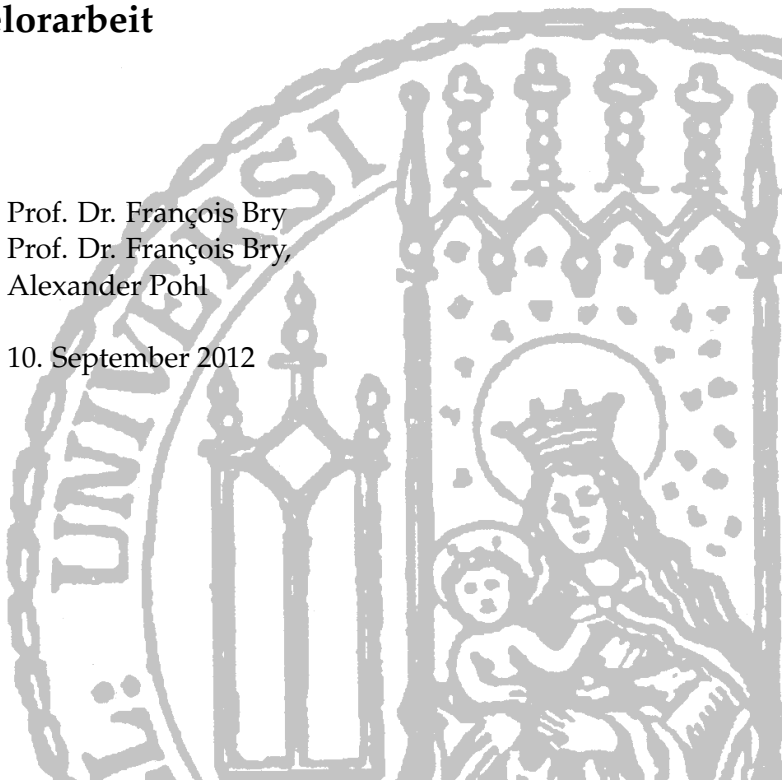
INSTITUT FÜR INFORMATIK
der Ludwig-Maximilians-Universität München

MEDIA STREAMING IN BACKSTAGE

Improving awareness of the audience outside the classroom

Florian Naß

Bachelorarbeit



Aufgabensteller	Prof. Dr. François Bry
Betreuer	Prof. Dr. François Bry, Alexander Pohl
Abgabe am	10. September 2012

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, den 10. September 2012 Florian Naß

Abstract

The digital backchannel known as Backstage, promotes active participation and collaboration of students in lectures. Introducing audiovisual streaming capability to Backstage not only enhances the awareness of co-located and non co-located students, but also broadens the spectrum of use cases beyond the initial concept of Backstage. The aim of this thesis is to develop a prototypical media streaming system and integrate it into Backstage. During this process, the system architecture is designed on a conceptual level. Five different technologies are analysed and compared against one another in order to find suitable software components for the creation of a prototype, which enables the broadcast, consisting of multiple perspectives, to create optimal teaching and learning environments. In addition to real-time broadcast, media streams are reserved on the media server in order to be made available on demand at a later point in time. Further, this thesis gives evidence that the storage of cue points along with the media streams is imperative to provide flawless synchronisation between streams and presentation slides.

Acknowledgements

I would like to express my gratitude to Prof. Dr. François Bry for offering me the opportunity to research and work on this exciting and versatile subject. Further, I wish to thank my personal supervisor, Alexander Pohl, for his support and imaginative contributions. Special thanks also to my family for their limitless moral and generous financial support and to Jodie, my partner, who spent copious amounts of time proof-reading drafts of this thesis.

Zusammenfassung

Der digitale Backchannel, Backstage, unterstützt die Zusammenarbeit und aktive Teilnahme der Studenten an Vorlesungen. Die Einführung von audiovisuellem Streaming steigert nicht nur die Awareness der im Vorlesungssaal anwesenden Studenten sowie derjenigen, die der Vorlesung von außerhalb folgen, sondern erweitert das Spektrum an Anwendungsfällen über das ursprüngliche Konzept von Backstage hinaus. Das Ziel dieser Arbeit ist ein prototypisches Mediastreamingsystem zu entwickeln und es in Backstage zu integrieren. Während dieses Prozesses wird die Systemarchitektur auf konzeptueller Ebene entworfen. Fünf verschiedene Technologien werden analysiert und miteinander verglichen, um geeignete Softwarekomponenten für die Erstellung eines Prototypen, der die Übertragung von mehreren verschiedenen Kameraperspektiven und Blickwinkeln ermöglicht, zu finden, so dass optimale Lehr- und Lernumgebungen geschaffen werden können. Zusätzlich zu Echtzeitübertragung werden die Mediastreams auf dem Mediaserver gespeichert, um zu einem späteren Zeitpunkt per Abruf zur Verfügung gestellt zu werden. Desweiteren ist die Speicherung von Cue Points zusammen mit den Mediastreams unerlässlich, um eine lückenlose Synchronisation zwischen Streams und Vortragsfolien zu gewähren.

Contents

1 Introduction **1**
- 1.1 Task description . . . 1
- 1.2 Media streaming . . . 2
- 1.3 Thesis overview . . . 4
- 1.4 Sample scenario . . . 4

2 Conception **7**
- 2.1 Technology-independent architecture . . . 7
- 2.2 Use cases and concepts . . . 9
  - 2.2.1 General use case . . . 9
  - 2.2.2 Combining roles . . . 9
  - 2.2.3 Simulating real-time lectures . . . 9
  - 2.2.4 Providing comparable lectures . . . 10
  - 2.2.5 Statistical analysis . . . 10
  - 2.2.6 Bi-directional awareness . . . 11
- 2.3 Integration into Backstage . . . 11
  - 2.3.1 Structure . . . 11
  - 2.3.2 Adjustments to Backstage . . . 12

3 Technology selection criteria **15**

4 Technology review **17**
- 4.1 Considering HTML5 for streaming . . . 17
- 4.2 Flash and Red5 Media Server . . . 18
  - 4.2.1 Adobe Flash . . . 18
  - 4.2.2 Red5 Media Server . . . 18
- 4.3 Darwin Streaming Server . . . 19
- 4.4 VideoLan and the VLC media player . . . 20
- 4.5 Java Media Framework . . . 20
- 4.6 Technology selection . . . 21

5 Implementing the system **25**
- 5.1 Stream management . . . 25

5.2 Audience 28
5.3 Integration into Backstage 29

6 Future work **31**
6.1 Synchronisation 31
6.2 Handheld devices 32
6.3 Usability 32
6.4 Bi-directional awareness 33
6.5 Technical improvement 33

7 Conclusion **35**

Appendix **37**
A.1 VLC test runs 37
A.2 Implementation draft using HTML5 37
The client for broadcast 37
The server 40
The listener client 43
A.3 Code snippets for flash development 44
A.4 Full implementation using Adobe Flash 45
Stream management 45
Shared artefact 51
Presenter 56
A.5 Development of Flash applications using Adobe Flex 59

List of Figures **61**

Bibliography **63**

CHAPTER 1

Introduction

1.1 Task description

"Backstage is a digital backchannel dedicated to large classes aiming at empowering not only the audience but also the speaker, at promoting the awareness of both audience and speaker, and at promoting active participation of students in the lecture. The backchannel supports different forms of inter-student communication via short microblog messages, social evaluation and ranking of messages by the students themselves" [1].
Currently, the backchannel is designed for students located inside the lecture hall. Backstage, as a communication medium, offers social presence which is appropriate and sufficient for co-located participants only. However, full social presence of the lecturer is not available to students who take part in lectures from outside the lecture hall. Reasons for physical absence include lecture buildings which are far apart from each other, thus preventing students from arriving to the lecture in time, inconvenient appointments and traffic-related delays. In the case of absence, presentation slides alone are usually not satisfactory in order to follow a lecture or prepare for exams. Further, even co-located students may find audiovisual tranmission necessary, for example to bypass noise or bridge the distance between the audience's seats and an unproportional small board.
The objective of this project is to enhance awareness for these students by transmitting the audiovisual social presence of the presenter. For this purpose, live streaming of audio and video from the lecture hall to students who are not co-located has to be implemented. Alternatively, the lecturer should be able to select a prerecorded version and stream it to the media server in order to simulate a real-time lecture. The task of the server then is to distribute the stream to remote students on the one hand, and to store the media for later access on demand on the other.

Regarding lectures or conference presentations, several different camera viewpoints are needed in order to fully promote the lecturer's social presence as well as transmit visual content of ongoing events.
Lecturers managing the capturing of media by themselves simultaneously have to deal with large audiences, thus the front-end user interface for lecturers must not consist of a complex layout showing many options and settings, but has to be very simple, basic and user friendly. Personnel who are exclusively dedicated to transmit the media stream may have a better overview of the lecture hall and are able to direct the broadcast appropriatly, thus they are provided with a seperate user interface, possibly a standalone application, that contains more detailed controls.
In addition, one has to be aware of certain peculiarities of university lectures or conference presentations when implementing a media streaming system for audio and video. For example, the presenter may give information which is not to be recorded or stored on the server for public access via replay. Hence, the interface has to be designed accordingly, making it possible to pause either element of the stream, be it audio or video.
The server will distribute the received live media stream to the remote audience's clients and store the extracted audio and video for later access on demand.
Both the media stream and the presentation slides, along with the visualisation of the interactions in Backstage, will have to be displayed on the remote students' interface. Therefore, the video has to be placed within the current layout of Backstage, so-called "dashboards", in order to add awareness about the lecturer appropriately.
The goal of this thesis is thus to provide a prototypical implementation of media streaming which should be operable separately from Backstage as a distinct system on the one hand. On the other hand, this system should offer the possibility to be visually integrated into the dashboards of Backstage. In order to achieve meaningful integration, a structural concept regarding the layout of the visual streaming interfaces needs to be developed.

1.2 Media streaming

The term "media", as used in this thesis, refers to electronic information comprising audiovisual content which is transmitted between two or more entities in computer networks. The continuous transport of media through networks is called "streaming". The difference between streaming and downloading media is that the end of a stream and thus its size, is not known when being transmitted, since there is no limit to the length of a stream. Further, streaming requires a constant rate of data packets, which must be sent and received in the same sequence. Additionally, playback within a certain time is crucial to maintain fluent audio and video when streaming data. Thus, late arriving packets may be discarded. Downloading data, however, requires each data packet to be received as fast as possible. The file is usually assembled once all packages have been obtained [2].
Streaming architectures generally consist of classic server-client connections. Requests from clients are handled by the server, which distributes media in accor-

dance with the pace of incoming requests. To channel media from one client to another, media is first sent to the server by the transmitting client and then forwarded to the receiving client.
Regarding Internet communication, various protocols may be used for data and file transmission, but not all of them are suitable for streaming purposes. One of the core protocols for Internet communication is the Transmission Control Protocol (TCP), which makes use of handshaking dialogues between the communicating entities in order to ensure reliable tranmission, when requesting web documents, for instance. Further, individual sequence numbers are added to each package, providing an orderly delivery. Retransmission of lost packets, as well as error handling are features of TCP. Hence, redundant bytes and large header information are added to the packets. Flow and congestion control mechanisms may slow down the transfer rate to prevent the receiver to be overwhelmed with data, meaning that real-time transmission of media cannot be assured [3].
When using the User Datagram Protocol (UDP), the sender only needs the IP address of the receiver and a port number which defines the enpoint of the connection where the received data is to be handled. UDP enables fast transmission of packages due to the neglection of handshaking dialogues and sequence numbering, which may result in media packages arriving out of order [3]. As a consequence, the Real-time Transfer Protocol (RTP) was developed in the mid 90s in order to achieve optimal streaming quality. This protocol is based on UDP, but timestamps and sequence numbers are joined to each media packet. In addtion to RTP, the Real-time Streaming Protocol (RTSP) was designed. RTSP is used in conjunction with RTP in order to establish connections and control media streaming servers [2].
By the use of codecs, audio and video signals are digitally encoded and compressed. Codecs differ in the strength of data compression, which affects quality. Hence, not every codec is suitable for media streaming. Regarding video, H.264 codec is the derivative of the H.263 and part of the ISO/IEC MPEG-4 standard, which is widely used on online streaming platforms and mobile devices. The H.263 codec thus has become deprecated but remains supported by most video streaming products [4]. Another major codec is the free video standard, Ogg Theora, which offers performance that is comparable to the H.264 codec [5]. Audio codecs mainly focus on lossy compression, especially in the field of voice encoding, because the voice frequency band ranges from 150 Hz to 5000 Hz only. The GSM codec is particularly designed for voice transmission and mostly used in telecommunications. Other codecs, such as the Advanced Audio Codec (AAC) and Ogg Vorbis provide sufficient quality for music streaming [6].

1.3 Thesis overview

At first, a system architecture is designed on a conceptual level, regardless of technical details. Actors participating in the system as well as basic hardware components will be identified and the general use case will be outlined. Hereby, certain peculiarities come to light and criteria and requirements on existing technologies, which could be used as a basis for later implementation, can be devised. These technologies include programming frameworks and interfaces as well as predefined media streaming components using different protocols. They are then evaluated and compared against one another based on the previously named criteria. Based on this comparison, the decision is made which technology will be used for the prototypical implementation.
The practical element of this thesis mainly deals with the technical implementation of a streaming system. However, questions regarding the layout arise and visual integration into Backstage has to be accomplished, too.
The last chapter exclusively deals with future work related to the prototype.

1.4 Sample scenario

In the following paragraph, a sample scenario is outlined, demonstrating the main features of a streaming system to be used in combination with a digital backchannel. This scenario aims at better understanding of streaming applications in academic environments and related abstract concepts.

For the last couple of weeks, the number of students attending the Chemistry lecture has been decreasing. When Dr Roberts asked his teaching assistants for possible reasons, they told him that the Biology lecture which is held at the campus on the other side of town, ends just half an hour before. Students would not have time to come to the Chemistry lecture in time. They also told him that he should not worry, because many of these students participate in the backchannel and can follow the lecture this way. For today's lecture, Dr Roberts has thought of an experiment which involves the use of hydrogen peroxide and sodium iodide. While he was preparing the experiment shortly before the beginning of the lecture, his teaching assistants were setting up several cameras. One camera is directed towards the lectern, another shows the blackboard and a third is placed in front of the experiment. Now Dr Roberts is introducing the experiment to the students, his teaching assistants capture video and audio and broadcast a live stream.
At the same time, Ann and Bob are on the train towards the city centre, which is not moving because of an overhead line defect. "Let's take a look at the backchannel", Bob suggests, "Last week, Dr Roberts mentioned a fascinating experiment and I don't want to miss it". After opening their laptops and signing into the backchannel, they can see Dr Roberts standing behind the lectern, his face full of excitement. Bob puts an ear plug into his ear. As Dr Roberts is moving towards the experiment desk, Bob selects the corresponding camera view. "What are the chemical substances he is using?", Ann asks and Bob replies: "I have no clue, check the blackboard!" Ann selects the video stream that shows the blackboard. On the backchannel, a discussion has commenced about what will happen in the experiment.

"Wow, that's cool!" Bob is amazed. After an hour stuck on the train, Ann and Bob meet their fellow students. "That was a brilliant experiment, don't you think?" They all agree. In the evening, Dr Roberts is at home, watching the lecture on the Internet. He is looking at the statistics that show him the analysed data of today's lecture: 150 students connected from inside the lecture hall and 200 students connected to the broadcast. That is 20 per cent more than the average amount. The chemical experiment as well as using the backchannel was a great success.

This scenario illustrates a sample use case of a digital backchannel with streaming capabilities: the system consists of several cameras, each capturing the lecture from a different angle. Students who are not present in the lecture hall are able to connect to the digital backchannel. There they find an option to connect to the live stream and follow the event in real time. After the lecture, the presenter and students can watch the recorded stream and assess their performace while being provided with statistical analysis of student behaviour, which is provided by the backchannel.

CHAPTER 2

Conception

2.1 Technology-independent architecture

In this section, a system architecture is designed on a conceptual level, regardless of technical details. The purpose of this structural sketch is to become aware of problematic conditions. This way, criteria and requirements can be devised in order to compare existing technologies which could be adapted for later implementation.
The concept is designed by identifying the participating roles, i.e. involved persons, arranging basic hardware components and outlining the general interactions within the system.
At the front-end, the graphical user interface of Backstage is accessible through any web browser. First, the lecturer, the capturing personnel, i.e. staff dedicated to capturing audio and video, and the students have to enter their login credentials in order to authenticate themselves as eligible participants of a session. Login details are transmitted to the login handler of the Backstage server that manages a database, in which user data is stored. This database is also used to determine the role of a participant within the system, i.e. lecturer, capturing personnel or student. Content on the user interface is displayed accordingly, which means that the interface components for capturing and streaming media are visible only to the capturing personnel, while a media player is shown to the students if they are located outside the physically shared space, i.e. the lecture hall. Co-located students are shown an option to play the live stream, for example, in case of bad acoustics within the hall. Roles do not necessarily have to be mutually exlusive, the lecturer should be at liberty to control the media stream.
Before being able to transmit or receive media streams, a connection from the respective clients to a separate server, dedicated to handling streamed media, is established.

A subsystem of this server, the communicator, verifies the the users' identity in order to prevent misuse. For this purpose, the participant's role within Backstage is retrieved.Once identified, the capturing personnel are able to transmit audio and video to the streaming server.
The communicator handles all client connections from lecturers and audiences, and receives the incoming media stream sent by the capturing personnel. This stream is then forwarded to every participant in the audience on the one hand and the extracted audio and video is stored simultaneously in a storage warehouse for access on demand on the other hand. Further, a media database is updated with references to the specific lectures, the actual media files within the storage warehouse and time stamps in order to synchronise the media content with lecture slides.
Therefore, a seperate communication path between the media server and the Backstage server is needed. The media player that is integrated in the layout of Backstage plays live audio and video or a stored media file on demand to enhance awareness of the remote audience.

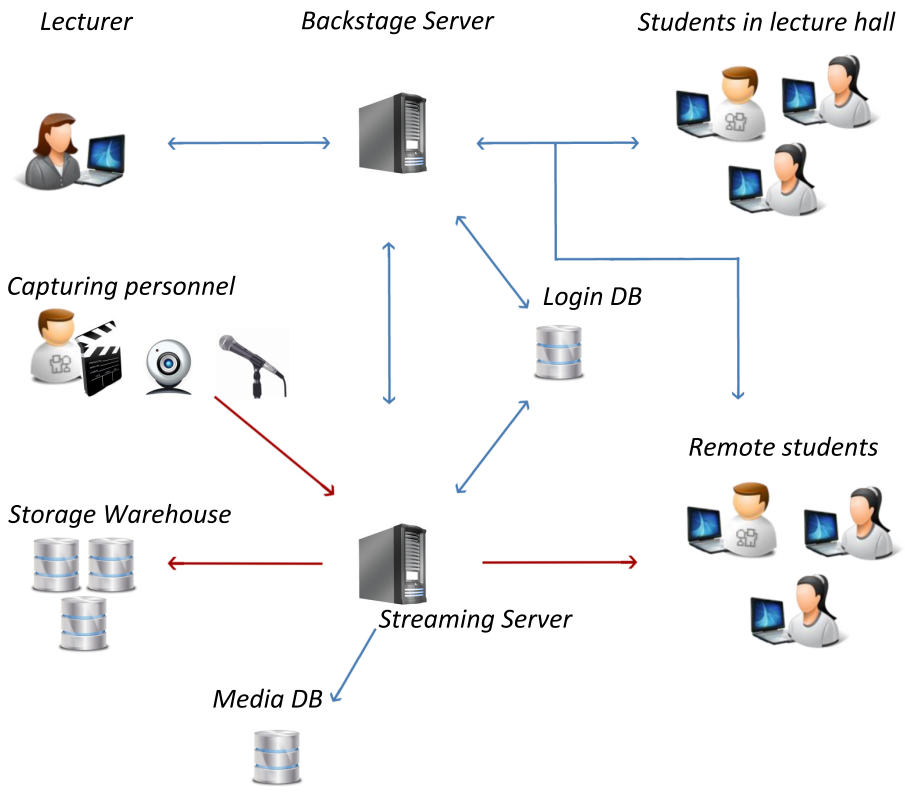


Figure 2.1: Draft of a technology-independent system architecture. Transmission of media is indicated by red arrows. Blue arrows represent communication paths and interactions between roles and hardware components.

2.2 Use cases and concepts

The introduction of streaming capability broadens the spectrum of use cases beyond the initial concept of Backstage. The ability to transmit audio and video to remote audiences makes room for novel concepts and further fields of application, some of which are presented later in this section. First, however, the general use case of the streaming system is outlined.

2.2.1 General use case

Before the lecture, the capturing personnel set up the cameras and microphones. They launch the streaming management application, select all input and then connect to the streaming server. As soon as the lecture begins, the capturing personnel starts to broadcast the video streams from the relevant cameras. The lecturer gives his presentation as usual without being considerably affected by the streaming system. After successfully signing into the backchannel, both co-located and remote students are given the option to connect to the media stream. They can manually select from the different available camera perspectives according to the displayed content which they find most important. During the lecture, the whole stream is not only broadcast live, but also stored on the server. This way, the lecturer, members of staff and students are able to watch the presentation on demand.

2.2.2 Combining roles

As afore mentioned, roles should not be mutually exclusive, which means that the lecturer is also able to assume the role of the capturing personnel. The streaming management is displayed on the lecturer's dashboards within the backchannel. The lecturer can choose from different cameras, pick a local video file and share the screen at the same time. For example, a camera showing the lecturer can be selected as input as well as a camera showing the chalk board and the screen containing source code. Even a virtual board, which is a real-time production tool for animated hand sketches, could be used for transmission [7].

2.2.3 Simulating real-time lectures

The ability to stream lectures to remote audiences also bares the potential to simulate live lectures. Pre-recorded audio and video could be streamed and broadcast live to create the illusion of a real-time lecture to non co-located students. A crucial difference between simulated real-time lectures and video on demand concerning the audience must be highlighted: while students can watch lectures which are available on demand at any time, simulated real-time lectures are aimed at an audience which is connected to backchannel at the same time and follow the stream simultaneously. Simulated lectures, for instance, allow the lecturer to solely focus on the backchannel while relieved from lecturing. Further, simulated lectures may be of high interest for a variety of studies. Gathering data could be achieved within the time frame of the lecture, whereas it would take an undefined amout of time

to accumulate data of sufficient quantity in the case of video on demand. Further advantages can not only be found in improved quality due to the possibility of cuts or retakes, but also in evaluating and assessing the lecture before broadcast. Further, high comparability of lectures is achieved by providing the same stream to different audiences.
Employing simulated lectures, certain functionalities of Backstage have to be considered. To begin with, collective message rating and feedback to the lecturer would be pointless for the current lecture. For example, the pace of lecturing cannot be changed while streaming pre-recorded video. However, this feedback may prove valuable and provide guidelines as to whether adjustments need to be made to the video, which could also mean that certain scenes need to be filmed again. Next, several features which promote awareness and interaction require manual activation. This implies the need of an agent, i.e. an authorized person controlling core features within Backstage. The role of an agent could be assumed by the lecturer or academic staff. The agent, for example, could be responsible for starting quizzes or lecture slide navigation, which would have to be synchronised with the students if lecture slides are not included in the video stream.

2.2.4 Providing comparable lectures

A well-known case, in which comparability is essential, is the annual repitition of courses at universities. Very often, different members of the same department hold lectures on a rotating basis, which results in different experiences and fluctuations in the quality of teaching. Simulated real-time presentations create persistent and highly comparable environments, but still leave the opportunity of adjusting content and pace of lecturing in future sessions according to feedback from the audience.
Further applications could include the deployment of Backstage to joint lectures or conferences. Presentations given in front of an audience at one conference and being broadcast to the audience at another location via live streaming have become widely spread in the scientific world. Combining the communicative and educational features of Backstage with streaming capability could enrich academic teaching. In a sample scenario, one lecturer offers a programming course shared by two different groups students, for example engineering and mathematics. Because of high attendance, sufficiently large lecture halls on the engineering campus are missing. Hence, it would be more convenient for maths students to use a venue in a math building. While being shown the live stream on the video projector, the maths students are connected to the engineering students via the backchannel.

2.2.5 Statistical analysis

Lecturers could be notified whenever a student connects to or disconnects from the live stream. Disconnections are more important, because a strong rise in this number could indicate a low niveau of the lecture, resulting in loss of interest.
However, providing this kind of information in real time could have a distractive effect on lecturers who might wonder what they could have done wrong, therefore

neglecting their task of leading the lecture. Such a notification system should be carefully considered. Thus, a statistical report in addtion to the standard report by Backstage could be made available on demand after a lecture. Lecturers can then draw conclusions from the students' streaming behaviour. For example, the Backstage report shows that some students complained about too much repetition at a certain point in time during the lecturer's presentation. At the same time, there was an increase in disconnects from students using the media stream. This additional information is further validation of comments that were posted by students and helps to identify the source of the problem.
Moreover, bare numbers of students connected to the media stream may also provide essential information. They provide a better overview of the total number of students who sit lectures and may take the final exams. Even temporal fluctuations could be used for analysis. If the lecture hall appears empty to the lecturer, it could help determine whether there is a general decline in the number of students sitting the lecture, or if students still hold interested and now just access the media stream from home instead of going to university.

2.2.6 Bi-directional awareness

Accessing information about the streaming behaviour in real time may increase the lecturer's awareness. For example, lecturers may be able to follow the streaming behaviour of their students. This could include information about the students' location. A method to determine the clients' location could be a comparison of IP ranges. The exact location does not have to be known, but it could be sufficient to know if members of the audience are connected from inside or outside the lecture hall. Being aware of the students' location, lecturers know if students are connected to the stream, even though they are in the lecture hall. Reasons for this specific behaviour could be, for example, high noise levels created by fellow students on the one hand, or the fact that lecturers speak too quietly on the other hand. Based on this real-time feed back, appropriate actions can be taken by lecturers.

2.3 Integration into Backstage

2.3.1 Structure

The visual layout of Backstage, consisting of so-called "dashboards", is phyiscally and semantically divided into three segments as shown in Figure 2.2. On the left, a communication panel can be found, showing messages by both fellow students and the lecturer alike. Here, users have the option to rate messages and, doing so, contribute to determine which questions are to be forwarded to the lecturer. The shared artefact, i.e. a canvas which not only displays lecture slides, but also provides controls to place questions or give feedback, is located in central area of the dashboard. On the right, the awareness panel shows the lecturer's status panel and

statistics of user behaviour.

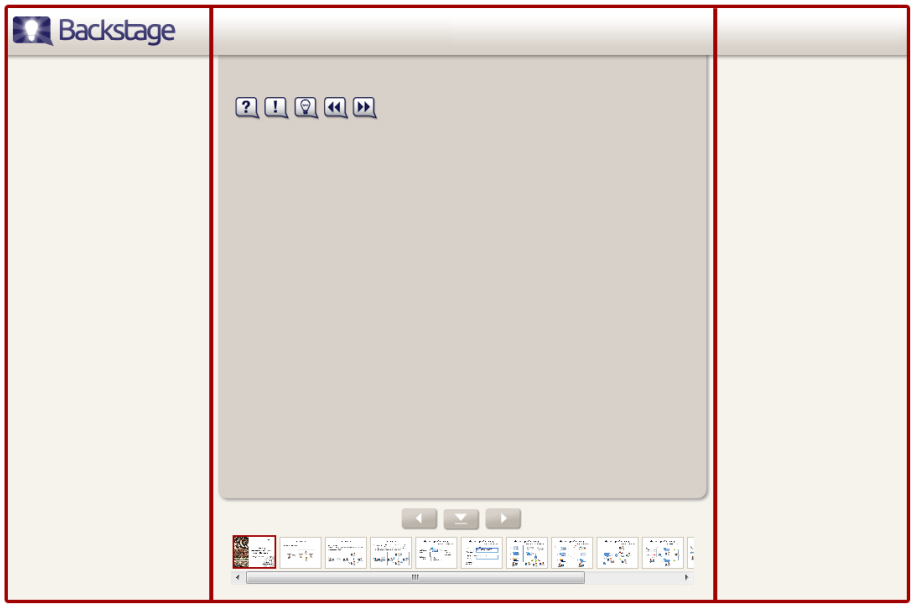


Figure 2.2: Tripartite layout of Backstage. The red lines indicate the structure of the front-end. The left frame encloses the communication panel, the shared artefact is placed in the centre and the right frame represents the awareness panel.

To integrate audiovisual streaming into Backstage, this fundamental structure has to be conserved. Components of the streaming system have to be added to appropriate parts of the dashboard in order to maintain the semantics of the current layout. The streaming system will be capable of broadcasting multiple video streams, which have to be presented in different locations within the students' dashboard. For instance, the video stream which shows the lecturer has to be added to the awareness panel. Streams of the board and the lecturer's screen are regarded as shared artefacts and thus are displayed in the centre of the dashboard. The lecture slides and streams have to presented in an appropriate way, so that the students have the option to switch easily between each shared artefact.

2.3.2 Adjustments to Backstage

The fact that multiple streams may be required raises questions regarding usability. Students should be made aware if live broadcasts are available. They should also be at liberty to decide whether they want to follow the stream or not. This could go hand in hand with the so-called "study mode" of Backstage, which hides the awareness panel to provide an environment without possible distractions. Further, the option to enable and disable single streams should also be offered in order to create an individual study environment. However, the lecturer or the capturing

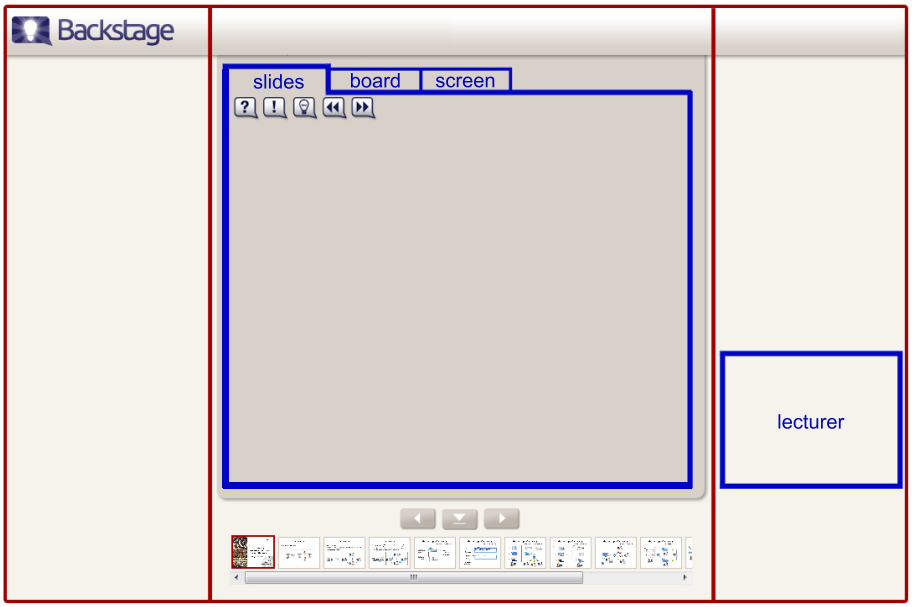


Figure 2.3: Integration of streaming components into Backstage. Blue lines mark the areas in which streamed content is displayed.

personnel could set the main focus to a certain camera, which could be indicated by a flashing light next to the respective user control. This way, students are notified in a subtle manner which carries a recommendation, but no obligation to activate a particular stream.
Presentation slides and streams representing shared artefacts must be located in the centre. Providing high usability, these artefacts can be organized on top of each other. By clicking on the corresponding tags, students are able to navigate through the artefacts, which will overlay the others when selected. Navigating through different layers is commonly implemented by tabbed document inferfaces, or `TDIs` [8].
Usually, lecture slides are projected to the wall in front of the audience. Hence, co-located students can navigate through slides according to visual information in the lecture hall. Students who are connected to the backchannel and live stream from outside the physically shared space, however, are not able to know when the current slide changes. A synchronisation mechanism has to be implemented for these students. Developing this mechanism, the fact that students may be writing a question to a topic on the current slide has to be taken into consideration. In this case, changing slides automatically must be prevented. Nevertheless, highlighting not only the slide that is visible to the student, but also the slide which the lecturer is refering to, would prevent confusion among non co-located students.

When following a streamed lecture on demand, video and slides have to be synchronised in a bi-directional manner. When the stream is played, slides change and are presented to the user according to the content of the stream. The stream should be structured by both the lecture slides and further cue points, which could refer to events, such as important questions or announcements. By selecting such a cue point, the user can navigate through the stream, which begins to play from the corresponding event. This way, whole lectures can be re-enacted, providing authentic remote teaching and learning environments.

CHAPTER 3

Technology selection criteria

Due to certain characteristics which have become evident within the technology-independent conception, criteria and requirements must be met by technologies that are available as candidates for a prototypical implementation.
First of all, only open-source software is to be used on the server side. The prototype which is to be implemented as part of this thesis should form the basis for further development. Hence, adjustments concerning the server may become necessary.
The next and arguably most important criterion is the multi-platform availability of front-end applications, since different operating systems and browsers will be used by both lecturers and students and this service should be available to a wide remote audience. In particular, handheld devices need to be considered for the implementation of the receiving clients.
Next, standalone applications at the front-end should be avoided in general. Because these usually have their own graphical user interface and run as own applications next to the browser, they would complicate the simultaneous use of the feedback and collaboration platform and the audiovisual service of Backstage. Thus, the media player as well as the components for capturing and streaming media need to be integrated into the current dashboards of Backstage. However, capturing personnel are not meant to participate in the communication on the backchannel. Access to the backchannel is thus not necessary for them in order to provide the media stream. As a result, standalone software is acceptable for personnel who is dedicated to capturing and transmitting the streams.
In addition, the visual component of the streaming system has to be integrated in the audience's dashboards. Layout and design of visual elements have to be assimilated to Backstage in order to provide a homogeneous appearance. Technology to be used for the prototope in this thesis should thus offer possibilities for flexible interface design.
Regarding usability, simplicity of user controls is a central criterion. If lecturers decide to manage the capturing themselves, they have to give presentations and deal

with large audiences in front of them at the same time. Therefore, the interface has to be very simple, basic and user friendly. This includes the implementation of only few buttons and controls. If there is additional personnel, this personnel can focus on the task of capturing and broadcasting. More detailed controls should be visible on that personnel's graphical interface. Nevertheless, simplicity of controls should enable a smooth and flawless workflow.
Furthermore, control over what is being streamed and stored should reside with the lecturer. For example, the lecturer may give information which is not to be recorded or stored on the server for replay and public access. A situation which occurs regularly, is that there will be short breaks during the lecture. Hence, it is essential that the interface is designed accordingly, allowing the lecturer to pause either element of the stream, audio or video. This makes individual adaptability an important guideline for the integration of capturing components in Backstage.
Moreover, data security is an equally important factor. Thus, external streaming servers, or service providers offering these, have to be avoided. All media data is to be stored on servers run by the university alone.
Requirements regarding the back-end part of the system are only hardware related. The connection handling component of the server, the communicator, needs to deal with many threads due to large audiences. The storage warehouse has to offer uninterrupted service. Hence, either the storage servers must support error-correcting mechanisms, or stored files have to be mirrored. The storage warehouse, however, only resides on the streaming system and its mechanisms should be already available within the storage system and are not dealt with in this thesis.
To offer the audiovisual service on demand, synchronisation between the streaming system and Backstage has to be established. Temporal relations between lecture slides and media streams must be specified and preserved by appropriate mechanisms within the media database.

CHAPTER 4

Technology review

4.1 Considering HTML5 for streaming

The main benefit of this approach is multi-platform availability due to HTML5 in combination with JavaScript. There is no need for additional software or browser plugins on the client side, because native device support is already integrated in browser frameworks, such as the MediaStream interface that is part of the Stream API [9]. This allows high usability and comfortable integration in Backstage.
Two browsers, namely Google Chrome[10] and Opera[11] already offer frameworks for native device support and thus allow a client to capture video and transmit a stream to a server. Receiving a stream and playback in an `img` element is also possible within two browsers: Google Chrome and Mozilla Firefox. Announcements give hope for full implementation of the MediaStream framework within further browers.
Distributing the stream as a broadcast, server technology which can communicate with WebSockets is required. The WebSocket protocol, which most modern browsers are compatible with, provides bi-directional communication channels. A great advantage of this is high data throughput compared to the HTTP protocol which would create a large overhead when many packets are transmitted [12].
An open-source project, suitable for a system based on HTML5, is the Jetty project, which offers a standalone server that is capable of handling WebSocket requests [13]. In addition, Jetty libraries are widely available and can be embedded in a Java project. This way, existing web socket handlers can be used within a seperate server program. Incoming media streams that are represented as byte arrays may be dealt with so that the server program broadcasts the stream to users who are logged in and stores the data at the same time.
The MediaStream framework itself does not offer methods for selecting a default device yet [9].

Nevertheless, if a computer has more than one device of the same kind, i.e. video or audio input, the person in charge of capturing can easily set the default device in the operating system before the lecture.
More crucially, while video input from webcams is currently supported by Google Chrome and Opera, incomplete and faulty implementation by browser vendors prevents audio capturing from microphones [14]. At the moment, there is no way to get samples from a microphone using the MediaStream framework. Thus, this approach cannot be pursued for the prototypical implementation in this project. However, it can be assumed that in the near future, HTML5 based audiovisual media streaming will be possible. For this reason, an implementation draft has been developed, which can be found in the appendix.

4.2 Flash and Red5 Media Server

4.2.1 Adobe Flash

The Adobe Flash authoring environment is widely used to create multimedia content for web pages in the form of embedded objects [15]. Adobe Flash offers tools for highly flexible and individual design. Flash content can only be played by so-called Flash Players, which also offer the efficiently compressed H.264 encoding for video in their most recent versions. Other streamable encoding types include On2 VP6 and Sorenson Spark for video and MP3, AAC and Speex for audio [16] [17].
Flash players are available to a variety of platforms, mainly as plugins for browsers. All major desktop browsers are supported. Although Flash is supported by some Android based smartphones, Windows Phone, Apple iPhone and iPad lack the ability to play Flash content. In addition, it has been announced that development of the Flash Player for Android will no longer be continued [18]. However, with Flex and Air, Adobe also offers open-source application frameworks, which allow the development of mobile applications for Android and Blackberry. Further, Adobe Flash employs the Air framework to create byte code applications for handhelds based on iOS, which could be considered for a future version of Backstage dedicated to mobile devices [19].
Using ActionScript, the object-oriented language for Flash, a high level of interactivity can be achieved. This language is promoted by Adobe and therefore well documented [20]. It not only provides movie controls, but also makes capturing from media devices and socket communication for media streaming via the Real Time Messaging Protocol, RTMP, possible. The proprietary RTMP is based on TCP and offers extensions for encryption or the use via SSL connections [21].

4.2.2 Red5 Media Server

Red5 Media Server, which is mainly Java-based, is an open source project trying to provide an alternative to the proprietary Flash Media Server by Adobe. Because the Red5 Media Server is mainly based on Java, libraries are provided for development in Java [22].

The server naturally supports NetConnections and NetStreams based on the Real Time Messaging Protocol. Red5 automatically handles incoming streams by assigning them a unique reference in the form of a URL, e.g. "rtmp://server:port/stream". The identifier can be set by the broadcasting client using the `publish()` method of the `NetStream` class. Clients that want to connect to a stream simply request the stream by transmitting the URL to the server. The server then forwards the reference to the media packages received from the broadcasting client [23].
Red5 Media Server also offers storage of received input. Media files are stored in an open container format for video and audio developed by Adobe. This format supports H.264 video encoding which can be played by the most recent Flash Player, and thus is suitable for fast transmission over the Internet. Further, cue points can be stored within the container format and would not have to be stored in a distinct database. Cue points are invisible markers in a video file which can be used to trigger external events, such as the change of lecture slides [24]. This way, lecture slides and media stream can be synchronised efficiently.
Moreover, the Red5 Media Server uses the Spring framework, which allows customized Java applications to be integrated into the server via dependency injection based on XML configurations [25]. The Spring framework enables developers to extend the server beyond the basic streaming functionalities, which could prove to be highly valuable for the realisation of concepts.
Despite the fact that the set target of the Red5 project is to create a sufficient alternative to the proprietary server by Adobe, the Red5 Media Server lacks essential functionalities. Most importantly, protocol changes and transcoding on the fly into a different streaming format are not offered [23]. Therefore, neither RTP nor HTTP streaming are applicable and devices to which Flash is not available, would be left out completely.

4.3 Darwin Streaming Server

The MacOS Forge works on several open-source projects in order to create freely available versions of well-established proprietary products. One of these projects is the Darwin Streaming Server, which is the open-source version of Apple's QuickTime Streaming Server (QTSS). While QTSS and its broadcast client, the QuickTime Broadcaster, are only available for Apple Mac OS X, the sources of the Darwin Streaming Server can also be compiled and run on linux machines [26].
QTSS, however, supports HTTP live streaming, which is essential for the use of mobile devices, such as the iPhone and iPad [27]. Due to the nature of open-source projects which are dedicated to rebuilding standard software, the Darwin Streaming Server is still behind the proprietary version and offers media delivery using RTP/RTSP only. Moreover, documentation is highly incomplete and individual changes to the server can only be made within the source code of the server. The last update of this project was in 2008, which raises doubts about future development.

Achieving multi-platform compability, Darwin Streaming Server provides support for the digital media standards H.264 and 3GPP, which, for example, can be decoded by the QuickTime plugins available to all browsers. Any plugin or applet for browsers, or standalone application which uses RTP and are capable of appropriate encoding, can be deployed in order to transmit media to the server [26].

4.4 VideoLan and the VLC media player

The VideoLan organisation hosts a variety of open-source projects including the VideoLan Client, or VLC. Originally, a seperate server, the VLS, was developed with the intention to provide media streams in real time or on demand. Because the client was enriched with server functionalities, VLS has been deprecated since 2003 [28].
The VideoLan team's deep commitment is the reason for comprehensive documentation and support for nearly all standard codecs and formats. Media devices are accessible efficiently and by the use of Java bindings for VLC, own applications can be developed that use the functionalities of VLC.
In principle, many different platforms and devices can be served by VLC and even a HTML5 client receiving the stream is theoretically possible. Considering mobile devices in particular, this would be highly beneficial [29].
There are, however, two drawbacks concerning this approach. Firstly, different browsers support different codecs. For example, Safari exclusively supports the H.264 video compression, while Google recently annouced the removal of this technology from its Chome browser. Secondly, real-time audio and video streaming with VLC is connected with high delay in transmission and playback, which has been observed in several test runs, the results of which can be found in the appendix.

4.5 Java Media Framework

The Java Media Framework (JMF), initially developed by Sun, is currently the only open-source, platform-independent and object-oriented media processing framework which supports media devices, such as cameras and microphones and also RTP streaming [2]. The development of the JMF has not been continued since 2002, mainly because of the introduction of JavaFX, which is a platform-independent framework used to design Rich Internet Application but lacks the support of media devices.
A variety of encodings appropriate for real-time streaming is offerend by the JMF which include MP3, GSM, G.723 and ULAW for audio, MJPEG and H.263 for video. For mobile devices, the Java Mobile Media API (MMAPI) offers part of the functionality of the JMF [2]. Applications and/or applets for webpages do not run on machines unless the JMF, or the MMAPI respectively, is available. Both are not included in the standard Java distribution and would have to be installed seperately. Further, this approach brings drawbacks concerning multi-platform compability.

The use of the RTP protocol prohibits cross-browser implemenation because of browser vendors' strong focus on the development of HTML5. This includes, for example, Google Chrome. To reach Chrome users, a Java applet would have to be implemented and integrated into the current dashboards of Backstage. However, Apple as well as other mobile device vendors do not support Java applications within browsers on their mobile devices.

4.6 Technology selection

So far, five different open-source technologies which provide basic components for streaming systems have been presented. Advantages and disadvantages regarding encoding, documentation, streaming technology, multi-platform availability and individual adaptability have been elaborated on. Figure 4.1 compares the different technologies based on the aforementioned criteria.

	A/V Support	Documentation	Streaming	Media Codecs	Multi Platform	Future Prospect
HTML5	–	+	WebSocket	Theora/VP8	+	+
Adobe Flash / Red5	+	+	RTMP	H.264/AAC	o	+
Darwin Streaming Server	+	+	RTP/RTSP	H.264/3GPP	–	–
VideoLan	+	+	HTTP/RTSP	H.264/AAC	+	+
JMF	+	+	RTP/RTSP	H.263/GSM	–	–

Figure 4.1: Comparison of open-source technology suitable for this project. Cells shaded in green indicate sufficient capability in regards to a specific criterion. Yellow cells show problems which may be circumvented, whilst red cells portend severe shortcomings which prevent deployment. White cells illustrate criteria which could not be assessed due to inadequate documentation.

Looking at the table, it can be observed that the HTML5 approach would offer a well-documented framework which uses modern streaming technology and media codecs, namely Ogg Theora and VP8, which are becoming widely spread among browsers. Further, native device support has been implemented, so that the browser itself can handle input media devices not depending on third-party plugins. Since browser vendors, for example Google, and companies that develop authoring tools for the creation of rich internet media, such as Adobe, focus on stronger HTML5 support and integration in their products, this approach has an excellenct future outlook. However, due to incomplete and faulty implementation, microphone capturing is not yet available. The HTML5 approach, thus, cannot be followed in order to develop a fully operational prototype within this thesis.
In contrast to the proprietary QuickTime Streaming Server, which is able to provide media streams to the majority of mobile devices, the open-source version, namely Darwin Streaming Server, does not support the HTTP Live Streaming technology. While the use of the RTP/RTSP protocol may not be a crucial disadvantage, documentation of the server software is not comprehensive and adding features to the server software would require applying changes to its source code. Deploying clients on different platforms is complicated. Although any application or plugin that supports RTP/RTSP could be used, none of them runs on a variety of platforms, such as mobile devices. Browser vendor Google even announced, that the real-time streaming protocol will not be supported in future releases of its Chrome browser.
The VideoLan project offers the VideoLan Client (VLC), which is an open-source standalone software capable of transmitting media over HTTP. Java applets can make use of the software via Java bindings which are also freely available. VLC supports all modern media codecs and is able to handle incoming media streams and broadcast them. The latter, however, suffers from high delay presumably caused by protocol overhead. This delay prevents deployment of the technology within this project. For example, a discussion may already have commenced in the backchannel, while the crucial event that has lead to the discussion, has not been transmitted yet, which causes confusion among the audience. Understandably, this is not acceptable for integration within a real-time backchannel. The future prospect of this technology, nonetheless, is promising due to the project team's strong commitment.
The Java Media Framework (JMF) was developed for the purpose of processing and streaming media over networks. Due to the use of the RTP/RTSP protocol and slightly outdated media codecs, delay occurs in transmission, which is, however, tolerable. Regarding serious disadvantages, the JMF - as well as its version for mobile devices - is not part of the standard Java distribution and thus has to be installed on the system seperately. More crucially, development of the JMF has been discontinued because of the introduction of JavaFX, which currently lacks the support of media devices.
Most promisingly, Adobe Flash makes use of modern media codecs as well as an excellent streaming protocol, namely RTMP. The object-oriented programming language used to implement Flash applications is well-documented and allows efficient development of applications. For example, packages for handling media de-

vices are predefined and only need to be imported. However, there is Flash plugins only for desktop and laptop browsers and a few mobile devices, which embodies a disadvantage concerning multi-platform availability. Because of this, Adobe is devoted to developing cross-platform solutions, for example Adobe Air, in order to bring Flash content to mobile applications. The Red5 project offers a streaming server that is able to handle RTMP streams in the same manner as the proprietary Adobe Media Server. Unfortunately, unlike the proprietary version, Red5 does not support transcoding streams, so that many mobile devices would be incompatible. Nevertheless, the Red5 project is still active and it can be assumed that necessary extensions will be included in the near future. Further, Red5 offers individual expandability, which could prove useful for the development of complex streaming systems. Alternatively, the media server can be replaced by the proprietary Adobe Media Server for more functionality after a fully operational prototype has been implemented based on the open-source server.
Consequently, the combination of Adobe Flash and the Red5 Media Server has been chosen as technology to be used in the practical part of this thesis.

CHAPTER 5

Implementing the system

The system architecture employing the prototypical implementation consists of two servers: the Backstage server and the Red5 streaming server. Front-end applications are built using Adobe Flash, an authoring environment that combines the creation of visual components and efficient development using the object oriented programming language, ActionScript. ActionScript provides APIs for uncomplicated communication between Flash applications and media servers.
The Red5 streaming server is capable of handling data object tranmissions and storing cue points in the FLV container format. Hence, a seperate database which manages timestamps for synchronisation between lecture slides and media streams is not required. Further, the Red5 server employs the Spring framework, allowing customized Java applications to be integrated into the server via dependency injection based on XML configurations [25]. Several demo applications are included in the initial server package, one of which, namely "`oflaDemo`", is equipped with all major functionalities. For instance, this demo application is able to handle incoming media streams by multiple providers, distribute these streams to consumers and store them in a container format simultaneously. Because the `oflaDemo` is also capable of dealing with data objects, cue points can be added to the stream within the container. Hence, the `oflaDemo` is suitable for deployment regarding not only live streaming, but also video on demand. As a result, the prototypical implementation of this thesis exploits the potential of the provided demo application.

5.1 Stream management

Concept and use cases require the stream management, i.e. the user interface for capturing and broadcasting media streams, to offer options for camera selection and broadcast intermission. The prototype enables the camera personnel to select and individually label up to three cameras and a microphone by the implemen-

tation of drop-down lists. When a camera or a microphone is selected, the Flash Player asks users for permission to access to media devices to prevent misuse and unauthorised recording, as seen in Figure 5.1.
Preview panels, which are located below each drop-down list for camera selection, show the video of the cameras accordingly, allowing the camera personnel to monitor and control camera settings, for example light conditions or camera angle. Further, check boxes are placed above each preview panel. Ticking these selects the corresponding camera for broadcast.

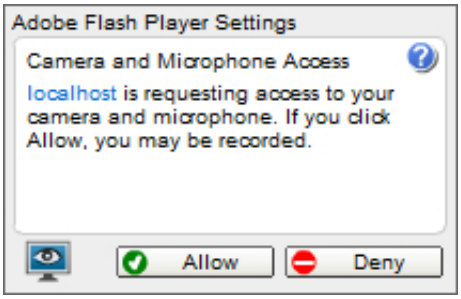


Figure 5.1: Flash Plugins request permission to use media devices.

Two buttons are added to the user interface: `Broadcast` and `Stop`. Naturally, `Broadcast` starts transmitting the streams from the selected cameras and microphone to the streaming server, which distributes and records the stream. `Stop` pauses the transmission. If the camera personnel activates the broadcast button after the tranmission has been paused within the same session, the newly transmitted video and audio are appended to the previously recorded stream, which is uniquely identified by the respective session of the lecture. The graphic user interface which represents the stream management can be seen in Figure 5.2. Not only physical cameras, but also virtual ones can be selected. By using external programs, for example ManyCam (Figure 5.3) [30], cameras can be simulated and registered with the system. As a consequence, it is possible to stream pre-recorded lectures or sections of the lecturer's screen.
The stream management has to be available to the lecturer and the camera personnel. If the lecturer does not assume the role of the camera personnel, there will be need for a distinct version of the interface, since the camera personnel then has no access to Backstage. This indicates the demand for a standalone application in addition to the Flash object that is integrated into the presenter's dashboard. While the integrated interface obtains details regarding address of the streaming server and session identifier of the current lecture from the backchannel, the standalone version offers user interface elements for manual configuration. Furthermore, the stream management which is included in the lecturer's dashboard provides methods for adding cue points to the stream, which could be triggered by controls for lecture slide navigation.

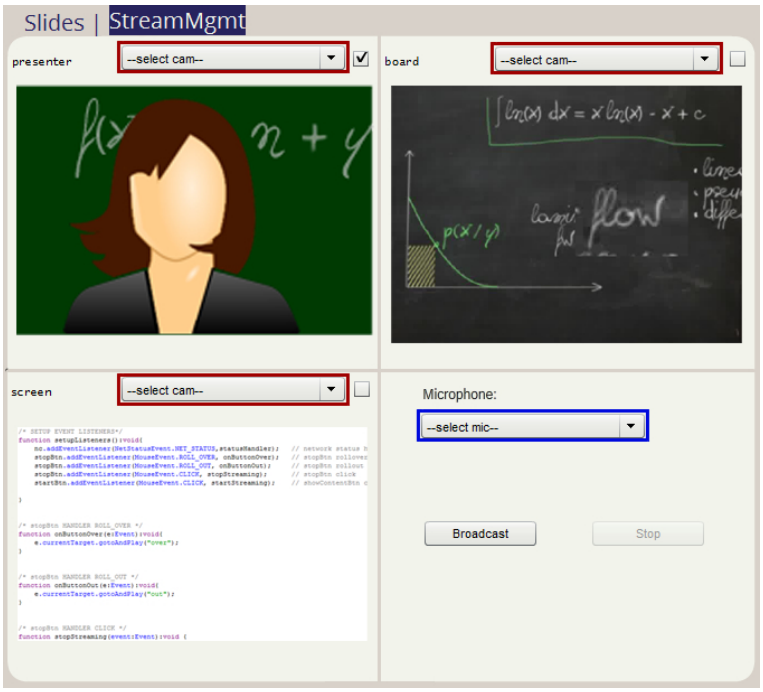


Figure 5.2: The stream management interface is integrated into the presenter's dashboard. Red and blue areas point to user controls for camera and microphone selection. Video panels offer previews of cameras. Check boxes indicate the cameras which are being broadcast.

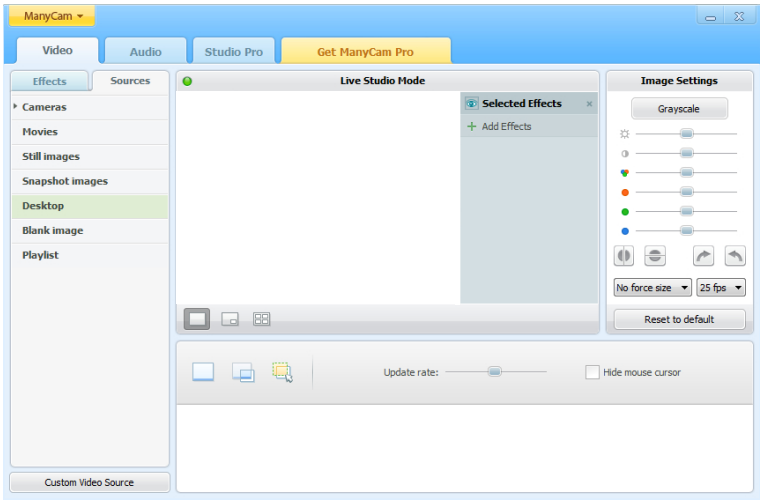


Figure 5.3: ManyCam supports screen grabbing and video files as input for simulating a camera.

5.2 Audience

Conserving the fundamental structure of Backstage, the students' interface is split into two segments. On the one hand, the element showing the lecturer is located inside the awareness panel, which itself is placed on the right side of the dashboard. On the other, elements representing shared artefacts, for instance the board or a sectio of the lecturer's screen, are placed in the central section of Backstage.
The awareness panel consists of statistical analysis of student behaviour and a section dedicated to the lecturer [31]. Below the lecturer's message of the day, the video element that shows the lecturer is placed. The layout of the prototypical implementation concerning the perspective of the lecturer is presented in Figure 5.4. This element is constantly visible, regardless of adjustments to the dashboard appearance made by the students. Students can decide whether to follow this stream or not, as disconnecting and reconnecting is possible at any point. However, audio transmission is only provided when connected to this stream, for it represents the main perspective.

Figure 5.4: The video stream showing the presenter is located at the bottom right of the students' dashboard.

Contrary to the video element dedicated to the presenter, shared artefacts, such as lecture slides and video streams of the board or screen, overlap and thus are not constantly visible. Each artefact is encased by its own layer and only displayed when activated by the user. Buttons above the shared artefact indicate the currently selected view.

There are no limitations as to which content is being streamed, but two of the arguably most streamed artefacts are the board and screen. To provide improved legibility, a magnifying tool is added to the Flash object. The video stream of a shared artefact is received twice, each in a different resolution. Using a layer mask with Flash, only part of the video with higher resolution is displayed. This way, the effect of a magnifying glass is created. In Figure 5.5, this effect as part of the prototype is demonstrated.
In Figure 5.6, one can see the transmission of the lecturer's screen as a shared artefact. In addition to a magnifying tool for enhanced legibility, quality and resolution of the transmitted video are elevated by the stream management. Moreover, students are at liberty to connect, disconnect and reconnect to the stream at any point.

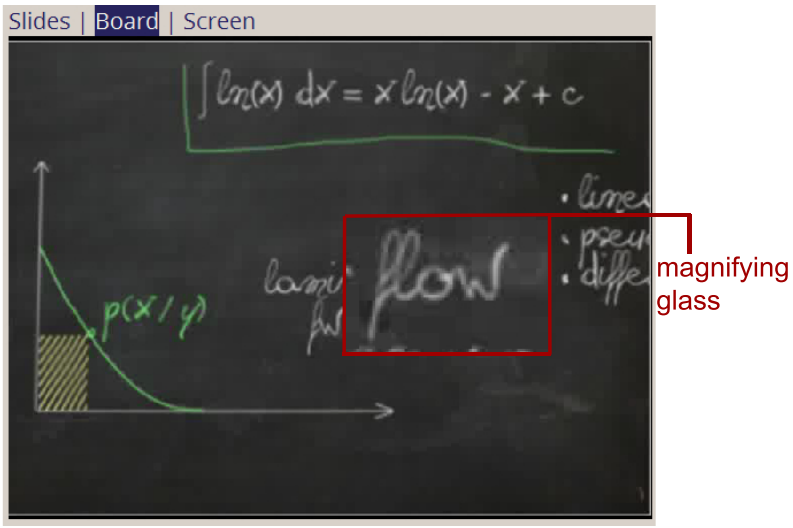


Figure 5.5: The board as a shared artefact with a magnifying tool for improved legibility.

5.3 Integration into Backstage

Flash applications can be embedded into HTML web pages, generally by using the `object` and `embed` tags. However, this approach may cause high complexity at the respective place in the source code. Thus, it is preferable to follow a JavaScript approach which only requires one `div` element in the HTML source code. SWFObject, an open-source library [32], is deployed in order to embed Flash content to the dashboards of Backstage. The `embedSWF()` method has several parameters for necessary and highly useful functions. For instance, dimension, attributes and initial parameters can be passed to the Flash application. This way, the flash application that is used to display the shared artefacts receives information about the corresponding stream, for example board or screen, in the initialisation phase.

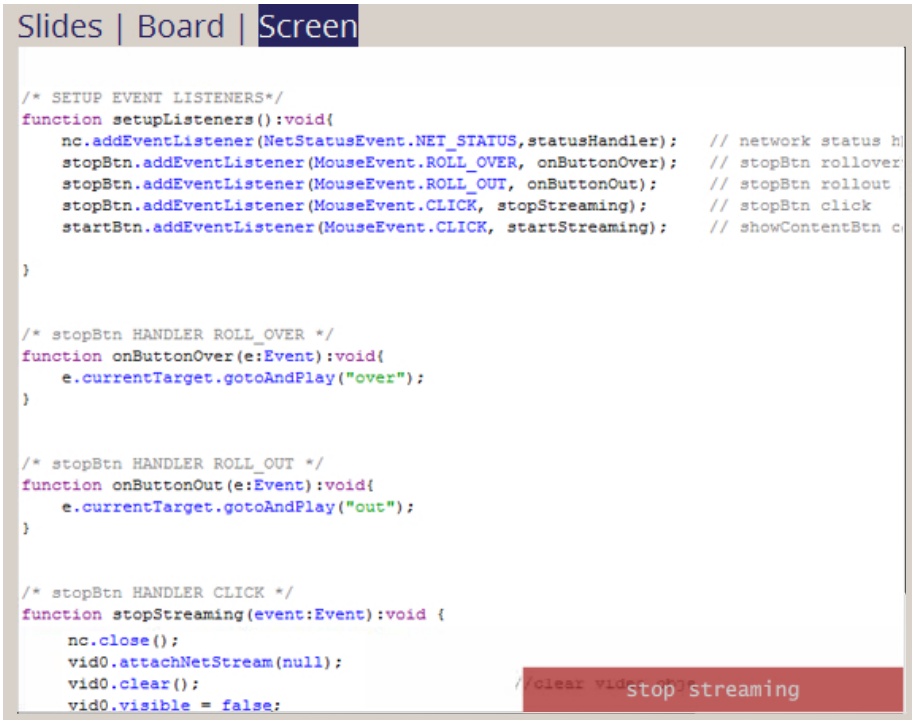


Figure 5.6: Shared artefact demonstrating the ability of screen grabbing and the option to disconnect from the stream.

As a result, only one Flash application is to be built for all shared artefacts, instead of distinct applications for each stream.
Communication between the Flash object and the web page is also achieved by using JavaScript. Flash provides the `fscommand()` function in order to call JavaScript methods deployed in the web page which hosts the Flash application. Functions within the flash application can be made externally available, accomplishing a communication path in the opposite direction.

CHAPTER 6

Future work

6.1 Synchronisation

Backstage is currently designed for co-located audiences alone. Technical means which synchronise the lecturer's and students' experience of the lecture are not necessary at present. Adding streaming capabilities to the backchannel though, expands the field of application beyond the lecture hall. If the lecturer changes presentation slides, which are usually projected on the wall in front of the audience, non co-located students must be made aware. Backstage has to be extended by adding a module which ensures synchronisation between the presentation slides visible to the lecturer and the students. When implementing this module, one has to consider the fact that students may be writing a question to a topic on the current slide.
The prototypical implementation as part of this thesis bears the functionality of requesting and playing media streams from the streaming server. Due to the configuration of the students' streaming interfaces, the live stream, which is associated to the respective lecture by a unique session identifaction number, is displayed. If a live stream is not provided, the applications request a recorded media stream with the same session identification number. This way, video on demand is already available to Backstage.
However, certain restrictions regarding synchronisation prevail, obstructing deployment in an appropriate manner. Currently, students can decide whether and when to start following the single streams showing different perspectives. This results in delay between the different streams, when activated one after the other. To prevent this undesirable behaviour, a synchronisation mechanism between the streaming applications would have to be implementated. Inter-object communication between the different interfaces may be necessary to regulate correct timing.
In addition, the media stream for on demand replay must be synchronised with lecture slides in two different ways. On the one hand, users should be able to navi-

gate through the media stream by the use of a timeline. Presentation slides change accordingly, enabling users to replay short parts of the lecture, for instance. On the other, navigation by selecting a distinct presentation slide should set the stream to the time this slide was shown at, allowing users to skip or replay logically seperated chunks of the lecture.

6.2 Handheld devices

According to a survey conducted among the students first to test Backstage within a lecture about ownership and usage of electronic devices, 90 per cent of students own a laptop computer, yet 30 per cent report that a laptop is impractical to be used with Backstage [33]. In the survey, the importance of more compact, handheld devices is clearly pointed out and thus the streaming functionalities should be made available to this sort of device. Peculiarities of mobile devices must be taken into consideration, for example multi-touch interaction and navigation using gestures. Based on the Adobe Flash authoring environment, mobile applications can be developed, which either exploit the Adobe AIR framework or, in the case of Apple iOS, are compiled down to byte code. A draft for a mobile stream player using this technology is included in the appendix of this thesis. Methods need to be explored as to how the stream player could be integrated into a mobile version of Backstage.

6.3 Usability

Based on the structural layout of Backstage, the prototype provides two split interfaces for video visualisation: the presenter panel and the shared artefact. User studies are fundamental to determine how intuitively users can interact with the new interfaces. Depending on the outcome of these studies, it may be unavoidable to completely change the base structure of the streaming interfaces in order to achieve optimal user experience.
Further, user controls have been largely neglected in the prototype, especially in the students interfaces. Students can only connect and disconnect from streams and use a magnifying tool for better legibility. Essential functions, such as adjustment of audio volume or manual resizing of video panels, are missing and will have to be added. Although audio volume can be adjusted by the use of operating system controls, controls within the streaming component could prove useful, for example if there is another audio track being listened to simultaneously.
Allowing students to follow a lecture, but study individually and remain undisturbed, Backstage is equipped with a so-called "study mode". This mode changes the students' dashboard in such a way that the message column and awareness panel are hidden. Regarding the media streams, it is mandatory to evaluate to which extent these streams affect students or if it should be entirely at the students' discretion to follow or disconnect from a stream.

6.4 Bi-directional awareness

Not only the students', but also the lecturer's awareness is to be enhanced. The lecturer may be allowed to access information about the students' streaming behaviour, for example the amount of students connected to a stream. Further, the students' location could be determined by comparing IP address ranges. The prototypical implementation is based on one of the demonstration applications that are delivered with the Red5 Media Server. Adjustments to this application may be required in order to extract essential information and provide valuable feedback to the lecturer. Hence, it is of high importance to conclude to what extent Backstage is capable of making this information available or if the server application must be extended.

6.5 Technical improvement

The prototype consumes vast resources and bandwidth, which could lead to repurcussions, such as delay of media transmission or deterioration of system performance to an extent that users are confused, irritated or even distressed. Once students have activated a stream, a connection to the streaming server is established. This happens again for each further stream the student connects to. As a result, there may be a maximum of three transmissions simultaneously, even though only up to two, the presenter and one shared artefact, can be visible at once. Technical improvement is therefore compulsory. By notifying the streaming component that it is not visible to the user, i.e. when the encasing layer is hidden, the stream can disconnect and automatically reconnect when the user selects the artefact to be displayed again. Alternatively, the streaming component for shared artefacts could be deployed only once. When the user selects a tab, the application could end the current transmission and trigger a reqeust for the respective stream, which is sent to the streaming server. To accomplish this behaviour, communication between the web page and the embedded streaming component is imperative and must be implemented.

CHAPTER 7

Conclusion

The outcome of an initial conception includes a technology-independent system architecture, which identifies the interacting roles and hardware components within the system. Roles comprise of the lecturer, camera personnel and co-located as well as non co-located students. It is necessary to combine certain roles. For example, the co-located students should have the choice to assume the role of non co-located students in order to follow the media streams. Moreover, the lecturer should be at liberty to handle the stream management and thus, adopt the role of the camera personnel.
The introduction of streaming capability to Backstage broadens the spectrum of use cases beyond the initial concept of Backstage. The simulation of real-time lectures could form the basis for a variety of studies and offer persistent and comparable teaching environments. Further, the deployment of Backstage to joint lectures or conferences is another field of application that could be imagined.
Furthermore, it is essential to integrate the components of the streaming system into the structural layout of Backstage, which is divided into three segments. On a conceptual level, the process of integration regarding structure and usability takes place, which implies guidelines concerning the graphic user interfaces.
Certain characteristics of the streaming system have become evident within the conception phase and define criteria and requirements that must be met by technologies that represent candidates for a prototypical implementation. These criteria include the exclusive use of open-source software, multi-platform support, flexible interface design regarding appearance and user controls and broadcast intermission functionality.
Five different technologies have been encountered and are compared against one another. The candidates consist of HTML5, Adobe Flash and the Red5 Media Server, the Darwin Streaming Server, VideoLan and the Java Media Framework. After evaluating each of these technologies, Adobe Flash and the Red5 Media Server protrude as the most promising software components for the implementation of the prototype, despite concerns regarding the multi-platform availability.

The goal of this thesis has been to develop and integrate a prototypical implementation of media streaming into Backstage. This prototype is composed of four distinct applications. The stream management interface is integrated into the lecturer's dashboard within Backstage as well as being available to seperate camera personnel as a standalone application. The lecturer's version provides special methods in order to allow cue points to be stored within the recorded media stream. These methods could be triggered whenever the lecturer changes the presentation slide. The audience's dashboard is equipped with an interface showing the lecturer and with further video panels capable of displaying streams of shared artefacts, such as the board, or a section of the lecturer's screen. To improve legibility, a magnifying tool is implemented in the video elements meant for transmission of shared artefacts.
In the future, there will be need for further development of the prototype that has been presented within this thesis. It is vital to add synchronisation mechanisms between the lecturer's and non co-located students' presentation slides. In addition, handheld devices must be taken into consideration, since there is high demand in a mobile version of Backstage. Enhancing the lecturer's awareness by accessing information regarding the students' streaming behaviour is a crucial prospective step. The prototype, further, needs to be technically improved, especially regarding performance and bandwidth usage. Finally, user studies are required to validate the previously mentioned concepts and ensure optimal user experience.

Appendix

A.1 VLC test runs

Several test runs were conducted, testing the streaming capability of the VideoLan VLC player. The test architecture consisted of a Dell M1330 laptop with a 2.5 Ghz Intel dual core processor and 4 GB memory (Computer A) and a Dell Precision Workstation with an 2.4 GHz Intel Quad Core and 8 GB memory (Computer B). VLC 1.1.5 was installed on both computers, which were connected over a 100 Mbit/s ethernet link. When streaming a webcam picture from Computer A to Computer B via RTP, a delay of 2 seconds occured, which increased to almost 15 seconds after 30 minutes of streaming. Transmitting in the opposite direction, i.e. from B to A, was equally delayed. Next, HTTP streaming was tested, where a consistent delay of 5 seconds was observed.
A second test architecture consisted of the same components, which, however, were linked over a 54 Mbit/s wireless connection. At first, the same delay could be seen, yet they increased to approximately 30 seconds after 20 minutes of streaming.

A.2 Implementation draft using HTML5

In the following, a prototype system is developed using the recent HTML 5 standard. The prototype is capable of native device support and broadcasting a media stream over the internet. The two clients to be implemented are HTML5/JavaScript web pages, the server consists of two Java classes. Due to a bug that prevents the processing of audio samples from a microphone within the browser, this implementation draft only deals with video that is captured from the default webcam.

The client for broadcast

The client the lecturer or the personnel who is responsible for capturing the stream uses varies slightly from the one available to the audience. In this example, only webcam streaming is to be described because of a bug in the implementation(Ref to CRbug). The prototype HTML5 layout is rather simple and basically consists of only two elements: a video element and a canvas element.

```
<html>
<div>
<video id="video" width="320" height="240" autoplay style="
    display: none"></video>
<canvas width="320" id="canvas" height="240" style="display:
    inline"></canvas>
</div>
</html>
```

Getting access to capture devices, such as the webcam, is done via the Media Stream API which is implemented in Opera 12+ and part of the webkit package used by Chrome. Because of these different packages, variations of the code have to be implemented for both cases. The variable `isWebkit` is used to store a boolean value that indicates whether Google webkit is used or not.
The `getUserMedia()` method offers native device support for both, audio and video. If the connection to the requested device is established, a seperate `success`-function is called and the media stream from that device is given as parameter. Next, the source of the video element in the document is set to the media stream which has been converted into a unique URL using `createObjectURL()`.

```
<script language="JavaScript">
var isWebkit = false;
var video = document.getElementById('video');

if(navigator.webkitGetUserMedia) isWebkit = true;

if(isWebkit){
    navigator.webkitGetUserMedia("video", onSuccess, onError);
}else{
        navigator.getUserMedia({video: true}, onSuccess,
            onError);
}


function onSuccess(stream){
 if(isWebkit){
        video.src = webkitURL.createObjectURL(stream);
 }else{
        video.src = URL.createObjectURL(stream);
 }
}

function onError(err){
// error handler
}
```

The next step is to create a timer which regularly captures the content of the video element and draws it on the canvas element. Here, every 10 milliseconds a new picture is taken from the video element and drawn to the canvas.

```
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');

timerCanvas = setInterval(function(){ ctx.drawImage(video, 0,
    0, 320, 240);}, 10);
```

This is necessary, so that a canvas frame can be grabbed and transformed into a DataURL, which is a unique identifier of a static object, such as a image. This way, also encoding takes place. In the prototype, JPEG encoding with a quality of 0.3 is used. Data is compressed enough to be transmitted over the Internet, but sufficient quality of the image is guaranteed. This process is executed by another timer, thus the effective frame rate is determined by this timer's set interval. The interval is set to 67 milliseconds, which equals 15 frames per second.

```
timerGrabFrame = setInterval(grabFrame, 67);

function grabFrame () {
var data = canvas.toDataURL('image/jpeg', 0.3);
}
```

In the bleeding edge approach, WebSockets are used to establish connections between the clients and the server via the WebSocket protocol that is based on TCP. The server address is given to the constructor of a WebSocket, which establishes the connection to the server automatically. As soon as the connection is established, the WebSocket is notified and the `onopen()` method called.

```
var ws = new WebSocket("ws://localhost:1234");
ws.onopen = function() {
                // action on open
}
```

Previously, a frame was grabbed from the canvas and transformed into a DataURL. In order to transmit data using WebSockets, so-called Blobs, which are byte representations of objects, have to be used. Blobs are created using the browser's BlobBuilder. However, to convert a DataURL to a blob, we need an auxiliary function. This function called `dataURLtoBlob` first converts the URL into a String of bytes. Then, the mime type is extracted. Next, an array buffer is created from the byte string. The single character of the byte string are put to the array buffer using its eight bit representation. Finally, a new BlobBuilder is generated, the array buffer is appended to the BlobBuilder and the mime type set.

```
var blobBuilder = (window.MozBlobBuilder || window.
    WebKitBlobBuilder || window.BlobBuilder);

function dataURLtoBlob(dataURL) {

    var byteString;
        if (dataURL.split(',')[0].indexOf('base64') >= 0){
        byteString = atob(dataURL.split(',')[1]);
        }else{
        byteString = unescape(dataURL.split(',')[1]);
        }

    var mimeString = dataURL.split(',')[0].split(':')[1].split(
        ';')[0]


    var ab = new ArrayBuffer(byteString.length);
    var ia = new Uint8Array(ab);
    for (var i = 0; i < byteString.length; i++) {
        ia[i] = byteString.charCodeAt(i);
    }


    var bb = new blobBuilder();
    bb.append(ab);
    return bb.getBlob(mimeString);
}
```

Whenever a frame is grabbed from the canvas, a new blob representation of the grabbed frame should be created and sent to the server. These functionalities are added to the `grabFrame` function.

```
function grabFrame () {
var data = canvas.toDataURL('image/jpeg', 0.3);
var newBlob = dataURLtoBlob(data);
ws.send(newBlob);
}
```

The server

The server handling an incoming media stream and broadcasting it to the connected clients is written in Java. The Jetty project offers libraries for server development which can be embedded in a Java project. The following packages of the Jetty project will be used in this server implementation.

```
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.server.nio.SelectChannelConnector;
import org.eclipse.jetty.websocket.WebSocket;
import org.eclipse.jetty.websocket.WebSocketHandler;
```

Naturally, the prototype server inherits from `jetty.server.Server`, which provides useful methods regarding handling socket connections. A `CopyOnWriteSet` is generated to manage all user connections and in the constructor of our class we establish a WebSocket connection as result of a `HttpServletRequest`. The `main` method eventually starts the server to listen on port 1234.

```
public class WebsocketServer extends Server {

        public final Set users = new CopyOnWriteArraySet();

        public WebsocketServer(int port) {

                SelectChannelConnector connector = new
                    SelectChannelConnector();
                connector.setPort(port);
                addConnector(connector);

                WebSocketHandler wsHandler = new
                    WebSocketHandler() {
                        public WebSocket doWebSocketConnect(
                            HttpServletRequest request, String
                            protocol) {
                                return new BroadcastSocket(
                                    users);
                        }
                };
                setHandler(wsHandler);
        }

        public static void main(String[] args) {
                WebsocketServer server = new WebsocketServer
                    (1234);
                try{
                server.start();
                server.join();
                }catch (Exception e){}
        }
}
```

A BroadcastSocket is returned for every http request from a client. It is given a reference to the global set of all users in order to register a new user when generated. The BroadcastSocket implements two interfaces, namely `WebSocket` and

`WebSocket.OnBinaryMessage`. Developing this bleeding edge prototype, it can be assumed that every binary message received by the server includes a media stream package sent from the broadcasting client and thus is forwarded to all connected clients.

```
private static class BroadcastSocket implements WebSocket,
    WebSocket.OnBinaryMessage {

        private Connection connection;
        private Set<BroadcastSocket> users;

        public BroadcastSocket(Set users) {
                this.users = users;
        }
```

As soon as a connection to the remote browser is established, a maximum size of binary data to be transmitted is set and the current BroadcastSocket added as new user to the existing ones. If the connection to a remote browser is lost, the socket is unregistered from the list of users.

```
public void onOpen(Connection connection) {
        this.connection = connection;
        this.connection.setMaxBinaryMessageSize(1024 * 512);
        users.add(this);
}

public void onClose(int code, String message) {
        users.remove(this);
}
```

As mentioned before, incoming binary data packages are assumed to be Blobs of grabbed images. A ByteArrayOutputStream is created from the incoming binary data and converted into a byte array which is directly forwarded to all registered users.

```
public void onMessage(byte[] data, int offset, int length) {

        ByteArrayOutputStream bytesOut = new
            ByteArrayOutputStream();
        bytesOut.write(data, offset, length);

        byte[] resultArray = bytesOut.toByteArray();

        for(BroadcastSocket user : users){
```

```
9
                try{
11                      user.connection.sendMessage(result, 0,
                            result.length);
                } catch (Exception e) { }
13      }
}
```

The listener client

This client is used by the audience and has the purpose of receiving and playing the media stream. This prototype offers video playback within an `img` element.
A WebSocket connection to the server is established in the very same way as it is done with the broadcasting client. Whenever the WebSocket receives a message, it can be assumed that it is a video frame in Blob format. Therefore, the byte representation can be converted into a DataURL and set as source for the image element.

```
<div>
2     <img id="displayArea" style="display: inline;"/>
</div>
4 <script type="text/javascript">
var isWebkit = false;
6 if (navigator.webkitGetUserMedia()) isWebkit = true;
var ws = new WebSocket("ws://localhost:1234");
8
ws.onmessage = function (msg) {
10
        var displayArea = document.getElementById('displayArea'
            );
12      if(isWebkit){
                url=window.webkitURL.createObjectURL(msg.data);
14              displayArea.onload = function() { window.
                    webkitURL.revokeObjectURL(url); };
        }else{
16              url= URL.createObjectURL(msg.data);
                displayArea.onload = function() { URL.
                    revokeObjectURL(url);     };
18      }
                displayArea.src = url;
20 }
</script>
```

A.3 Code snippets for flash development

The following code snippet demonstrates how access to a webcam can be achived in ActionScript. First, two objects, a video object and a camera object, are created. The capture mode of the camera is set to 320 pixels in width, pixels 240 in height and 15 frames per second. Next, the camera object is attached to the video object and the video object is added to the scene.

```
1 import flash.media.Camera;

3 var video:Video = new Video();
  var camera:Camera = Camera.getCamera();
5
  camera.setMode(320, 240, 15);
7 video.attachCamera(camera);
  addChild(video);
```

In a similiar manner, audio capture from a microphone is achieved.
Connecting to a server and transmitting a media stream, the classes `NetConnection` and `NetStream` have to be used. `NetConnection` establishes the connection to a remote server and `NetStream` generates a stream using this connection. Video and audio captured from local devices can be attached to the stream. The stream can also be published via parameters telling the server to either treat the stream as live broadcast or store it.

```
  netCon = new NetConnection();
2 netCon.connect("rtmp://localhost/stream");

4 outStream = new NetStream(netCon);
  outStream.attachVideo(camera);
6 outStream.publish("videoFromCam", "live");
```

In the follwing, flash applications for the stream management and broadcasting client as well as the students' client are presented.

A.4 Full implementation using Adobe Flash

Stream management

The capturing personnel's client consists of both GUI elements which are placed on the stage of the flash document and ActionScript code. The only difference between the embedded stream management client and the standalone application is the GUI elements for manual configuration of server address and session within the standalone client instead of accessing external configuration parameters. The code segment shows the ActionScript for the embedded client. The elements that are already placed on the stage are: `cam-drop` and `micdrop` (comboboxes for camera and microphone selection), `cam-check` (checkboxes to select camera to be broadcast), `startBtn` and `stopBtn` (buttons for starting and stopping broadcast), `stream-id` (labels for unique stream identifier). Note that the resolution for the third camera is 800px x 600px, while the others are set to 640px x 480px. This camera is dedicated to screen grabbing, which requires a high resolution in order to provide content that can be identified and read easily.
Furthermore, a cue points can be added to the stream using the `newCuePoint()` function. This function is made externally available, which means that it can be triggered by JavaScript code.

```
/** streamMgmt.fla development flash document for embedded
    flash client **/

// ++ SETUP GLOBAL OBJECTS ++ //

var serverAddress:String;
var streamID:String;
var appID:String;
var videoQuality:int;

var nc:NetConnection = new NetConnection();
var ns0:NetStream;
var ns1:NetStream;
var ns2:NetStream;

var cam0:Camera;
var cam1:Camera;
var cam2:Camera;
var mic:Microphone;

var vid0:Video = new Video();
var vid0Cont:Sprite = new Sprite();
var vid1:Video = new Video();
var vid1Cont:Sprite = new Sprite();
var vid2:Video = new Video();
var vid2Cont:Sprite = new Sprite();

```

```
28
// ++ SETUP STAGE ++ //
30 readParams();
setupStage();
32

34 // ++ FUNCTIONS ++ //


36

38 /* READ PARAMETRES/FLASHVARS FROM EXTERNAL CONFIG  */
function readParams():void{
40          serverAddress=root.loaderInfo.parameters.serverAddress;
                //get
            serverAddress
          appID=root.loaderInfo.parameters.appID;
                                    //get
            applicationID
42          streamID=root.loaderInfo.parameters.streamID;
                           //get StreamID from
            SessionID
          videoQuality=root.loaderInfo.parameters.videoQuality;
                //get
            videoQuality
44 }

46
/* CALL SETUP FUNCTIONS */
48 function setupStage():void{
          setupGUI();
50          setupVideoPanels();
          setupListeners();
52          runFailSafe();

54 }

56
/* SETUP GRAPHICAL USER INTERFACE */
58 function setupGUI():void {
          //camera drop down menus
60          cam0drop.addItem({label: "--select cam--", data:"-"});
          cam1drop.addItem({label: "--select cam--", data:"-"});
62          cam2drop.addItem({label: "--select cam--", data:"-"});

64          for(var i:int = 0; i < Camera.names.length; i++){
          cam0drop.addItem({label: Camera.names[i], data:i});
66          cam1drop.addItem({label: Camera.names[i], data:i});
          cam2drop.addItem({label: Camera.names[i], data:i});
68          }

70          //microphone drop down menu
          micdrop.addItem({label : "--select mic--", data:"-"});
72          for(var j:int = 0; j < Microphone.names.length; j++){
```

```
            micdrop.addItem({label: Microphone.names[j], data:j});
74          }

76          //labels
            stream0id.text = "presenter";
78          stream1id.text = "board";
            stream2id.text = "screen";
80  }

82
    /* SETUP EVENT LISTENERS*/
84  function setupListeners():void{
            nc.addEventListener(NetStatusEvent.NET_STATUS,
                statusHandler);
86          broadcastButton.addEventListener(MouseEvent.CLICK,
                broadcastClick);
            stopButton.addEventListener(MouseEvent.CLICK, stopClick
                );
88          cam0drop.addEventListener(Event.CHANGE, cam0select);
            cam1drop.addEventListener(Event.CHANGE, cam1select);
90          cam2drop.addEventListener(Event.CHANGE, cam2select);
            micdrop.addEventListener(Event.CHANGE, micSelect);
92
    }
94

96  /* SETUP VIDEO PANELS */
    function setupVideoPanels():void{
98          vid0.width = 320;
            vid0.height = 240;
100         vid0Cont.x = 10;
            vid0Cont.y = 50;
102         vid0Cont.addChild(vid0);
            addChild(vid0Cont);
104
            vid1.width = 320;
106         vid1.height = 240;
            vid1Cont.x = 345;
108         vid1Cont.y = 50;
            vid1Cont.addChild(vid1);
110         addChild(vid1Cont);

112         vid2.width = 320;
            vid2.height = 240;
114         vid2Cont.x = 10;
            vid2Cont.y = 335;
116         vid2Cont.addChild(vid2);
            addChild(vid2Cont);
118 }

120
    /* SELECT CAMERA FOR cam0 */
```

```
122 function cam0select(event:Event):void{
        if(cam0drop.selectedItem.data != "-"){
124     cam0 = Camera.getCamera(cam0drop.selectedItem.data);
        cam0.setMode(640,480,20,true);
                                            //normal camera
            resolution
126     cam0.setQuality(0, videoQuality);
        vid0.attachCamera(cam0);
128     }
    }
130

132 /* SELECT CAMERA FOR cam1 */
    function cam1select(event:Event):void{
134     if(cam1drop.selectedItem.data != "-"){
        cam1 = Camera.getCamera(cam1drop.selectedItem.data);
136     cam1.setMode(640,480,20,true);
                                            //normal camera
            resolution
        cam1.setQuality(0, videoQuality);
138     vid1.attachCamera(cam1);
        }
140 }

142
    /* SELECT CAMERA FOR cam2 */
144 function cam2select(event:Event):void{
        if(cam2drop.selectedItem.data != "-"){
146     cam2 = Camera.getCamera(cam2drop.selectedItem.data);
        cam2.setMode(800,600,12,true);
                                            //higher resolution
            for screen capture
148     cam2.setQuality(0, videoQuality);
        vid2.attachCamera(cam2);
150     }
    }
152

154 /* SELECT MICROPHONE */
    function micSelect(event:Event):void{
156     if(micdrop.selectedItem.data != "-"){
                mic = Microphone.getMicrophone(micdrop.
                    selectedItem.data);
158             mic.framesPerPacket = 1; // minimize latency
                mic.setSilenceLevel(0,2000); //set silence
                    level to 0
160             mic.gain=50;
        }
162 }

164
    /* NETWORK STATUS HANDLER */
166 function statusHandler(evt:NetStatusEvent):void {
```

```
        switch(evt.info.code) {
168     case 'NetConnection.Connect.Success':
            ns0 = new NetStream(nc);
170                 ns1 = new NetStream(nc);
                    ns2 = new NetStream(nc);
172                 beginStreaming();
                            break;
174     }
    }
176

178 /* BEGIN STREAMING TO SERVER */
    function beginStreaming():void{
180         var streamCount:int = 0;

182         if(cam0check.selected){
            ns0.attachCamera(cam0);
184         ns0.attachAudio(mic);
                                                                    //
                mic only attached to cam0
                !
            ns0.publish(streamID + "_" + stream0id.text, "append");
                        //"append" for pause
                functionality
186         streamCount++;
            }
188

            if(cam1check.selected){
190         ns1.attachCamera(cam1);
            ns1.publish(streamID + "_" +stream1id.text, "append");
192         streamCount++;
            }
194

            if(cam2check.selected){
196         ns2.attachCamera(cam2);
            ns2.publish(streamID + "_" + stream2id.text, "append");
198         streamCount++;
            }
200 }


202


204
    /* BROADCAST BUTTON CLICK HANDLER */
206 function broadcastClick(event:MouseEvent):void{
            nc.connect("rtmp://" + serverAddress + "/" + appID,
                true);      //connect to streaming
                server
208         broadcastButton.enabled = false;
            stopButton.enabled = true;
210         fscommand("streamEvent", "streamStart");
                                            //event trigger to
                javascript in website
    }
```

```
212

214 /* STOP BUTTON CLICK HANDLER */
    function stopClick(event:MouseEvent):void{
216         ns0.close();
            ns1.close();
218         ns2.close();

220         stopButton.enabled = false;
            broadcastButton.enabled = true;
222         fscommand("streamEvent", "streamStopped");
                                //event trigger to javascript in
                website
    }
224
    /* CHECK CONFIGURATION AND PREVENT MALFUNCTIONS */
226 function runFailSafe():void{
            if(serverAddress == null || appID == null || streamID
                == null || videoQuality == 0){
228                 videoQuality = 80;
            }
230 }

232 /* newCuePoint adds a cue point with slide number to the stream
        */
    function newCuePoint(slideNo:int):void{
234         var cueData = {slide: slideNo};
            ns0.send("@setDataFrame", "onCuePoint", cueData);
236 }
    ExternalInterface.addCallback("newCuePoint", newCuePoint);
                    //make newCuePoint() externally
       available
```

Shared artefact

Next, the code for the client being displayed in the area of the shared artefact is given. This client can be externally configured in order to be embedded more than once and display different streams. In addition, a layer mask is deployed in order to create a magnifying glass effect. The dimensions of the overlaying video object are twice the ones of the one that is placed below. The video is required to be streamed twice, since one `NetworkStream` can not be attached to two video objects. Further, two buttons - `startBtn` and `stopBtn` are added to the stage.

```
/** listener_shared_artefact.fla development flash document for
    embedded flash client **/

// ++ SETUP GLOBAL OBJECTS ++ //

var serverAddress:String;
var streamID:String;
var view:String;
var appID:String;
var nc:NetConnection = new NetConnection();
var ns:NetStream;
var ns1:NetStream;
var vid0:Video = new Video();
var vid0Cont:Sprite = new Sprite();
var vid1:Video = new Video();
var vid1Cont:Sprite = new Sprite();

// ++ SETUP STAGE ++ //
readParams();
setupStage();


// ++ FUNCTIONS ++ //


/* READ PARAMETRES/FLASHVARS FROM EXTERNAL CONFIG  */

function readParams():void{
        serverAddress=root.loaderInfo.parameters.serverAddress;
            //get
           serverAddress
        appID=root.loaderInfo.parameters.appID;
                                    //get
           applicationID
        streamID=root.loaderInfo.parameters.streamID;
                           //get StreamID from
           SessionID
```

```
34          view=root.loaderInfo.parameters.view;
                                        //get
                view
            setStartBtnLabel();
36  }

38

    /* WRITE startBtn LABEL */
40  function setStartBtnLabel():void{
            startBtn.label = "show " + view;
42  }

44

    /* SETUP STAGE */
46  function setupStage():void{
            runFailSafe();                                  //check
                configuration
48          stopBtn.buttonMode = true;              //stopBtn movie
                clip button in button mode
            stopBtn.visible = false;                //stopBtn
                invisible
50          setupVideo();                                   //call
                setupVideo function
            setupListeners();                               //call
                setupListeners function
52  }

54

    /* SETUP VIDEO OBJECT AND CONTAINER */
56  function setupVideo():void{
            vid0.width = 560;                       //video width
58          vid0.height = 420;                      //video heigh
            vid0Cont.x = 0;                         //position
                video x
60          vid0Cont.y = 0;                         //position
                video y
            vid0Cont.addChild(vid0);        //video in container
62          addChildAt(vid0Cont,0);         //add video container
                to stage below video1

64          vid1.width = 1120;
            vid1.height = 840;                      //video heigh
66          vid1Cont.x = 0;                         //position
                video x
            vid1Cont.y = 0;                         //position
                video y
68          vid1Cont.addChild(vid1);        //video in container
            addChildAt(vid1Cont,1);         //add video container
                to stage above video0
70  }

72

    /* SETUP EVENT LISTENERS*/
74  function setupListeners():void{
            nc.addEventListener(NetStatusEvent.NET_STATUS,
                statusHandler);  //network status
```

```
            handler
76          stopBtn.addEventListener(MouseEvent.ROLL_OVER,
                onButtonOver);   //stopBtn
                rollover
            stopBtn.addEventListener(MouseEvent.ROLL_OUT,
                onButtonOut);             //stopBtn
                rollout
78          stopBtn.addEventListener(MouseEvent.CLICK,
                stopStreaming);              //stopBtn
                click
            startBtn.addEventListener(MouseEvent.CLICK,
                startStreaming);    //showContentBtn
                click
80          stage.addEventListener(MouseEvent.MOUSE_MOVE, mGlass);
    }
82


84
    /* stopBtn HANDLER ROLL_OVER */
86  function onButtonOver(e:Event):void{
            e.currentTarget.gotoAndPlay("over");
88  }

90
    /* stopBtn HANDLER ROLL_OUT */
92  function onButtonOut(e:Event):void{
            e.currentTarget.gotoAndPlay("out");
94  }

96  /* stopBtn HANDLER CLICK */
    function stopStreaming(event:Event):void {
98          nc.close();
            vid0.attachNetStream(null);
100         vid0.clear();
                                      //clear video
                object
            vid0.visible = false;
102         vid1.attachNetStream(null);
            vid1.clear();
104         vid1.visible = false;
            startBtn.visible = true;
                                //show
                showContentBtn
106         event.currentTarget.gotoAndPlay("out");
            stopBtn.visible = false;
                                //hide
                stopBtn
108 }

110
    /* startBtn HANDLER CLICK */
112 function startStreaming(event:Event):void{
            nc.connect("rtmp://" + serverAddress + "/" + appID);
                           //connect to streaming server +
                application
```

```
114         stopBtn.visible = true;
                                                                    //
            stopBtn
            visible
        vid0.visible = true;
116         vid1.visible = true;
    }
118

120 /* START DISPLAYING VIDEO STREAM */
    function showVideoStream():void{
122         vid0.attachNetStream(ns);                               //
            attach video stream to video
            object
        vid1.attachNetStream(ns1);                                  //
            attach video stream to video
            object
124
        ns.play(null);
126         ns.play(streamID + "_" + view);
        ns1.play(null);
128         ns1.play(streamID + "_" + view);


130


132
        startBtn.visible = false;                                   //hide
            showContentBtn
134         glass.visible = true;                                   //
            magnifying
            glass
        vid1Cont.mask = glass;
136 }

138
    /* NETWORK STATUS HANDLER */
140 function statusHandler(evt:NetStatusEvent):void {
        switch(evt.info.code) {
142     case 'NetConnection.Connect.Success':                       //
            connection
            successful
                    ns = new NetStream(nc);
144                 ns1 = new NetStream(nc);
                    showVideoStream();
146                 break;
        }
148 }

150
    /* MAGNIFYING GLASS */
152 function mGlass(e:Event):void {

154         vid1Cont.x = (mouseX * -1);
            vid1Cont.y = (mouseY * -1);
```

```
156
            glass.x = (mouseX);
158         glass.y = (mouseY);

160         //Mouse.hide();

162 }

164
/* CHECK CONFIGURATION AND PREVENT MALFUNCTIONS */
166 function runFailSafe():void{
            if(serverAddress == null || appID == null || streamID
                == null || view == null){
168                 startBtn.label = "Wrong Config";
                    startBtn.enabled = false;
170         }
}
```

Presenter

Last but not least, the client playing the video stream which is meant to be shown in the awareness column is implemented. Because it is configurable externally, a fail safe method is added to guarantee robustness in case of wrong configuration. For future development, the NetStream is already configured in such a way that it calls a function whenever cue points are reached in the stream.

```
// ++ SETUP GLOBAL OBJECTS ++ //

var serverAddress:String;
var appID:String;
var streamID:String;
var view:String;
var nc:NetConnection = new NetConnection();
var ns:NetStream;
var vid0:Video = new Video();
var vid0Cont:Sprite = new Sprite();


// ++ SETUP STAGE ++ //
readParams();
setupStage();


// ++ FUNCTIONS ++ //


/* READ PARAMETRES/FLASHVARS FROM EXTERNAL CONFIG  */

function readParams():void{
        serverAddress=root.loaderInfo.parameters.serverAddress;
            //get
           serverAddress
        appID=root.loaderInfo.parameters.appID;
                                    //get
           applicationID
        streamID=root.loaderInfo.parameters.streamID;
                           //get StreamID from
           SessionID
        view=root.loaderInfo.parameters.view;
                                    //get
           view
}


/* SETUP STAGE */
function setupStage():void{
        runFailSafe();                                  //check
           configuration
```

```
        stopBtn.buttonMode = true;              //stopBtn movie
            clip button in button mode
37      stopBtn.visible = false;                //stopBtn
            invisible
        setupVideo();                                   //call
            setupVideo function
39      setupListeners();                               //call
            setupListeners function
}
41

43 /* SETUP VIDEO OBJECT AND CONTAINER */
function setupVideo():void{
45      vid0.width = 240;                       //video width
        vid0.height = 180;                      //video heigh
47      vid0Cont.x = 0;                         //position
            video x
        vid0Cont.y = 0;                         //position
            video y
49      vid0Cont.addChild(vid0);        //video in container
        addChildAt(vid0Cont,0);         //add video container
            to stage
51 }

53
/* SETUP EVENT LISTENERS*/
55 function setupListeners():void{
        nc.addEventListener(NetStatusEvent.NET_STATUS,
            statusHandler);   //network status
            handler
57      stopBtn.addEventListener(MouseEvent.ROLL_OVER,
            onButtonOver);   //stopBtn
            rollover
        stopBtn.addEventListener(MouseEvent.ROLL_OUT,
            onButtonOut);             //stopBtn
            rollout
59      stopBtn.addEventListener(MouseEvent.CLICK,
            stopStreaming);              //stopBtn
            click
        startBtn.addEventListener(MouseEvent.CLICK,
            startStreaming);    //showContentBtn
            click
61
}
63

65 /* stopBtn HANDLER ROLL_OVER */
function onButtonOver(e:Event):void{
67      e.currentTarget.gotoAndPlay("over");
}
69

71 /* stopBtn HANDLER ROLL_OUT */
function onButtonOut(e:Event):void{
73      e.currentTarget.gotoAndPlay("out");
}
75
```

```
77 /* stopBtn HANDLER CLICK */
   function stopStreaming(event:Event):void {
79         nc.close();
           vid0.attachNetStream(null);
81         vid0.clear();
                                        //clear video
              object
           vid0.visible = false;
83         startBtn.visible = true;
                              //show
              showContentBtn
           event.currentTarget.gotoAndPlay("out");
85         stopBtn.visible = false;
                              //hide
              stopBtn
   }
87

89 /* startBtn HANDLER CLICK */
   function startStreaming(event:Event):void{
91         nc.connect("rtmp://" + serverAddress + "/" + appID);
                        //connect to streaming server +
              application
           stopBtn.visible = true;
                                                                  //
              stopBtn
              visible
93         vid0.visible = true;
   }
95

97 /* START DISPLAYING VIDEO STREAM */
   function showVideoStream():void{
99         vid0.attachNetStream(ns);                          //
              attach video stream to video
              object
           ns.play(streamID + "_" + view);          //show
              presenter stream
101        startBtn.visible = false;                          //hide
              showContentBtn
   }
103

105 /* NETWORK STATUS HANDLER */
   function statusHandler(evt:NetStatusEvent):void {
107    switch(evt.info.code) {
       case 'NetConnection.Connect.Success':                  //
          connection
          successful
109               ns = new NetStream(nc);
                  ns.client = {
111                       onMetaData:function(obj:Object):void{
                             trace(obj);},
```

```
                        onCuePoint:function(obj:Object):void{
                            requestExternalSlide(obj.slide); }
113                         }                       // cue points
                            trigger the request of a lecture
                            slide being displayed
                    showVideoStream();
115                 break;
        }
117 }

119
    /* CHECK CONFIGURATION AND PREVENT MALFUNCTIONS */
121 function runFailSafe():void{
            if(serverAddress == null || appID == null || streamID
                == null || view == null){
123                 startBtn.label = "Wrong Config";
                    startBtn.enabled = false;
125         }
    }
127
    /* CALL EXTERNAL JAVASCRIPT FUNCTION TO DISPLAY SLIDE */
129 function requestExternalSlide(slideNo:int):void{
            fscommand("slideChange", slideNo);
131 }
```

A.5 Development of Flash applications using Adobe Flex

The prototype as part of this thesis was created by the use of Adobe Flash, an proprietary authoring environment. However, Adobe offers an open source framework called Flex SDK for compiling ActionScript into Flash Applications. The main difference is the exclusive use of ActionScript when developing with Flex. User controls, such as buttons and input fields, are added and placed by calling constructor functions instead of placing them on the visual interface manually. The ActionScript source code which is listed in the previous section of the appendix represents the core functionalities of the system and can be integrated into a comprehensive ActionScript file. This file is then compiled by the use of a command line tool ("mxmlc"), which is part of the Flex SDK [1].

[1] http://livedocs.adobe.com/flex/3/html/help.html?content=compilers_13.html

List of Figures

2.1 Technology-independent system architecture 8
2.2 Structural layout of Backstage 12
2.3 Integration of streaming components into Backstage 13

4.1 Comparision of open-source technology suitable for this thesis 21

5.1 Flash Plugins request permission to use media devices. 26
5.2 Stream management interface 27
5.3 ManyCam supports screen grabbing and video files as input for simulating a camera. 27
5.4 Video panel of the presenter on the students' dashboard 28
5.5 The board as a shared artefact with a magnifying tool for improved legibility. 29
5.6 Shared artefact demonstrating the ability of screen grabbing and the option to disconnect from the stream. 30

Bibliography

[1] F. Bry, V. Gehlen-Baum, and A. Pohl, *Introducing Backstage - a digital backchannel for large class lectures*, 2011.

[2] H. M. Eidenberger and R. Divotkey, *Medienverarbeitung in java*, 1 ed., dpunkt.verlag, 2004.

[3] A. S. Tanenbaum, *Computer networks*, 4 ed., Prentice Hall, 2002.

[4] Edward Primeau, *Digital Video Formats - Codecs, Compression and CCTV Video*, http://ezinearticles.com/?Digital-Video-Formats--\discretionary{-}{}{}Codecs-Compression-and-CCTV-Video&id=6609430/.

[5] Ryan Paul, *Ogg Theora vs H.264: head to head comparisons*, http://arstechnica.com/information-technology/2010/02/ogg-theora-vs-h264-head-to-head-comparisons/.

[6] Auvisoft, *Audio Formats Overview*, http://www.auvisoft.com/audio_format.htm.

[7] J. Loviscach, *A Real-Time Production Tool For Animated Hand Sketches*, Proceedings of the 9th European Conference on Visual Media Production (CVMP), 2011.

[8] Rich Newman, *User Interface Design For Business Applications*, http://richnewman.wordpress.com/category/tabbed-document-interface/.

[9] D.C. Burnett and A. Narayanan, *W3C - Media Capture and Stream*, http://dev.w3.org/2011/webrtc/editor/getusermedia.html.

[10] *The Chromium Projects - Web Platform Status*, http://www.chromium.org/developers/web-platform-status.

[11] *Web specifications support in Opera products*, http://www.opera.com/docs/specs/productspecs.

[12] I. Fette and A. Melnikov, *The WebSocket Protocol*, http://tools.ietf.org/html/rfc6455.

[13] *Jetty HTTP Web Server*, http://jetty.codehaus.org/jetty.

[14] *getUserMedia() doesn't work for microphone input*, http://code.google.com/p/chromium/issues/detail?id=112367, [Online; accessed 28-May-2012].

[15] _____, *Adobe Flash Professional*, http://www.adobe.com/en/products/flash.html.

[16] _____, *Supported codecs | Flash Player*, http://helpx.adobe.com/flash/kb/supported-codecs-flash-player.html.

[17] Adobe, *ActionScript 3.0 Developer's guide*, http://help.adobe.com/en_US/as3/dev/as3_devguide.pdf.

[18] D. Winokur, *Flash to Focus on PC Browsing and Mobile Apps; Adobe to More Aggressively Contribute to HTML5*, http://blogs.adobe.com/conversations/2011/11/flash-focus.html.

[19] _____, *Adobe AIR 3*, http://www.adobe.com/de/products/air.html.

[20] _____, *ActionScript 3.0 overview*, http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html.

[21] _____, *Real-Time Messaging Protocol (RTMP) specification*, http://www.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp_specification_1.0.pdf.

[22] _____, *Red5 project website*, http://www.red5.org.

[23] Red5, *Red5 Documentation*, http://trac.red5.org/wiki/Documentation.

[24] Media College, *Cue Points in Flash Video*, http://www.mediacollege.com/flash/video/cuepoints.html.

[25] E. Wolff, *Spring 3*, 3 ed., dpunkt.verlag, 2010.

[26] MacOS Forge, *Darwin Streaming Server Project*, http://dss.macosforge.org/.

[27] Apple, *QuickTime Streaming Server*, http://www.apple.com/de/quicktime/extending/resources.html.

[28] _____, *VideoLan Streaming* , http://www.videolan.org/vlc/streaming.html.

[29] VideoLan, *Streaming features list*, http://www.videolan.org/streaming-features.html.

[30] ManyCam LLC, *ManyCam for Windows and Mac*, http://www.manycam.com.

[31] A. Pohl, F. Bry, J. Schwarz, and M. Gottstein, *Sensing the classroom: Improving awareness and self-awareness of students with backstage*, 2012.

[32] <swfobject>, *An easy-to-use and standards-friendly method to embed Flash content*, http://code.google.com/p/swfobject/.

[33] F. Bry, *erlebt - Die Computer unserer Studenten*, http://www2.pms.ifi.lmu.de/erlebt/?p=7061.