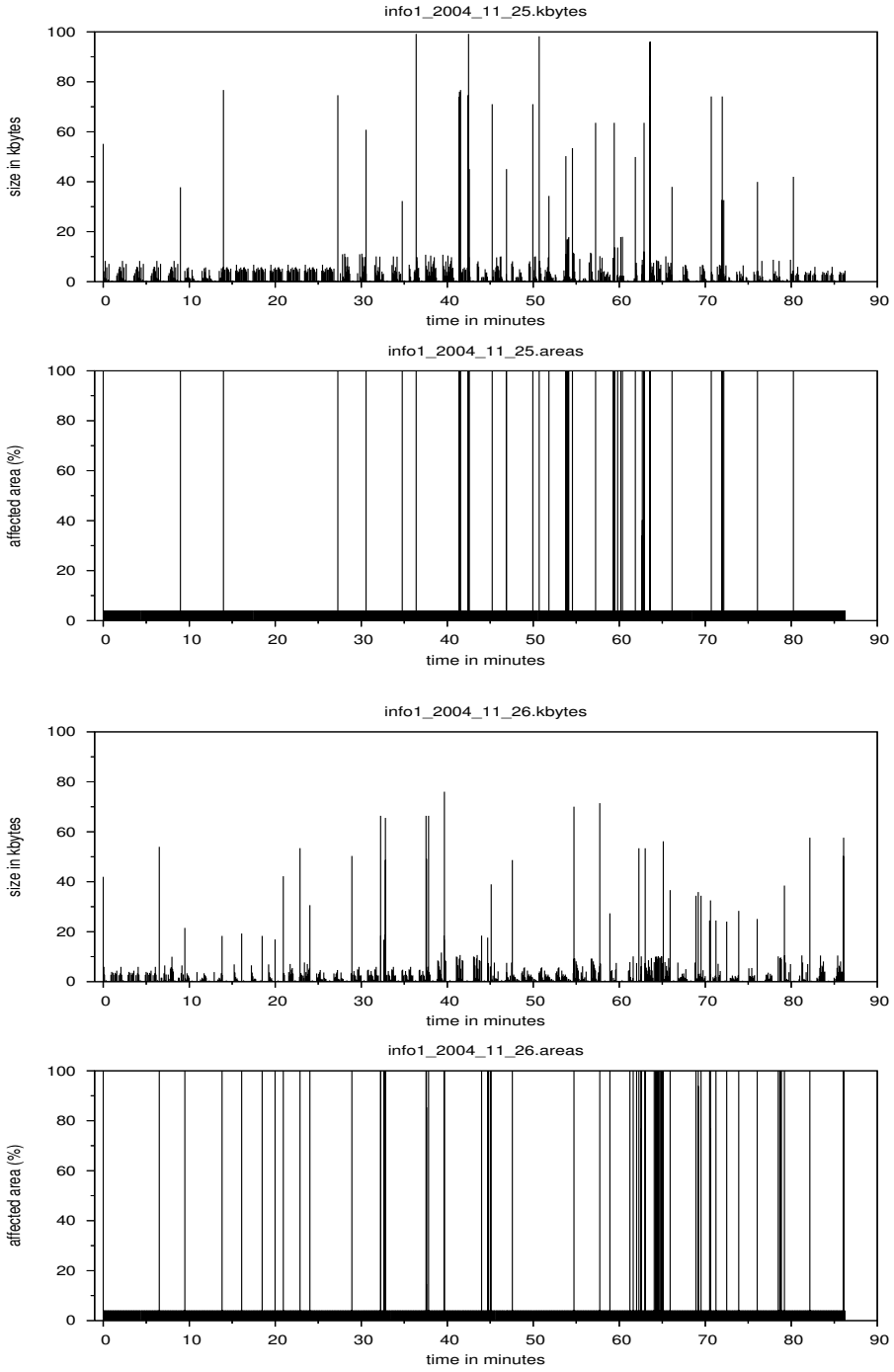
Prof. Dr. Helmut Seidl / Technische Universität München

1024 x 768 (16 bit truecolor), 16 bits per pixel, 2 bytes per pixel
with keyframe stripes




info1_2004_10_29.kbytes
size in kbytes
time in minutes
info1_2004_10_29.areas
affected area (%)
time in minutes
info1_2004_11_04.kbytes
size in kbytes
time in minutes
info1_2004_11_04.areas
affected area (%)
time in minutes

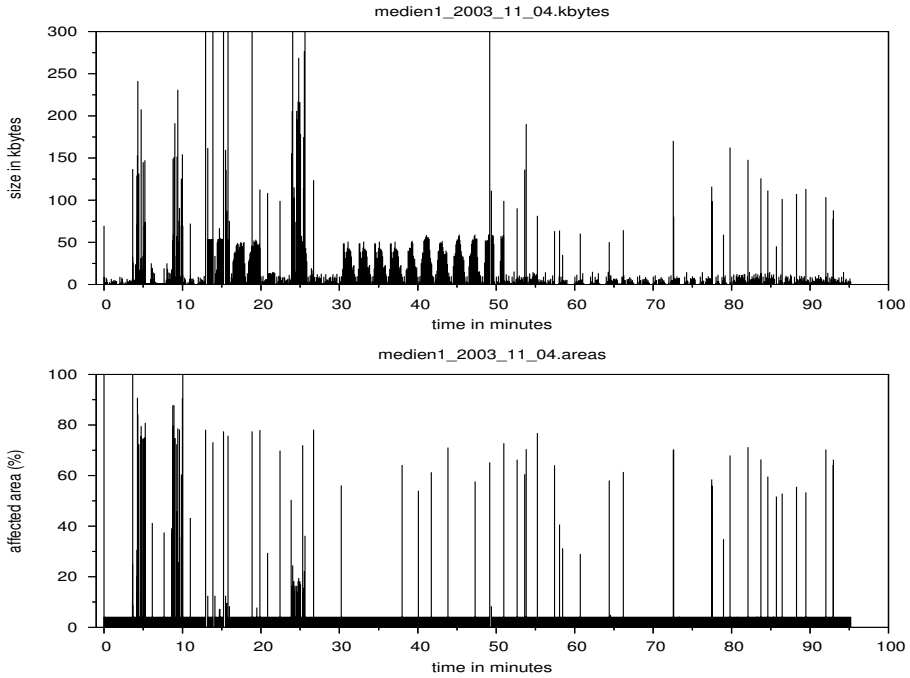

info1_2004_11_11.kbytes
size in kbytes
time in minutes
info1_2004_11_11.areas
affected area (%)
time in minutes
info1_2004_11_18.kbytes
size in kbytes
time in minutes
info1_2004_11_18.areas
affected area (%)
time in minutes

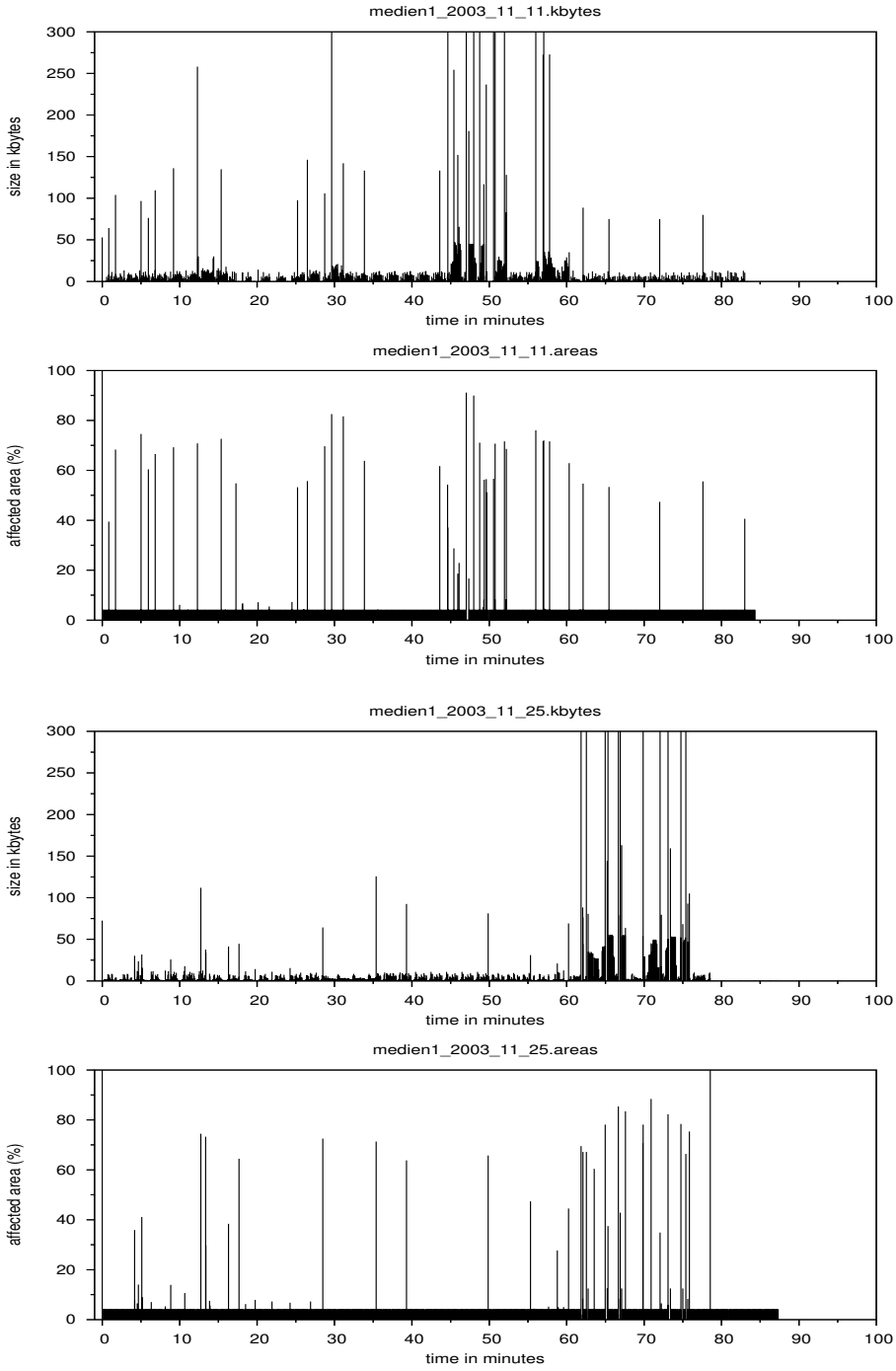

info1_2004_11_25.kbytes
size in kbytes
time in minutes
info1_2004_11_25.areas
affected area (%)
time in minutes
info1_2004_11_26.kbytes
size in kbytes
time in minutes
info1_2004_11_26.areas
affected area (%)
time in minutes

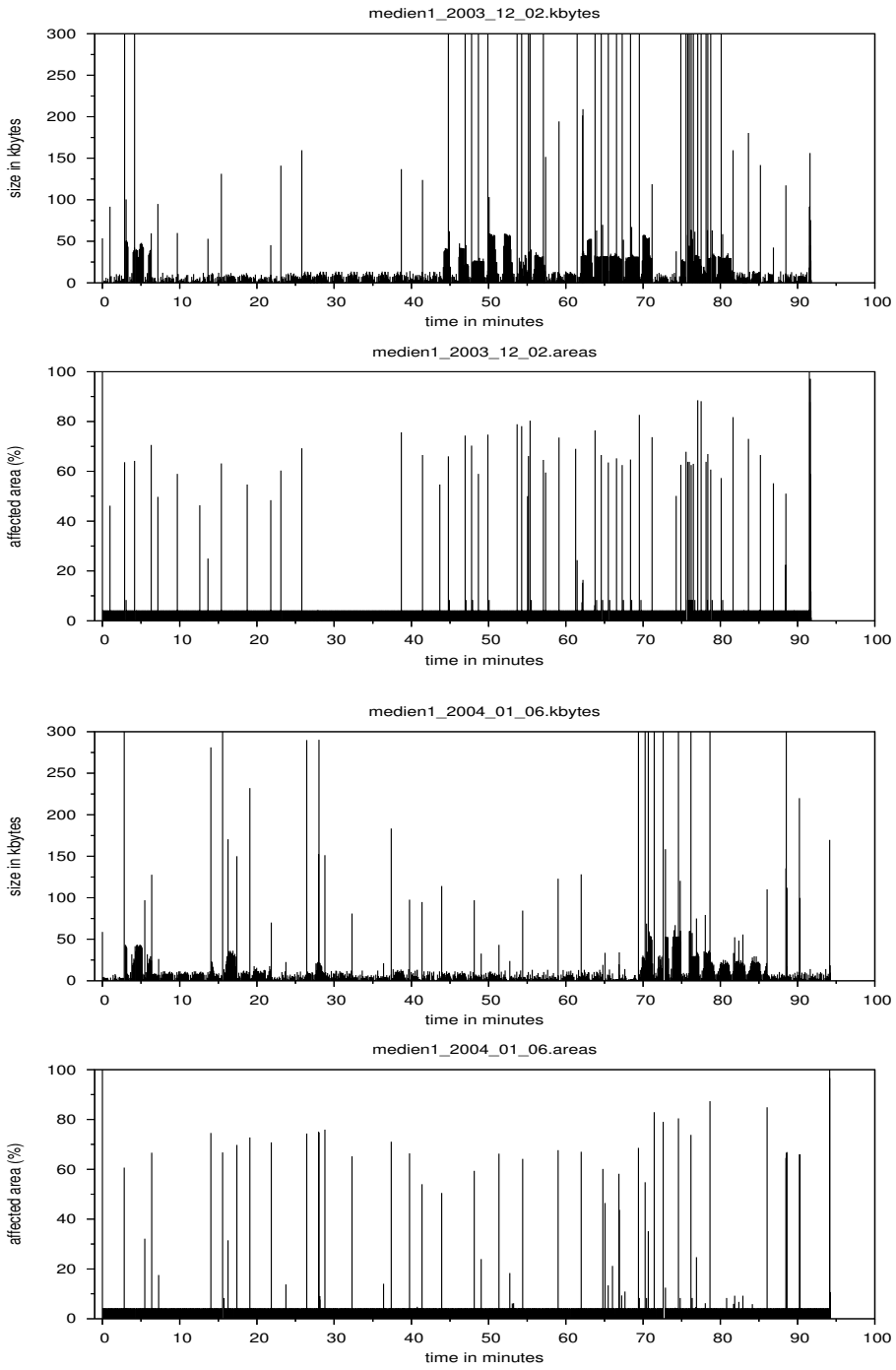
B.2 Medienwissenschaft I: Theorien und Methoden [WS2003/04]

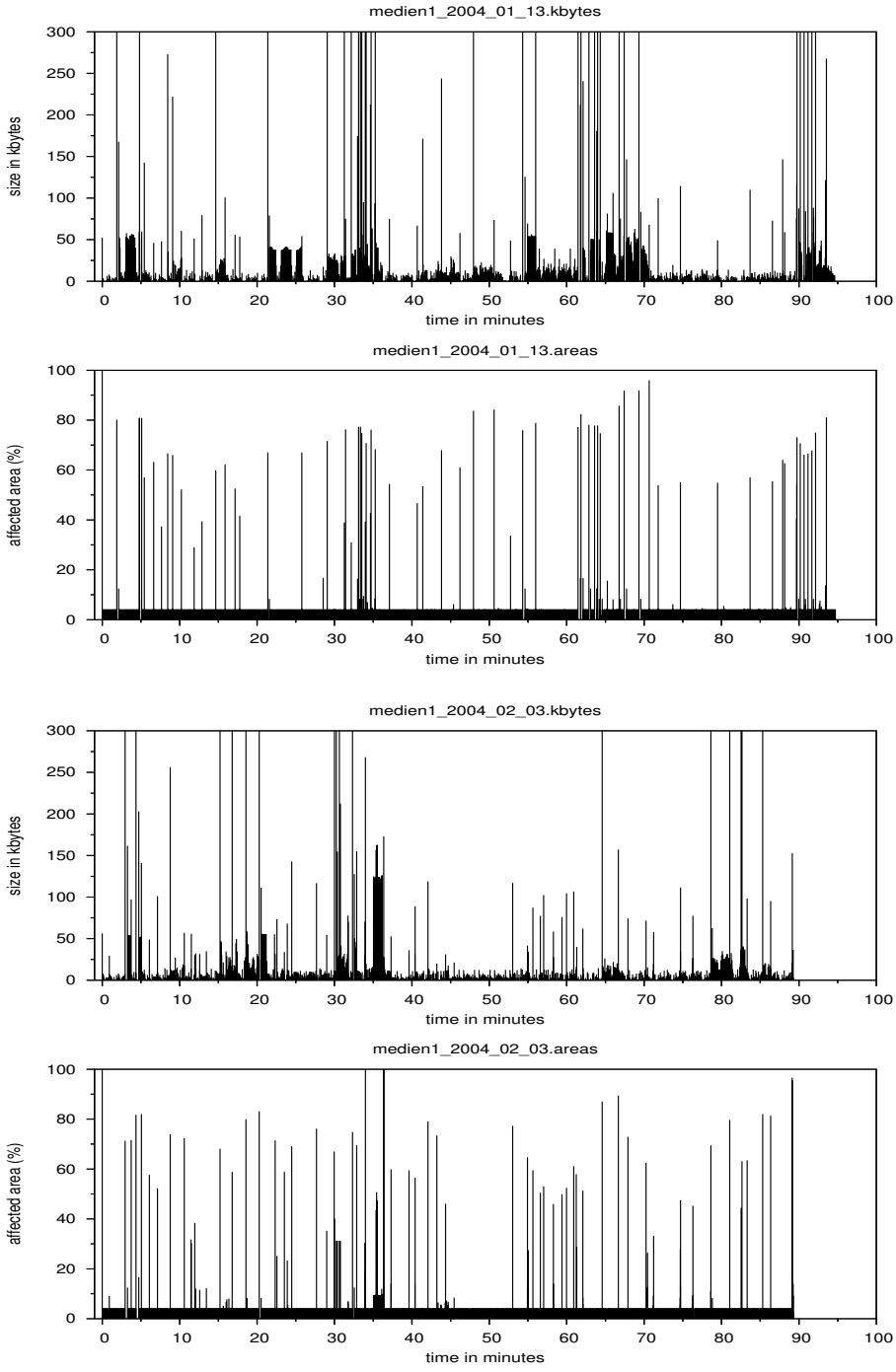
Prof. Dr. Hans-Jürgen Bucher / Universität Trier

1024 x 768 (16 bit truecolor), 16 bits per pixel, 2 bytes per pixel
with keyframe stripes




medien1_2003_11_11.kbytes
size in kbytes
time in minutes
medien1_2003_11_11.areas
affected area (%)
time in minutes
medien1_2003_11_25.kbytes
size in kbytes
time in minutes
medien1_2003_11_25.areas
affected area (%)
time in minutes

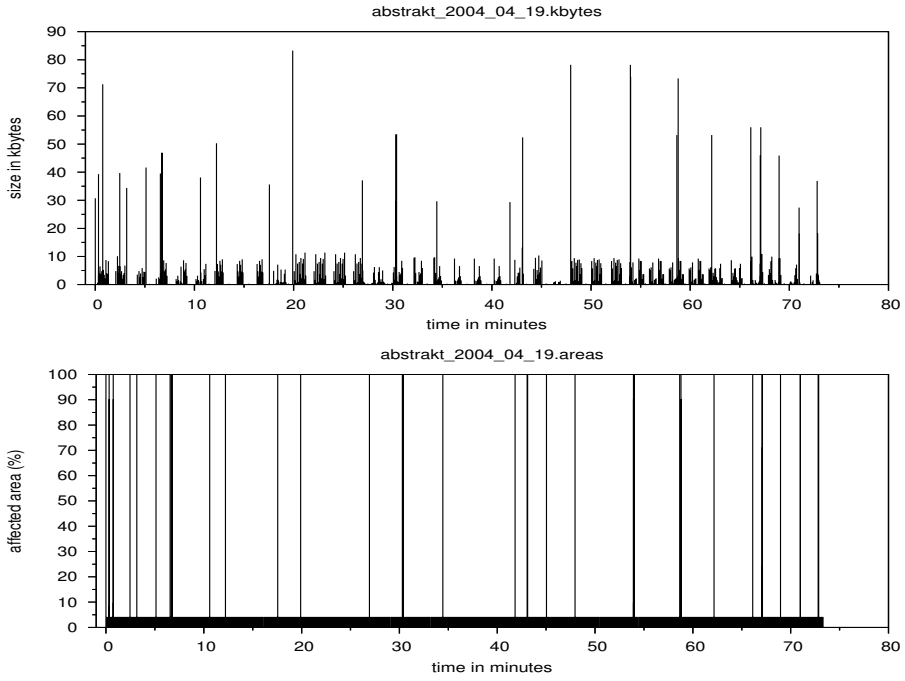

medien1_2003_12_02.kbytes
size in kbytes
time in minutes
medien1_2003_12_02.areas
affected area (%)
time in minutes
medien1_2004_01_06.kbytes
size in kbytes
time in minutes
medien1_2004_01_06.areas
affected area (%)
time in minutes

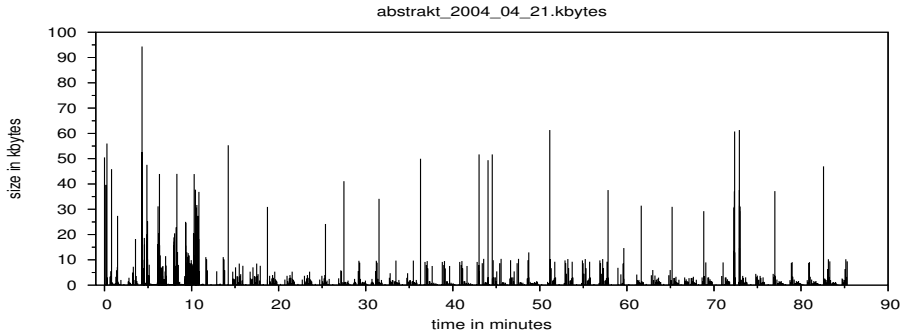

medien1_2004_01_13.kbytes
size in kbytes
time in minutes
medien1_2004_01_13.areas
affected area (%)
time in minutes
medien1_2004_02_03.kbytes
size in kbytes
time in minutes
medien1_2004_02_03.areas
affected area (%)
time in minutes

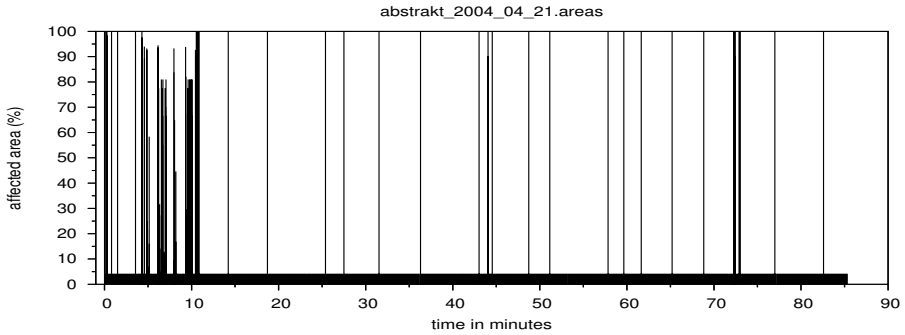
B.3 Abstrakte Maschinen im Übersetzerbau [SS2004]

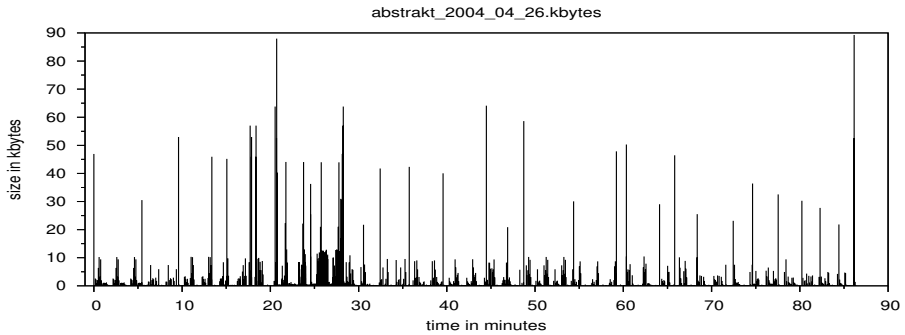
Prof. Dr. Helmut Seidl / Technische Universität München

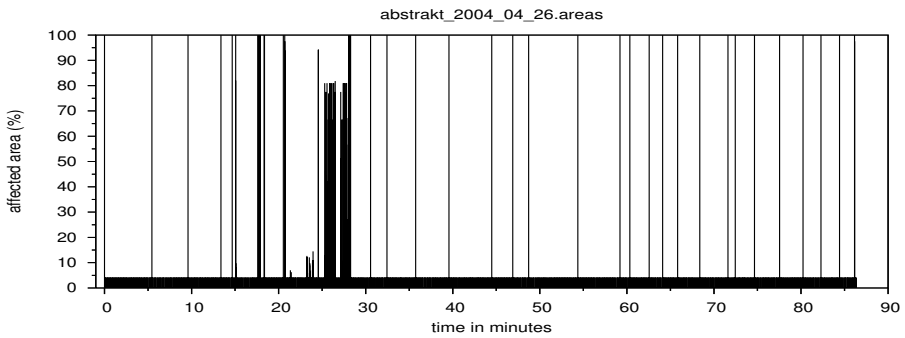
1024 x 768 (16 bit truecolor, BigEndian), 16 bits per pixel, 2 bytes per pixel with keyframe stripes

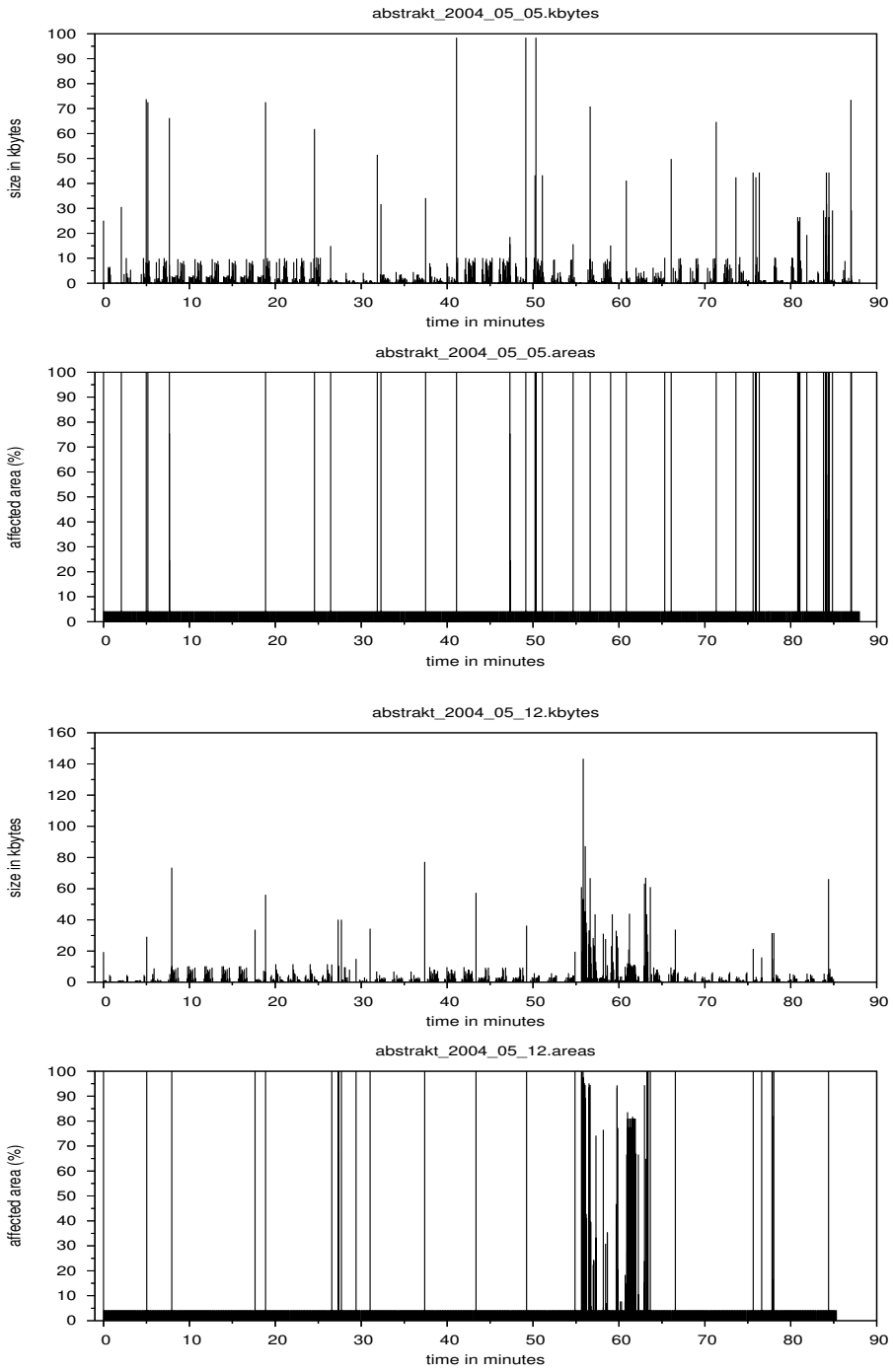



abstrakt_2004_04_21.kbytes
size in kbytes
time in minutes


abstrakt_2004_04_21.areas
affected area (%)
time in minutes


abstrakt_2004_04_26.kbytes
size in kbytes
time in minutes

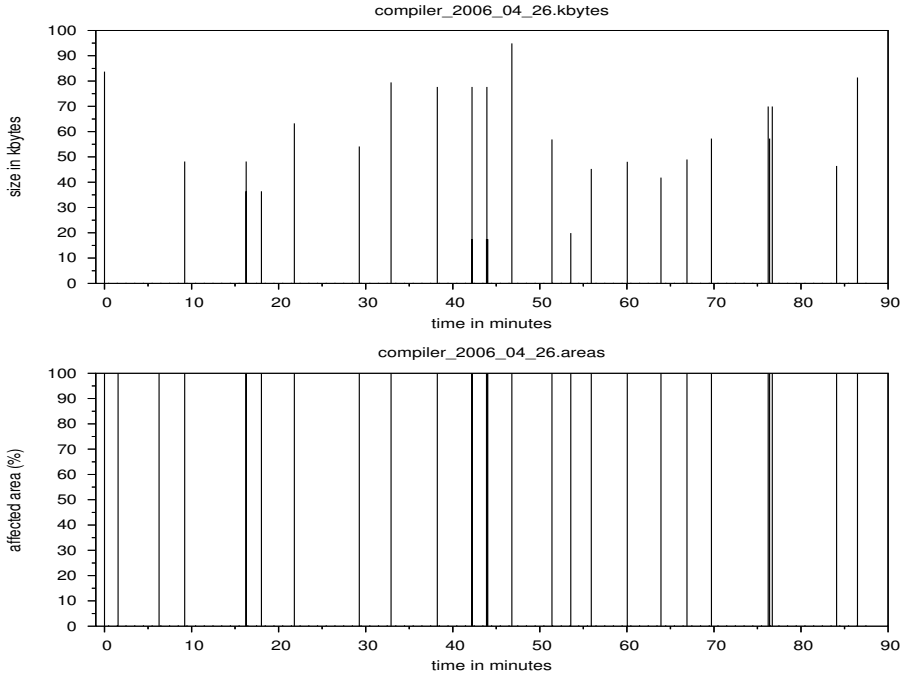

abstrakt_2004_04_26.areas
affected area (%)
time in minutes

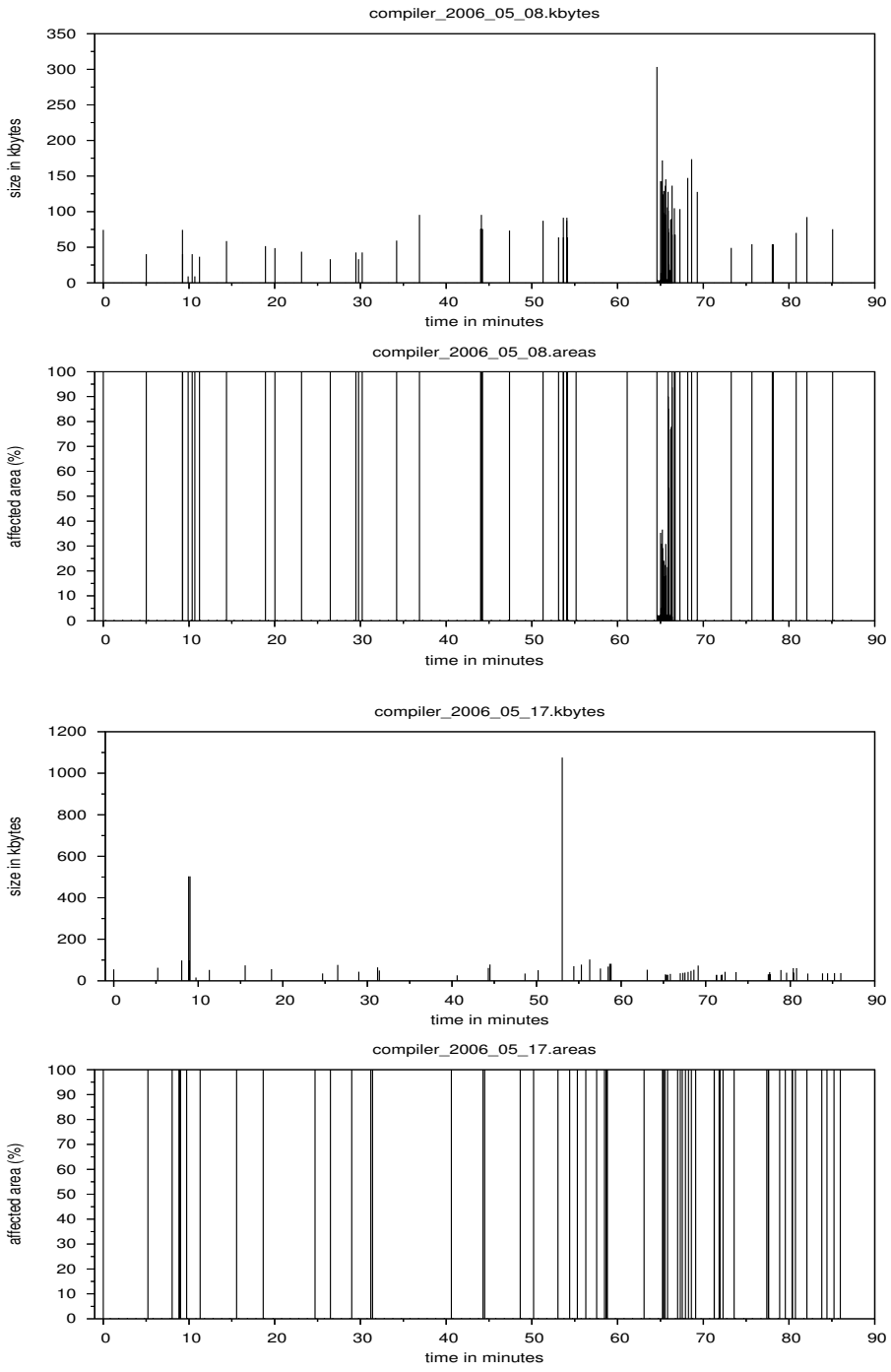

abstrakt_2004_05_05.kbytes
size in kbytes
time in minutes
abstrakt_2004_05_05.areas
affected area (%)
time in minutes
abstrakt_2004_05_12.kbytes
size in kbytes
time in minutes
abstrakt_2004_05_12.areas
affected area (%)
time in minutes

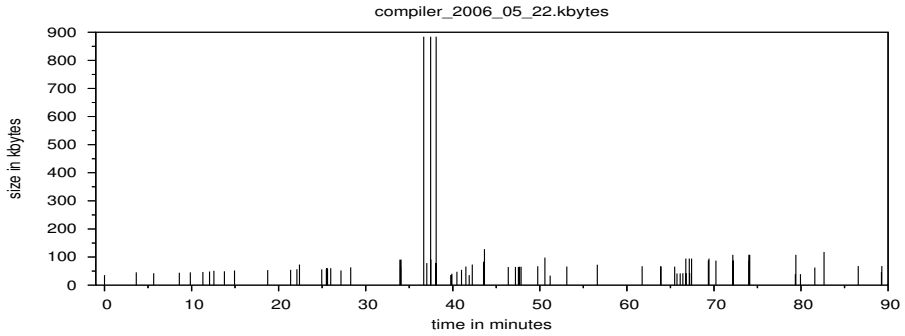
B.4 Compilerbau [SS2006]

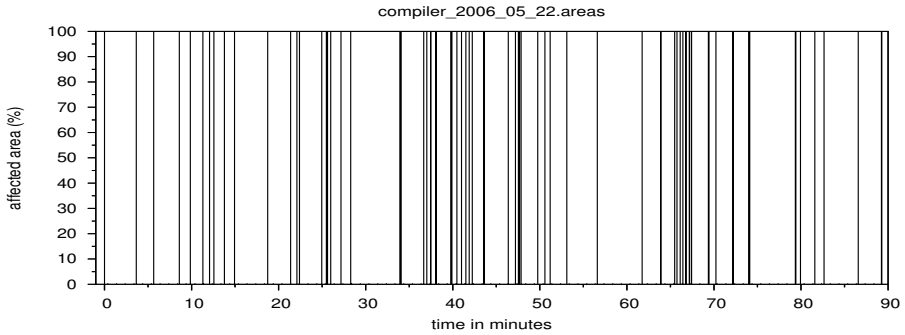
Prof. Dr. Helmut Seidl / Technische Universität München

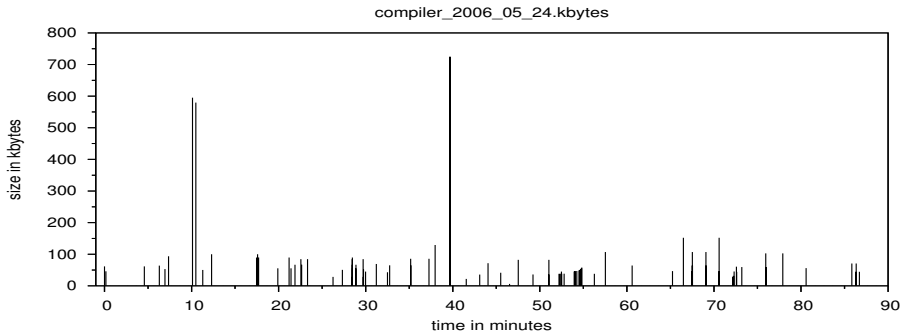
1024 x 768 (24 bit truecolor), 32 bits per pixel, 4 bytes per pixel
no keyframe stripes

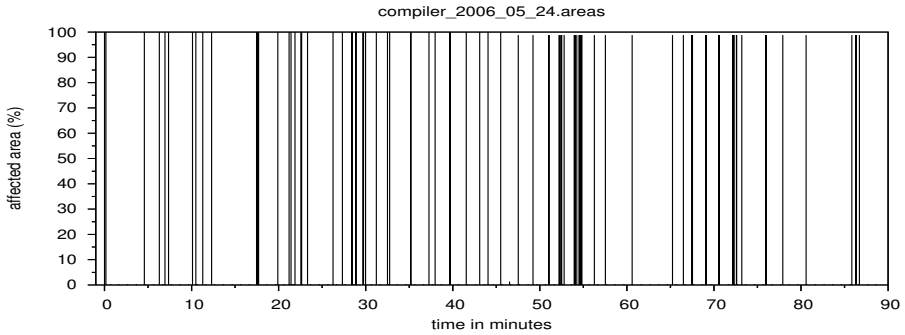


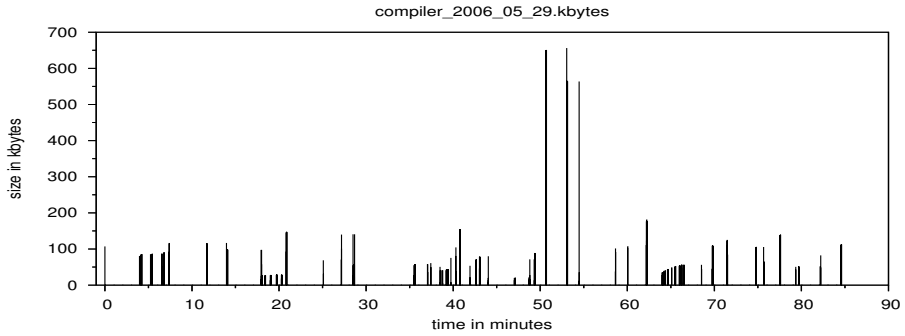

compiler_2006_05_08.kbytes
size in kbytes
time in minutes
compiler_2006_05_08.areas
affected area (%)
time in minutes
compiler_2006_05_17.kbytes
size in kbytes
time in minutes
compiler_2006_05_17.areas
affected area (%)
time in minutes

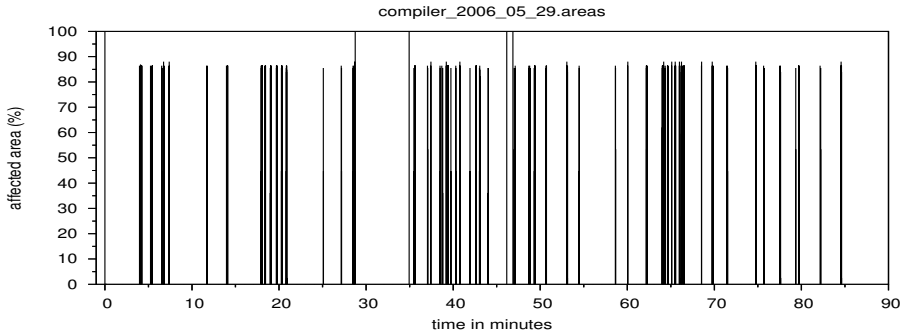

compiler_2006_05_22.kbytes
size in kbytes
time in minutes

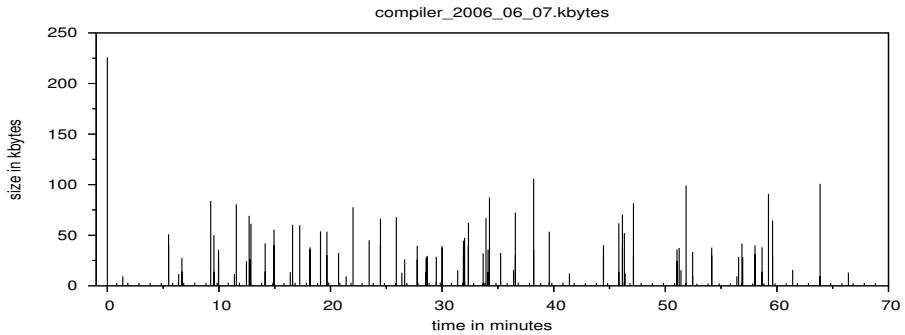

compiler_2006_05_22.areas
affected area (%)
time in minutes

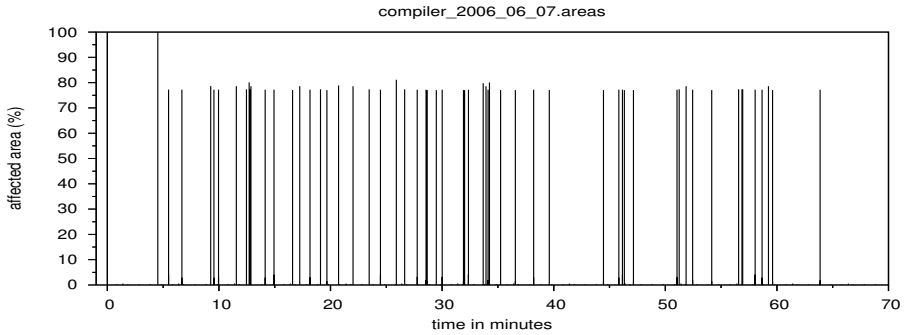

compiler_2006_05_24.kbytes
size in kbytes
time in minutes

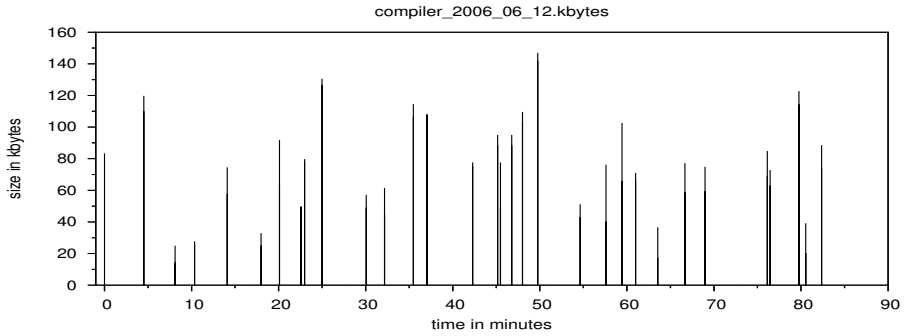

compiler_2006_05_24.areas
affected area (%)
time in minutes

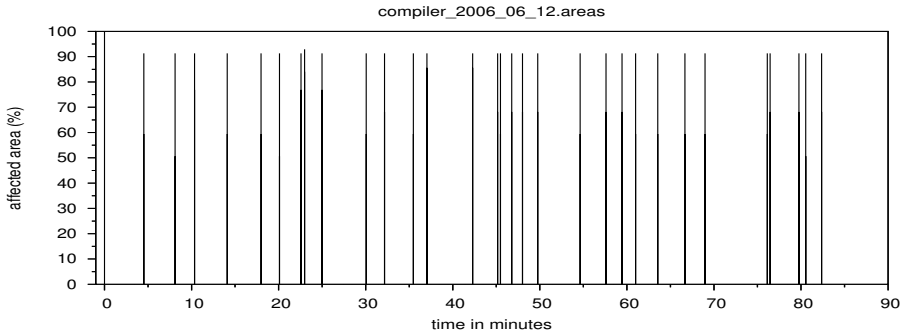

compiler_2006_05_29.kbytes
size in kbytes
time in minutes
0
100
200
300
400
500
600
700
10
20
30
40
50
60
70
80
90

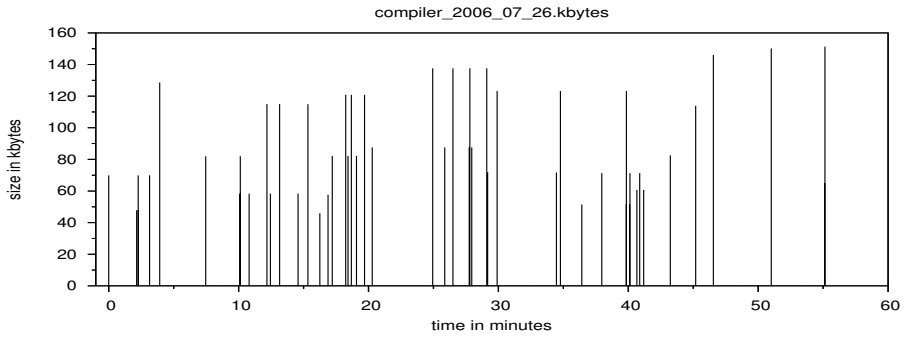

compiler_2006_05_29.areas
affected area (%)
time in minutes
0
10
20
30
40
50
60
70
80
90
100

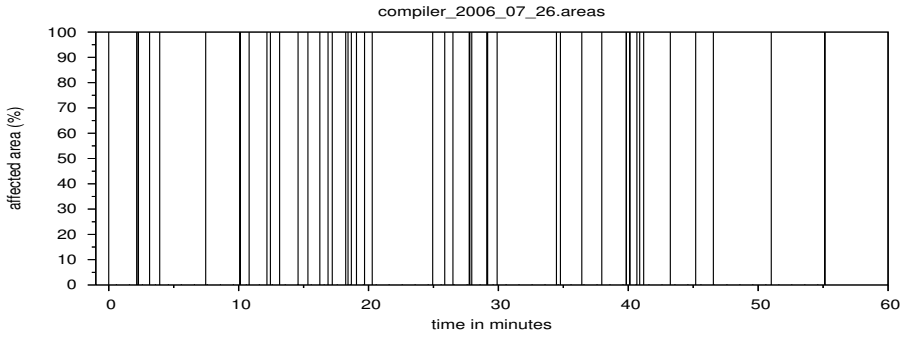

compiler_2006_06_07.kbytes
size in kbytes
time in minutes
0
50
100
150
200
250
10
20
30
40
50
60
70


compiler_2006_06_07.areas
affected area (%)
time in minutes
0
10
20
30
40
50
60
70
80
90
100


compiler_2006_06_12.kbytes
size in kbytes
time in minutes


compiler_2006_06_12.areas
affected area (%)
time in minutes


compiler_2006_07_26.kbytes
size in kbytes
time in minutes


compiler_2006_07_26.areas
affected area (%)
time in minutes

C

Publishing Script

We use the following script in order to pack and publish our electronic lectures:

```
#!/bin/bash

echo
echo "**************************"
echo "*      Recording Info    *"
echo "**************************"
echo
test -s $1.html/index.html &&
    (grep title $1.html/index.html | cut -d "<"  -f 2 | cut -b 7-)

echo
tttinfo $1

echo
echo "**************************"
echo "*          Files         *"
echo "**************************"
echo
ls -l $1*
echo
test -s $1.ttt || echo $1.ttt not found
test -s $1.ttt.orig || echo $1.ttt.orig not found
test -s $1.mp3 || echo $1.mp3 not found
test -s $1.mov || echo $1.mov not found
test -s $1.txt || echo $1.txt not found - ASCII Searchbase
test -s $1.xml || echo $1.xml not found - XML Searchbase
test -d $1.html || echo $1.html not found - HTML Script folder

echo
echo "**************************"
echo "*        HTML Script     *"
echo "**************************"
echo
test -d $1.html || echo html script missing: $1.html
```

```
test -d $1.html &&
(scp $1.html/thumbs/*.png
    ziewer@ttt.uni-trier.de:/usr/local/httpd/htdocs/search/thumbs/;
scp $1.html/images/*.png
    ziewer@ttt.uni-trier.de:/usr/local/httpd/htdocs/search/images/;
scp $1.html/html/*.html
    ziewer@ttt.uni-trier.de:/usr/local/httpd/htdocs/search/html/)

echo
echo "**************************"
echo "*        Searchbase      *"
echo "**************************"
echo
test ! -s $1.txt && test -s $1.xml &&
    java -cp /home/ziewer/workspace/TTT3/classes ttt.XMLHandler -txt $1.xml
test -s $1.txt || echo searchbase missing: $1.txt - ASCII Searchbase
test -s $1.txt && scp $1.txt
    ziewer@ttt.uni-trier.de:/usr/local/httpd/htdocs/search/searchbase/
test -s $1.xml || echo searchbase missing: $1.xml - XML Searchbase
test -s $1.xml && scp $1.xml
    ziewer@ttt.uni-trier.de:/usr/local/httpd/htdocs/search/searchbase/

echo
echo "**************************"
echo "*      Backup Original    *"
echo "**************************"
echo
echo backup original recording to server ttt.uni-trier.de
scp $1.ttt $1.mp3 $1.mov ziewer@ttt.uni-trier.de:/data/ttt/
test -s $1.ttt.orig && scp $1.ttt.orig ziewer@ttt.uni-trier.de:/data/ttt/
test -d $1.orig && scp -r $1.orig ziewer@ttt.uni-trier.de:/data/ttt/

echo
echo "**************************"
echo "*       Zip Archives      *"
echo "**************************"
echo
echo zip $1.zip
test -e $1.zip && rm -f $1.zip
zip $1.zip $1.ttt $1.mp3
echo zip $1_v.zip
test -e $1_v.zip && rm -f $1_v.zip
zip $1_v.zip $1.ttt $1.mp3 $1.mov
echo
ls -hl $1.zip
ls -hl $1_v.zip

echo
echo copy zip to server ttt.uni-trier.de
scp $1.zip $1_v.zip ziewer@ttt.uni-trier.de:/data/webspace
ls -hl $1*zip
```

```
echo
echo "*************************"
echo "*        Finished         *"
echo "*************************"
echo


echo
echo "*************************"
echo "*      Recording Check    *"
echo "*************************"
echo
echo "Missing files:"
test -s $1.ttt || echo $1.ttt not found
test -s $1.ttt.orig || echo $1.ttt.orig not found
test -s $1.mp3 || echo $1.mp3 not found
test -s $1.mov || echo $1.mov not found
test -s $1.txt || echo $1.txt not found - ASCII Searchbase
test -s $1.xml || echo $1.xml not found - XML Searchbase
test -d $1.html || echo $1.html not found - HTML Script folder

echo
echo "Titel ok?"
test -s $1.html/index.html &&
    (grep title $1.html/index.html | cut -d "<"  -f 2 | cut -b 7-)
echo
echo "Download Sizes:"
ls -hl $1.zip
ls -hl $1_v.zip
echo
echo "Length of recoding:"
tttinfo $1
echo
echo "Remember to edit download page:"
echo "  ziewer@ttt.uni-trier.de:/usr/local/httpd/htdocs/recordings/*.html"
echo
```

Note that `ttt.info` lists the durations of the audio,video and desktop streams.

References

[Abowd et al., 1998] Abowd, G. D., Atkeson, C. G., Brotherton, J. A., Enqvist, T., Gulley, P., and LeMon, J. (1998). Investigating the Capture, Integration and Access Problem of Ubiquitous Computing in an Educational Setting. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI 1998*, pages 440–447. ACM Press/Addison-Wesley Publishing Co.

[AdderLink, 2006] Adder Technology (2006). AdderLink IP product site. http://www.adder.com/.

[AOF, 2006] AOF Team (2006). Authoring on the Fly product site. http://ad.informatik.uni-freiburg.de/aof/index.html.

[QTJava, 2006] Apple Computer, Inc. (2006). QuickTime for Java product site. http://developer.apple.com/quicktime/qtjava/.

[Bacher et al., 1997] Bacher, C., Müller, R., Ottmann, T., and Will, M. (1997). Authoring on the Fly: a new way of integrating telepresentation and courseware production. In *Proceedings of the International Conference of Computers in Education (ICCE'97)*, pages 89–96, Kuching, Sarawak, Malaysia.

[Bankhead, 2005] Bankhead, P. (2005). The Design and Implementation of an Editor for the TeleTeachingTool Environment. Master thesis, School of Computer Science, The Queen's University of Belfast and Technische Universität München.

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

[Brusilovsky and Miller, 2000] Brusilovsky, P. and Miller, P. (2000). Course Delivery Systems for the Virtual University. In Tschang, T. and Senta, T. D., editors, *Access to Knowledge: New Information Technologies and the Emergence of the Virtual University*, pages 167–206. Elsevier Science, Amsterdam, The Netherlands.

[Windows Media, 2006] Corporation, M. (2006). Windows Media product site. http://www.microsoft.com/windows/windowsmedia/.

[Damerau, 1964] Damerau, F. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176.

[Deering and Hinden, 1998] Deering, S. and Hinden, R. (1998). RFC 2460: Internet Protocol, Version 6 (IPv6) Specification. ftp://ftp.rfc-editor.org/in-notes/rfc2460.txt.

[Deutsch and Gailly, 1996] Deutsch, P. and Gailly, J.-L. (1996). RFC 1950: ZLIB Compressed Data Format Specification version 3.3. ftp://ftp.rfc-editor.org/in-notes/rfc1950.txt.

[Ecma, 1999] Ecma International (1999). Standard ECMA-262: ECMAScript Language Specification. http://www.ecma-international.org/publications/standards/Ecma-262.htm.

[Edelmann, 1995] Edelmann, W. (1995). *Lernpsychologie*. Psychologie-Verlags-Union, Weinheim.

[Effelsberg and Geyer, 1998] Effelsberg, W. and Geyer, W. (1998). Tools for Digital Lecturing - What We Have and What We Need. In *Proceedings of BITE '98 (Bringing Information Technology to Education)*, pages 151–173, Maastricht, The Netherlands. Kluwer Verlag.

[Garofolo et al., 2000] Garofolo, J., Auzanne, G., and Voorhees, E. (2000). The trec spoken document retrieval track: A success story.

[Gruber and Leiter, 2006] Gruber, J. and Leiter, U. (2006). Erweiterung der Suchfunktion des TeleTeachingTool: Das Search&BrowsingTool. Systementwicklungsprojekt, Technische Universität München.

[Hauptmann and Wactlar, 1997] Hauptmann, A. and Wactlar, H. (1997). Indexing and search of multimodal information.

[Hilt et al., 2001] Hilt, V., Mauve, M., Vogel, J., and Effelsberg, W. (2001). Interactive media on demand: Generic recording and replay of interactive media streams. In *ACM Multimedia 2001, Association of Computing Machinery*, pages 593–594, Ottawa, Canada.

[Hogrefe et al., 2003] Hogrefe, D., Köster, R., Werner, C., and Zibull, M. (2003). Teleteaching an der universität göttingen: Systemarchitektur und problematiken. In *DeLFI*, pages 129–133.

[Finereader, 2006] House, A. S. (2006). Finereader OCR product site. http://www.abbyy.com/finereader_ocr/.

[Hunter and Steiglitz, 1979] Hunter, G. M. and Steiglitz, K. (1979). Operations on Images Using Quad Trees. *IEEE Transaction on Pattern Analysis and Machine Intelligence (PAMI)*, 1(2):145–153.

[Hürst, 2003] Hürst, W. (2003). Suche in aufgezeichneten Vorträgen und Vorlesungen. In Bode, A., Desel, J., Rathmeyer, S., and Wessner, M., editors, *Tagungsband: Die 1. e-Learning Fachtagung Informatik (DeLFI 2003)*, volume 37 of *LNI*, pages 27–36, Garching bei München. GI-Edition Lecture Notes in Informatics.

[Hürst et al., 2003] Hürst, W., Kreuzer, T., and Wiesenhütter, M. (2003). A Qualitative Study Towards Using Large Vocabulary Automatic Speech Recognition to Index Recorded Presentations for Search and Access over the Web. *IADIS International Journal on WWW/Internet*, I(1):43–58.

[CLIX, 2006] imc AG (2006). CLIX Campus product site. http://www.clix.de/.

[Lecturnity, 2006] imc AG (2006). Lecturnity product site. http://www.lecturnity.de/.

[Flash, 2006] Incorporated, A. S. (2006). Adobe Flash (Macromedia Flash) product site. http://www.adobe.com/de/products/flash/.

[DublinCore, 2003] Initiative, D. C. M. (2003). Dublin Core Metadata Element Set, Version 1.1. ISO Standard 15836-2003.

[DublinCoreVocabulary, 2004] Initiative, D. C. M. (2004). DCMI Type Vocabulary. http://dublincore.org/documents/dcmi-typevocabulary/.

[DCMI, 2006] Initiative, D. C. M. (2006). Dublin Core Metadata Initiative web page. http://dublincore.org/.

[Ishii and Miyake, 1991] Ishii, H. and Miyake, N. (1991). Toward an open shared workspace: Computer and video fusion approach of teamworkstation. *Commun. ACM*, 34(12):36–50.

[Jackson et al., 2000] Jackson, J. R., Anderson, D. V., and Hayes III, M. H. (2000). Effective and Efficient Distance Learning Over the Internet: Tools and Techniques. In *Proceedings of the International Conference on Engineering Education (ICEE 2000)*, Taipei, Taiwan.

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

[Kandzia and Maass, 2001] Kandzia, P.-T. and Maass, G. (2001). Course Production - Quick and Effective. In *Proceedings of the 3rd International Conference on New Learning Technologies (NLT 2001)*, Fribourg, Switzerland.

[TightVNC, 2006] Kaplinsky, C. (2006). TightVNC product site. http://www.tightvnc.com/.

[Knopper, 2006] Knopper, K. (2006). KNOPPIX product site. http://www.knopper.net/knoppix/.

[LaRose et al., 1997] LaRose, R., Gregg, J., and Heeter, C. (1997). An evaluation of a Web-based distributed learning environment in higher education. In *Proceedings of ED-MEDIA/ED-TELECOM'97 - World Conference on Educational Multimedia / Hypermedia and World Conference on Educational Telecommunications*, pages 1286–1287, Calgary, Canada. AACE Press.

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

[Levenshtein, 1966] Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

[Li and Hopper, 1998a] Li, S. F. and Hopper, A. (1998a). A Framework to Integrate Synchronous and Asynchronous Collaboration. In *Proceedings of the 7th IEEE Workshops on Enablings Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 96–103, Stanford, CA. IEEE Computer Society Press.

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

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

[Li et al., 1999a] Li, S. F., Stafford-Fraser, Q., and Hopper, A. (1999a). Applications of Stateless Client Systems in Collaborative Enterprises. In *International Conference on Enterprise Information Systems*, pages 665–673.

[Li et al., 1999b] Li, S. F., Stafford-Fraser, Q., and Hopper, A. (1999b). Frame-buffer on Demand: Applications of Stateless Client Systems in Web-based Learning. In *Proceedings of the 5th International Conference on Information Systems Analysis and Synthesis (ISAS'99)*, Orlando, Florida.

[Li et al., 2000b] Li, S. F., Stafford-Fraser, Q., and Hopper, A. (2000b). Integrating Synchronous and Asynchronous Collaboration with Virtual Network Computing. In *Proceedings of the First International Workshop on Intelligent Multimedia Computing and Networking*, volume 2, pages 717–721, Atlantic City, New Jersey.

[Lienhard and Lauer, 2002] Lienhard, J. and Lauer, T. (2002). Multi-layer recording as a new concept of combining lecture recording and students' handwritten notes. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 335–338, New York, NY, USA. ACM Press.

[Lienhard and Zupancic, 2003] Lienhard, J. and Zupancic, B. (2003). Annotieren von vorlesungsaufzeichnungen während der aufnahme- und wiedergabe-phase. In *DeLFI*, pages 95–99.

[ULI, 2006] Management, U. P. (2006). Universitärer Lehrverbund Informatik (ULI) project site. http://www.uli-campus.de/.

[Mauve et al., 2001] Mauve, M., Hilt, V., Kuhmünch, C., and Effelsberg, W. (2001). Rtp/i - towards a common application level protocol for distributed interactive media. In *IEEE Transactions on Multimedia (TMM'01)*, volume 3, pages 152–161.

[Mertens and Rolf, 2003] Mertens, R. and Rolf, R. (2003). Automation Techniques for Broadcasting and Recording Lectures and Seminars. In *Proceedings of SINN03 - Third International Technical Workshop and Conference*.

[Minneman et al., 1995] 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. In *Proceedings of the The Third ACM International Multimedia Conference and Exhibition (ACM MULTIMEDIA '95)*, pages 523–534, San Francisco, CA. ACM Press.

[Mitchell et al., 1996] Mitchell, J. L., Pennebaker, W. B., Fogg, C. E., and Legall, D. J., editors (1996). *MPEG Video Compression Standard*. Chapman & Hall, Ltd., London, UK, UK.

[Muehlhaeuser and Trompler, 2002] Muehlhaeuser and Trompler (2002). Digital lecture halls...

[Nopoudem, 2006] Nopoudem, E. W. T. (2006). Transformation von TeleTeachingTool-Aufzeichnungen zur Wiedergabe mit Standard-Software. Diplomarbeit, Technische Universität München.

[OmniPage, 2006] Nuance Communications, I. f. S. (2006). OmniPage product site. http://www.nuance.com/omnipage/.

[FMJ, 2006] open source community (2006). Freedom for Media in Java (FMJ) product site. http://fmj.sourceforge.net/.

[OSXvnc, 2006] OSXvnc (2006). OSXvnc product site. http://sourceforge.net/projects/osxvnc/.

[Postel, 1980] Postel, J. (1980). RFC 768: User Datagram Protocol. ftp://ftp.rfc-editor.org/in-notes/rfc768.txt.

[Postel, 1981a] Postel, J. (1981a). RFC 791: Internet Protocol. ftp://ftp.rfc-editor.org/in-notes/rfc791.txt.

[Postel, 1981b] Postel, J. (1981b). RFC 793: Transmission Control Protocol. ftp://ftp.rfc-editor.org/in-notes/rfc793.txt.

[RealMedia, 2006] RealNetworks, I. (2006). RealMedia product site. http://www.realnetworks.com/products/codecs/.

[RealVNC, 2006] RealVNC Ltd (2006). RealVNC product site. http://www.realvnc.com/.

[Richardson, 2005] Richardson, T. (2005). The RFB Protocol, Version 3.8. http://realvnc.com/docs/rfbproto.pdf. RealVNC Ltd.

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

[Samet, 1980] Samet, H. (1980). Region Representation: Quadtree from Binary Arrays. *Computer Graphics & Image Processing*, 13(1):88–93.

[Schillings and Meinel, 2002] Schillings, V. and Meinel, C. (2002). tele-TASK - Teleteaching Anywhere Solution Kit. In *Proceedings of the ACM SIGUCCS 2002*, pages 130–133, Providence, Rhode Island, USA.

[UltraVNC, 2006] Schneider, O., Scharpf, M., Vos, R. D., and UltraSam (2006). UltraVNC product site. http://www.ultravnc.com/.

[Schulzrinne et al., 2003] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V. (2003). RFC 3550: RTP: A Transport Protocol for Real-Time Applications. ftp://ftp.rfc-editor.org/in-notes/rfc3550.txt.

[Schütz, 2002] Schütz, F. (2002). Annotations: Though standard in conventional learning a stepchild of elearning. In Vilas, A. M. and Gonzalez, J. M., editors, *Proceedings of the International Conference on Information and Communication Techonologies in Education (ICTE2002)*, Badajoz, Spain.

[Schütz, 2003] Schütz, F. (2003). Producing eLearning materials on the fly – only a great dream? In Vilas, A. M. and Gonzalez, J. M., editors, *Proceedings of the Second International Conference on Multimedia and ICTs in Education (m-ICTE2003)*, Badajoz, Spain.

[Schütz, 2005] Schütz, F. (2005). *Annotationen in der Lehre – Eine Annotationsarchitektur zur Erweiterung bestehender elektronischer Lehrsysteme.* Dissertation, Technische Universität München.

[Smartboard, 2006] SMART Technologies (2006). Smartboard product site. http://www.smartboard.com/.

[Spiteri and Bates, 1998] Spiteri, M. D. and Bates, J. (1998). An architecture to support storage and retrieval of events. In *Proceedings of MIDDLEWARE 1998, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Lancaster, UK.

[RAT, 2006] SUMOVER Project, University College London (2006). Robust Audio Tool (RAT) product site. http://mediatools.cs.ucl.ac.uk/nets/mmedia/wiki/RatWiki.

[VIC, 2006] SUMOVER Project, University College London (2006). Videoconferencing Tool (VIC) product site. http://mediatools.cs.ucl.ac.uk/nets/mmedia/wiki/VicWiki.

[JMF, 2006] Sun Microsystems, Inc. (2006). Java Media Framework (JMF) product site. http://java.sun.com/products/java-media/jmf/.

[Camtasia, 2006] TechSmith Corporation (2006). Camtasia Studio product site. http://www.techsmith.com/camtasia.asp.

[TSCC, 2006] TechSmith Corporation (2006). TechSmith Screen Capture Codec product site. http://www.techsmith.com/codecs/tscc/default.asp.

[Targeteam, 2006] Teege, G. (2006). Targeteam product site. http://www11.in.tum.de/forschung/projekte/targeteam/.

[Teege and Breitling, 2002] Teege, G. and Breitling, P. (2002). Targeteam: Adaptierbare Lehrinhalt auf Basis on XML und XSLT. In *Informatik bewegt: Informatik 2002 - 32. Jahrestagung der Gesellschaft für Informatik e.v. (GI)*, pages 364–368. Gesellschaft für Informatik e.v. (GI).

[teleTASK, 2006] teleTASK (2006). tele-TASK product site. http://www.tele-task.com/.

[Thong et al., 2000] Thong, J. V., Goddeau, D., Litvinova, A., Logan, B., Moreno, P., and Swain, M. (2000). Speechbot: A speech recognition based audio indexing system for the web.

[Wacom, 2006] Wacom Technology (2006). Wacom product site. http://www.wacom.com/.

[Wayne Hodgins, 2002] Wayne Hodgins, Erik Duval, e. (2002). Draft standard for learning object metadata, ieee 1484.12.1-2002. Technical report, IEEE Learning Technology Standards Committee.

[Ziewer, 2001] Ziewer, P. (2001). Visualisierung Abstrakter Maschinen. Diplomarbeit, Universität Trier.

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

[VAM, 2006] Ziewer, P. (2006). VAM Simulator product site. http://wwwseidl.in.tum.de/projekte/vam/.

[Ziewer and Seidl, 2002] Ziewer, P. and Seidl, H. (2002). Transparent Teleteaching. In Williamson, A., Gunn, C., Young, A., and Clear, T., editors, *Proceedings of the 19th Annual Conference of the Australian Society for Computers in Tertiary Education (ASCILITE)*, volume 2, pages 749–758, Auckland, New Zealand. UNITEC Institute of Technology.

[Ziewer and Seidl, 2004] Ziewer, P. and Seidl, H. (2004). Annotiertes Lecture Recording. In Engels, G. and Seehusen, S., editors, *Tagungsband: Die 2. e-Learning Fachtagung Informatik (DeLFI 2004)*, volume 52 of *LNI*, pages 43–54, Paderborn, Germany. Gesellschaft für Informatik e.V (GI.

[Zupancic and Horz, 2002] Zupancic, B. and Horz, H. (2002). Lecture recording and its use in a traditional university course. In *ITiCSE '02: Proceedings of the 7th annual*

conference on Innovation and technology in computer science education, pages 24–28, New York, NY, USA. ACM Press.