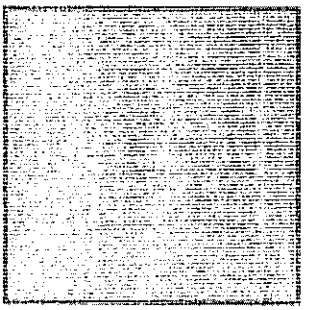
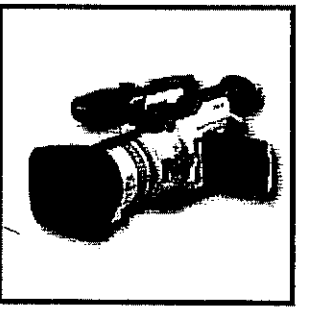
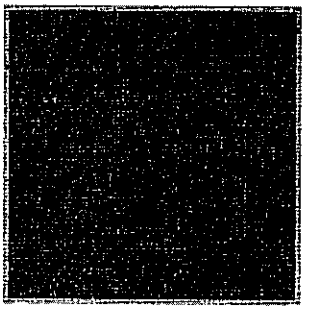
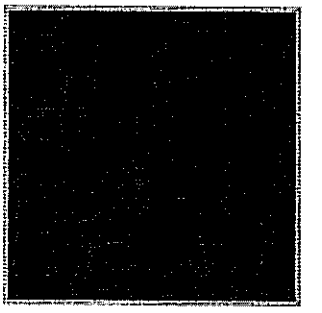


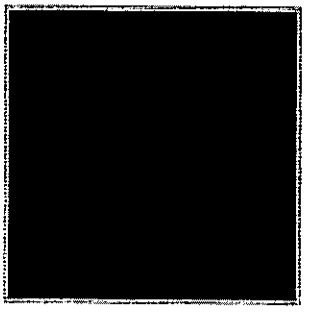
Digital Lectures





A Media Processing Framework and Interface Evaluation





Allister Cournane

University of Canterbury, 2003

Supervisor: Dr. T. Bell

Abstract

Lectures are an environment where a number of different forms of communication exist such as speech, writing and projected images, much of which is currently lost or poorly recorded. Digital Lecture systems have been developed to capture these events, but there is little published research regarding the media processing systems, and previous work has established the need for research into the interfaces used by students.

This research proposes the initial design and implementation of a media processing framework aimed at supporting the low-cost capture of lectures with a flexible and generic architecture utilising stream indexers, media compressors and an interface generator. Seven guidelines for interface development are proposed, and a prototype interface has validated six of these. An interface evaluation was carried out and resulted in a participant suggestion that students be permitted to create their own index points in the interface. Implementation of this feature would change the way the system is used in courses, and the way the framework indexers were originally designed to behave. Further research is recommended to evaluate the learning implications of such a facility.

Content

1. Introduction

"The traditional lecture remains a defining element of most university courses" [7]. It is an environment where a number of different forms of communication exist such as speech, writing and projected images, much of which is currently lost or poorly recorded [3]. Developments over the past decade have allowed this issue to be addressed via the application of computing technology to "facilitate automatic capture, integration and access to multimedia information in the educational setting of the university classroom" [3]. The systems that facilitate this capture process are referred to as *Digital Lecture* systems, and encompass the processes of lecture capture, processing, eventual distribution and student access via distribution mediums such as CD, DVD or the Internet.

While it is not uncommon for students to bring recording devices to lectures such as portable tape recorders and minidiscs, the lack of indexing reduces the usefulness of the recording, as rapid access to points of interest is limited to the use of *fast-forward* and *rewind* facilities. Digital Lecture systems aim to produce indexed output with flexible delivery mechanisms that can aid students interested in revision, or who were unable to attend a lecture [4]. The output of Digital Lecture systems may be based on data captured from multiple devices such as video cameras, microphones, and electronic copies of documents.

The types of Digital Lecture systems vary depending on the event being recorded, and the technique used to capture it. Systems can be effectively classified using the two-dimensional taxonomy illustrated in Figure 1 (example systems in the figure are discussed further in section 2). Events to be captured may be classified as *structured* or *unstructured*. Structured events include lectures and formal presentations, where some predefined event organisation has been established. Contrastingly, events such as meetings or tutorial discussions are classified as unstructured, as there may not necessarily be an established speaker or presentation structure [2]. Further, the methods used to capture the event may be classified as *invasive*, or *passive* (uninvasive). While invasive approaches have been successfully used in systems such as Classroom 2000, they constrain the teaching style of the lecturer, forcing them to teach within the limitations of the capture hardware and software of the system [19]. Passive approaches do not impose such restrictions and instead adapt to the environment and teaching style of the lecturer.

Distance learning systems [17, 18, 22, 33] share similarities with Digital Lecture systems in their capture and delivery of multimedia using mediums such as the Internet. However, the system architectures and communication modes differ. Figure 2 illustrates the modes of communication possible. Distance learning architectures incorporate lecturer-student interaction, chat and collaboration facilities [18], and other realtime interaction features to allow a student to experience a lecture in the same manner as a student in a theatre. Such systems are generally classified as two-way/synchronous in Figure 2. Digital Lecture systems are typically one-way/asynchronous due to random temporal access facilitated by index points, and the nature of student revision being asynchronous to the captured lecture event.

		Environment	
		Unstructured	Structured
Capture	Invasive	*CSCW Systems*	*Classroom 2000*
	Passive	*Experience on Demand*	*Cornell Lecture Browser*

Figure 1: Digital Lecture System Taxonomy (from [19])

Communication Type	Communication direction: One-way	Communication direction: Two-way
Asynchronous	*Static webpages, CDs, Videos, Streaming of pre-recorded video*	*Mail, E-mail, fax, online forums*
Synchronous	*Radio/TV broadcasts, realtime streaming of video*	*Instant messaging, chat rooms, audio/video conferencing, interactive TV*

Figure 2: Communication modes (from [18])

The phases of lecture capture can be likened to those of movie production with phases of *pre-production*, *production* and *post-production* [3]. The pre-production phase involves the organisation, setup and synchronisation of capture equipment. If invasive capture methods are utilised, presentation materials such as slides to be projected are entered and processed by the system during this phase. The production phase involves the use of capture hardware to record all relevant activity that will be useful to students in later review. Generally, aspects to be captured include what is seen, what is said, and what is written down. Again, the recording methods used vary according to the capture methods in use. Non-invasive approaches emphasise an unintrusive approach toward the presenter, and aim to capture lectures without altering the teaching style of the speaker, while being organised to transparently support wide a range of teaching styles [4]. Intrusive methods encourage the delivery of lectures according to installed hardware and processing software. The final phase is post-production and involves the processing of captured material to generate output useful for students to review. Because student access is ongoing once the lecture is captured, this phase has no endpoint in principle [2].

This report presents a framework designed to support the capture of lectures in a flexible, low-cost and low-impact manner. The report also focuses on the development and evaluation of an access interface used by students to view the material created by the framework.

The concept and philosophies of Digital Lecture systems have been introduced in section 1. A number of Digital Lecture systems have been developed previously, and those most relevant to this research are introduced and discussed in section 2. The motivations driving the research into the development of a framework are formally presented in section 3, and the objectives of the research are stated.

The development of the Media Processing Framework is discussed in detail in section 4, providing an overview of the requirements established for the framework, its design and aspects of its implementation.

One of the outputs of the framework, the access interface is discussed in section 5. The design of a prototype interface is presented and elements of its design are justified according to guidelines proposed as part of this research. An evaluation of the interface is presented, and the results discussed.

A brief discussion of further applications of the framework are outlined in section 6, including the generalisation of the framework for use in applications outside of a lecture environment, analysis of its limitations, and future directions for its continued development.

The concluding section of this report contains a summary of the research presented.

2. Related Systems

The concepts presented in section 1 have been realised in a number of Digital Lecture systems developed as both research and commercial ventures. Those systems most closely related to this research are presented in this section, specifically the Classroom 2000 project, the Berkeley and Cornell University Lecture Browsers, and some other related work.

2.1. Classroom 2000

The Classroom 2000 project (first presented by Abowd [3] in 1996) was a project aiming to "facilitate automatic capture, integration and access of multimedia information in the educational setting of the university classroom." Dividing the capture process into three phases (pre-production, live recording and post-production), the Classroom 2000 project initially developed a number of prototype systems to establish which technologies would best support the capture process, including technology to record notes by students. A classroom specifically designed to support the capture of lectures was constructed (see Figure 4) in 1997 consisting of:

- Embedded ceiling microphones and video cameras
- An electronic whiteboard (LiveBoard) to digitally record lecturer annotations
- Two ceiling projectors
- Electrical power and network access ports for each student seat.

The system captured lectures in the media forms of audio, video, student and lecturer notes. Each stream was processed using a proprietary multimedia processing system named *StreamWeaver*, the output of which was a series of HTML pages to be viewed by students in a standard web browser with plugins such as Quicktime managing presentation of video or audio. Sample webpage output is illustrated in Figure 3 with slides augmented by lecturer notes on the right.

Significant findings after three years of use were published [2] after the system was used to capture the lectures of over 60 courses. Several points and recommendations were made:

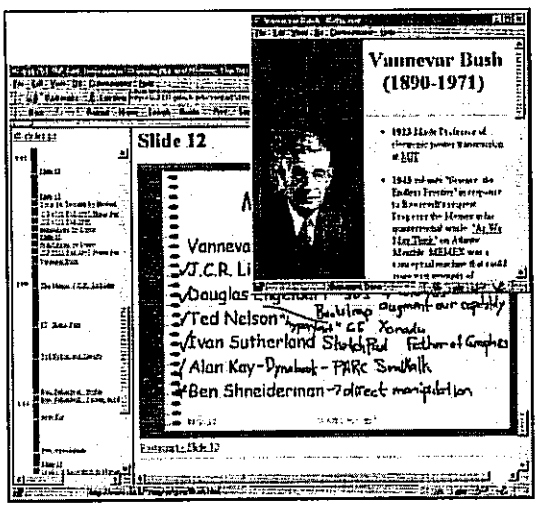


Figure 3: StreamWeaver webpage output

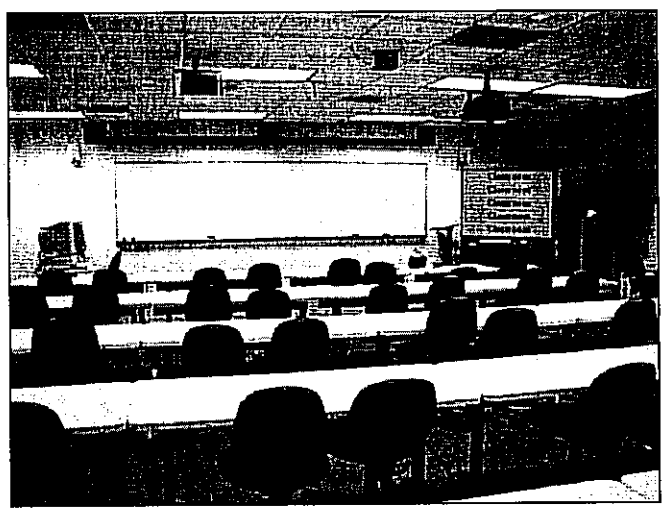


Figure 4: Classroom 2000 Theatre

- Students had requested the system be extended to support searches for topics across all recorded lectures in a course, or even across multiple courses.
- Much flexibility in generation of the access interfaces is necessary. Static access interfaces are not recommended.
- Evaluations of the access interface are important, and require further research.
- Development of flexible capture frameworks are necessary which can evolve to support more streams as technology makes them available.

2.2. Berkeley Lecture Browser

The Berkeley Media Research Centre (BMRC) Lecture Browser [9] is a web-based tool written in Javascript designed to web cast large class lectures on the internet. While the system utilises a media processing system, many of the tasks must be carried out manually and are yet to be automated. Specifically, generation of slide index points, labelling of individual slides, and compression of video and audio are tasks carried out by a human operator.

The interface for the system incorporates an embedded media player, graphical timeline, presentation of slides, and a slide index facility to support seeking with slide-granularity. A noted limitation of the system is the poor image and sound quality due to the selection of the internet as the distribution medium.

The Lecture Browser system has also been extended to support realtime display of lectures using multiple streams broadcast by the Berkeley Internet Broadcasting System (BIBS). The system allows viewers to simultaneously watch the lecturer as well as presentation tools such as OHPs and data projectors in separate windows.

Research related to the production of transcripts from the audio track is in progress, and the production of a system to facilitate course-wide searching of archived lectures is under investigation.

2.3. Cornell University Lecture Browser

The Cornell University Lecture Browser is based on a non-invasive capture system utilising multiple video cameras. One camera uses hardware tracking technology to maintain a head-and-shoulders shot of the lecturer, while the second camera captures a wide angle view of the theatre. The system automatically detects changes in slides based on analysis of video images using algorithms based on comparisons with slides uploaded by the speaker at the conclusion of the lecture.

The focus of this research is on the technical challenges in creating a system to automatically detect slide changes, editing between camera views and synchronisation establishment between the individual camera setups. Experiments with actual lectures resulted in slide change detection accuracy of 97.2%. The implemented interface for the lecture browser is a web-based interface with an embedded RealMedia player.

2.4. Other related work

The AutoAuditorium [8] project has been used to broadcast presentations to remote audiences while simultaneously recording them to tape. Multiple cameras are utilised in a totally automated system

incorporating editing based on context. The system utilises an invasive approach and requires that presentations be delivered in an appropriately equipped theatre. The output of the system is a single taped presentation that must be viewed sequentially. No attempt is made to structure the presentation or include non-video data sources. Since its conception in 1999, the project has evolved into a commercial product[1].

STREAMS [10] is another prototype system for recording lectures and creating a presentation for later viewing. Passive capture techniques are utilised, and video and audio streams are generated. Unlike the AutoAuditorium project, the STREAMS system does not use automatic editing to assemble a single stream from the multiple camera views, based on the premise that the *viewer* should choose which stream to view. The system stores all streams, and presents the viewer with low resolution versions of each camera view. The viewer can then choose a preferred stream and view it in full resolution. This approach is disputed by Mukhopadhyay et al. [19] where the authors point out that pushing the editing task on the viewer distracts them from the topic of the presentation.

The Canterbury Digital Lectures Project [4, 7] involved both research into the technical issues surrounding the capture and indexing processes, and the social impact of the system over a four-month period with first year university students. The research approach involved 'wizard-of-oz' strategy to simulate elements of the system which were not implemented (specifically, the automatic indexing component) and observations regarding student use of the captured lectures was also noted. The research concluded that providing too much flexibility to students can have negative learning results. By providing a 'digital surrogate' for lectures, the researchers found that students made less effort to attend the lectures, and that they never fulfilled their intention to catch up using the digital media.

The Experience-on-Demand project [30] uses speech, image and natural language processing combined with GPS information to capture and communicate personal experiences. The capture method used by the system is non-invasive (passive) and attempts to capture everyday tasks using audio, video, and spatial information, with each stream being merged into a single unit representing the "experience." The system allows users to annotate their experiences, with voice recognition software providing a transcript which can be used to access the captured material by keyword.

[1] http://www.autoauditorium.com/

3. Research Goals

The concept of a Media Processing Framework is introduced in this section, and motivations driving its design and development as part of this research are established in section 3.1. In addition to this, the motivations defining the need for interface evaluations are discussed, and the objectives of the research are established in section 3.2.

3.1. Motivations

While other lecture browsers exist, little published research exists regarding the development of a common processing framework. Many Digital Lecture systems use proprietary processing systems—examples include Classroom 2000's StreamWeaver, or the stream processor component of the AudioAuditorium project. These systems have been designed and implemented to function specifically for a given setup, and as pointed out by Abowd [2], the need to update and reengineer portions of the systems to accommodate new data streams—such as stream output from multiple video cameras, student PDAs, or electronic whiteboards—is often problematic.

The design of the interface is important, since this is the only portion of the Digital Lecture system students actually see and experience. While student surveys indicate a general acceptance of Digital Lecture systems [2, 3], further interface generation and HCI-related research has been recommended [2] in order to better accommodate student requirements. Once established, the interface requirements of the students can drive the overall processing system to produce output best matched to their needs.

Further, a number of browser technologies have emerged in the past few years such as new CSS (Cascading Style Sheet) standards, enhanced flexibility in embedded media player application support and the introduction of standardised technologies such as SMIL; all of which have applications in the interfaces of Digital Lecture Systems. These technologies can be applied to create interfaces that are richer and more flexible than the interface technologies used in systems such as Classroom 2000 or the Cornell Lecture Browsers, which have their roots in browser technologies of the late 1990s.

3.2. Objectives

The approach of this research differs from other published work primarily by its focus to address the problems of inflexible processing systems by the construction of a generalised media processing framework. The proposed framework architecture is designed to support the low-cost capture of lectures, simplify the incorporation of future data streams, and minimise the time and effort required to maintain framework modules. Additionally, this research addresses the interface issues raised by Abowd [2] in the establishment of a set of interface design guidelines, accompanied by the evaluation of a prototype interface generated by the framework.

Formally, three general research goals have been established:

- The construction of a low-cost, flexible media processing framework composed of pluggable components to support evolution of technology and simplify the integration of new data streams into the Digital Lecture system.
- The establishment of general guidelines to aid in the design and implementation of a student-access interface.
- The design and implementation of an access interface, and its subsequent evaluation to assess the validity of the proposed guidelines and to augment the framework to better support student requirements.

Evaluation of the learning and social implications of this system are not part of the research scope, and research regarding such issues has been published by Abowd et al. [2, 3] and Bell et al. [7].

4. Media Processing Framework

As outlined in section 3.2, one of the objectives of this research is the development of a flexible processing framework. The research decisions made when planning the framework are discussed in section 4.1, and the framework requirements established in section 4.2. The basic design of the framework is introduced, and discussed in detail in sections 4.3 thru 4.8. Aspects such as the framework's audio, video, document processing, and interface generation capabilities are described, and their implementation is presented in section 4.9.

4.1. Introduction

One aspect which is to be considered during the design of a Digital Lecture processing a framework is the types of media—or input *streams*—the framework must support. A 'stream' may be output from a capture device such as a video camera, electronic whiteboard, or may take the form of an electronic document such as PowerPoint slides. The input from which a Digital Lecture processing system produces its output could conceivably be any 'capturable' artefact of lectures considered to be a stream.

One problem faced initially in this research is that framework needed to be flexible, but also a low-cost solution to lecture capture. At one extreme, a "blanket" approach could be used where support for all conceivable types of input would be considered in order for additional input to be incorporated in the future with the use of pre-built processing facilities. Such an approach would require minimal system changes to incorporate new streams, but overall has several drawbacks. Firstly, such an undertaking would require ready access to all types of capture hardware that could generate input used in the system—such as tablet PCs, wireless PDAs and electronic whiteboards—some of which may never be used. Analysis of the data streams generated by these types of hardware would require significant investment and time. Secondly, the future evolution of hardware may change to the extent that they could render current development redundant or unusable. For example, video storage has matured in the past decade from standard VHS cassettes to digital tape storage, both having differing format specifications [1] and processing requirements. Incorporating processing facilities for as-yet undeveloped hardware is not feasible, and it cannot be assumed that current processing techniques will suffice for future technology. Furthermore, not all input to a Digital Lecture system is hardware-captured [3]; documents, for example, are a common artefact of lectures and any blanket approach would require processing support for all conceivable document formats.

An alternative approach may be to develop a simple system that meets very specific needs, and would be designed to process a very narrow range of input such as one video stream and one audio stream. While such an approach is effective in the short term, it may not be effective in the long term as "early decisions made in organising a large system greatly impact the longevity of the system developed" [2]. A small system supporting specific types of input may meet current needs, but the complexity of the modifications and maintenance required for the system to support new streams may limit its future usefulness.

A compromise between these two approaches is presented in this research. The presented system is a framework is neither a blanket, nor highly specialised approach to digital lecture capture. The proposed framework does not require pre-built support for all hardware, but is sufficiently flexible to allow new

data streams to be added as necessary with minimal framework changes. The architecture presented in this section is the result of an initial development iteration cycle and includes support for three basic types of input streams—video, audio and documents.

4.2. Framework Requirements

A driving philosophy of the framework is the understanding that the system must not be designed under the assumption that certain types of input will never be used, even if, at the time of the framework's development, certain hardware was deemed too expensive to consider its support. With this philosophy in mind, the requirements presented in this section were established for the framework.

4.2.1. Modularity

To remain flexible, the framework must incorporate specialised processing modules, each with well-defined inputs, outputs, and processing requirements. For example, a specialised task one module may perform is compression of an audio stream. The input of such a module would be one uncompressed audio stream, and required output would be a stream of bits in a specific compression format (such as MP3 or AAC).

The justification for this modularised requirement is to allow the architecture to evolve to support new streams (by including new modules), and also to modify the processing of existing streams (by altering the behaviour of existing modules). The flexibility offered by a modularised design allows individual components to be altered without adverse affects on the overall architecture. For example, incorporation of a new audio compression algorithm simply requires altering the framework's audio compressor module to use the new algorithm. Such a change would ideally have minimal impact on the architecture, and few (if any) other components would need modification.

4.2.2. Basic media support

Support for the most basic media types must be incorporated into the initial framework design. Specifically support for processing of lectures from streams based on:

- audio only
- still image sequences with audio
- full motion video with audio

must be supported in order to create a functional prototype.

4.2.3. Output of indexed material

The system should output indexed material, otherwise it would just be a simple playback system. Indexed streams are vital to the usefulness of a Digital Lecture system, and the framework must be capable of parsing and performing analysis of streams based on heuristic algorithms to identify the occurrence of important events, and to output these events in a manner that allows students to seek to their occurrence in the lecture.

4.2.4. Multi-platform support

Support for platform independent output should be available, and is important to allow students to access the system using diverse platforms such as Linux, MacOS or Windows. The framework must support the output of interfaces in a platform independent format, such as a web- or java-based interface.

4.2.5. Non invasive capture

The capture process is considered part of the framework, and part of any Digital Lecture system. The method selected for this research is the non-invasive approach.

As introduced in section 1, non-invasive (or passive) capture methods must not require the lecturer(s) to alter their presentation methods, wear any additional equipment, or force any increase in workload before, during, or after a lecture. Further, the system must not be intrusive to students attending lectures, such as installing cameras obstructing the students' view, or requiring that lights remain on during the lecture to accommodate video camera light level requirements.

This requirement affects the entire framework, as optimal recording scenarios may not be assumed. For example, computer vision-based components cannot assume an ideal view of the projection screen, and must operate based on footage captured from a variety of angles, such as the side of the theatre, or with students in frame. Likewise, the system must adapt to record sound with or without a lecture sound system, and must not impose requirements on unwilling lecturers such as having to wear a microphone. However, the quality of the recorded lecture can be improved in situations where lecturers express a willingness to support the capture process.

4.2.6. Low cost

Taking a different approach to some of the systems presented in section 2, the approach of this research is the construction of a system to support lecture capture with minimal costs. Such an assumption has a benefit of not requiring the existence of a lecture theatre equipped with expensive embedded video cameras, microphones or projectors. Instead, the system assumes the use of low cost portable devices which may (or may not) be used in the capture of a lecture. At a minimum, a microphone may be the only device used to capture a lecture. The Thunderwire research by Hindus et al. [11] illustrates that a system containing high quality audio is often sufficient to establish a telepresence and an effective learning environment. Thus even a simple system will be useful, and the availability of more media will create an even richer experience.

4.2.7. Capture location independence

Capture independence allows the system to utilise equipment present in the theatre (such as existing projectors, sound systems, and whiteboards) in addition to portable recording equipment transported to the location. This requirement effectively forces the system to (within reason) *adapt* to any lecture room environment, including its natural stage layout, student seating, lighting conditions, or noise levels.

Likewise, the system must not assume that certain hardware will always be available in a theatre. For example, while audio systems and projectors may be installed, they may not always function or a lecturer may choose not to use them. The framework must be capable of producing output from lectures captured at locations where only one or two streams are available (such as audio only, or slides only).

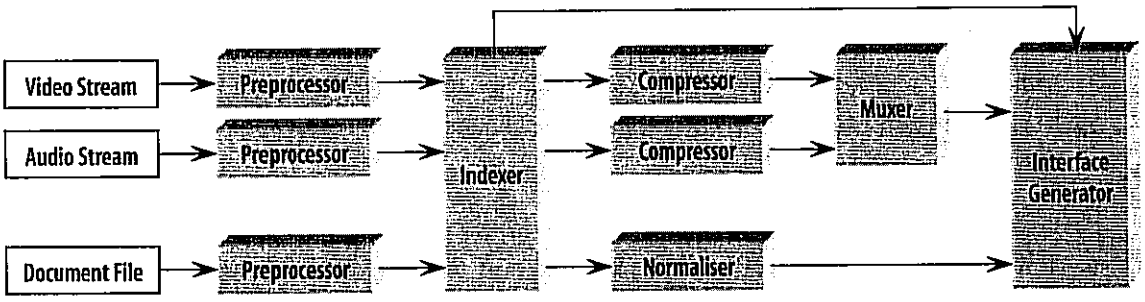


Figure 5: Architecture overview

4.3. Baseline Architecture Design Overview

The goal of this research is not the complete development a comprehensive processing architecture for lecture processing. Instead, the research has focused on the development of the concept of a framework which can be implemented to support capture of (primarily) lectures, but also other types of events.

Illustrated in Figure 5 is a high-level abstraction of a *baseline* framework architecture design, capable of supporting the three basic media types established in section 4.2.2. Stream inputs enter on the left, are processed, with final output by the *interface generator* module on the right. In general, each raw input stream—audio, video or documents—is pre-processed, indexed, post-processed, and sent to the interface generator, which will output an interface (such as an HTML page) to allow users to access the captured material. The activities involved in each processing stage are specific to the type of stream.

The architecture outlined in the remainder of section 4 is very much the initial iteration of the framework design which may mature significantly in the future. As will be discussed, not all modules offer a complete or comprehensive set of features, but instead offer a proof of concept. Each module is discussed with an emphasis on the technical challenges associated with its design and function. Due to limitations on time and research scope, the framework implementation supports only a limited range of input formats, compression schemes and output formats. Justification for the selection of each of these is provided. However, the *design* of the framework is intended to be scalable, and in the future, modules are expected to expand to incorporate a more extensive set of file formats, compression schemes and indexing techniques.

4.4. Indexing Overview

The process of indexing is encapsulated in the Indexer module of the processing framework, and as established in section 4.2.3, the task performed by this module is an essential function of the framework.

Due to the projected complexity of its implementation, the design ideas presented in this section have not been realised in a fully functional Indexer module. Utilising the 'wizard-of-oz' approach of Bell et al. [7], the indexing process was manually executed in this research, allowing for time to be devoted to other framework implementation aspects. However, the absence of a functional indexer module does not affect the framework, as human intervention provided the output which would have otherwise been generated by the module. It should be noted, however, that a functional document indexer was developed and fully implemented in this research.

The remainder of this subsection presents a general overview of the tasks this module has been designed to perform.

Internally, the indexer is designed to be composed of several sub-indexer components, each specialised in analysis of one of the input streams (see Figure 6). Details about indexing techniques specific to video, audio and document streams may be found in later sections of this document. Sub-indexers can operate autonomously, but the results of each are collected by a central indexer *controller* which may correlate the results of multiple streams to identify high-level events (see section 4.4.1).

While this module has inputs and outputs, no modification of the original stream is carried out. This module is a read-only observer, and overall executes two major tasks:

1. Time-based event identification
2. Keyword identification

The output of the indexer module is a list of time-based index points and a list of keywords. The output is used by the interface compiler. For example, the list of keywords may be used in searches and the time-based index points may be visibly presented as a graphical timeline in the interface.

4.4.1. Time-based Event Identification

Time-based event identification is the process by which semantically important events are extracted from

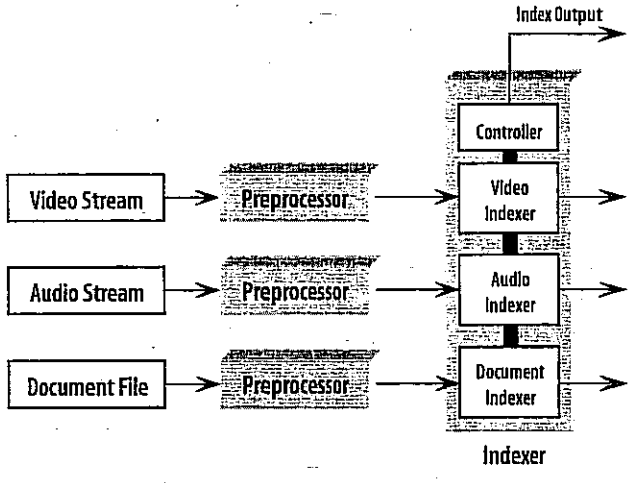


Figure 6: Indexer module design

the input streams. In the case of lectures, important events to be indexed may include:

- Slide or OHP transparency changes
- Change of speakers (such as a student asking a question)
- Change of topic
- Practical demonstrations
- Change of media (between slides, transparencies, or the whiteboard)

Similar to *chapter stops* on DVDs[2], index points take the form of timestamps and can be used in the access interface to allow users to navigate the captured lecture. For example, index points may be presented as a list, allowing students to skip to the presentation of a particular topic, or a point when a specific slide was discussed. The absence of index points significantly reduces the usefulness of a Digital Lecture system to a simple playback system without support for users with specific information needs. In such cases, the only available browsing tools are the "fast forward" and "rewind" features, leading to an inefficient information retrieval system [23].

Identification of events may require correlation of one, or all input streams to establish the occurrence of the event. For example, detection of a slide change is a visual event, and the video stream alone may be used to determine this. However, detection of changes in topic is a complex event, and may require analysis of gestures and facial features (from video), correlated with changes in speaker pitch, or silence (from audio). Similarly, algorithms designed to extract and correlate features from video and audio streams can be used to determine speaker identities [24, 28, 29]. Such algorithms can utilise voice analysis and facial feature recognition methods to identify speaker change events within 2% error margins [24].

The indexer in the framework can incorporate these algorithms into the *controller* sub-component which may use the video and audio sub-indexers to perform primitive event detection, and cross reference these to establish whether a semantically significant high-level event has occurred. Even if no correlations can be established between streams, the primitive events identified by each sub-indexer may be used as independent index points. Further, if *no* index points can be established, the indexer could default to the generation of index points every n seconds. While only moderately useful, the generated index points could be used to extract thumbnails from the video for display, allowing the user to visually determine whether an significant event as occurred.

All events identified by the Indexer module are compiled into a timestamped list, with one entry per event, potentially with some semantic description of the event type such as "speaker change," "slide change," or "student question".

4.4.2. Keyword Extraction

While time-based event identification extracts a list of times when important events occur, generation of a meaningful description may be difficult from time-based event identification alone. For example, while

[2] http://sanyolaserproducts.com/dvd/glossary.htm#chapter_stop

a change of slide may be detected, the system may be unable to determine a meaningful *name* for the new slide. Further, searches of captured material are extremely difficult without the construction of a searchable index of human-readable terms associated with a lecture. Hence, a second component has been designed into the Indexer module of the framework to address these issues.

Keyword extraction is the process of extracting important words and phrases from the input streams. Documents are an important source of keyword identification, and documents in ASCII format may be parsed to directly extract all text to be converted into a keyword list. Alternative technologies such as OCR (optical character recognition) may be used to recognise and extract text from images of documents such as scanned OHP transparencies [32], including recognition of handwriting with high levels of accuracy [12]. These techniques are further discussed in section 4.7.2.

Keywords may also be extracted from video images using textual region isolation algorithms which identify segments of video frames containing text, and passing these segments to an OCR program for text identification [14].

Extracted keywords are a valuable resource used in aiding searches of captured lectures for specific terms mentioned in lecture handouts, or keywords spoken by a lecturer. Students may use these indexes to carry out powerful searches on anything a lecturer has spoken, handed out, or drawn on a whiteboard.

Indexed terms may also be used to automatically associate streams with documents. For example, cross referencing words spoken by a lecturer with their own slides may provide an additional cue for the system to detect slide changes or references to other handouts.

4.5. Video Processing

The video processing component of the framework spans a number of modules, and is directly associated with the processing of footage captured by video cameras, and also still-image digital cameras. The design of the video processing aspect of the architecture (as illustrated in Figure 5) has been subdivided into five stages:

1. Video capture
2. Preprocessing
3. Indexing
4. Compression
5. Muxing

The last four stages are self-contained tasks, and exist as separate modules in the framework (see Figure 5). Stage one (video capture) is a *process* rather than a module, but is nevertheless an important part of the framework. Each of the five stages are briefly discussed in the remainder of section 4.5.

4.5.1. Video Capture

The video capture phase of the architecture involves the active capture of lectures with digital/analog video and still-image cameras, and transferring the captured data to disk for processing.

The goal of the framework is to support a low-cost solution to the capture of lectures, hence the system has been designed to support input from typical handheld or tripod-mounted video cameras. Typical video formats used by modern digital video (DV) cameras include DV, DVCAM and Digital8[3] which produce video streams at rates of 25 megabits at 25 or 30 frames per second, requiring approximately 10 gigabytes of storage per hour of footage.

Incorporating the non-invasive requirement (see section 4.2.5) into this phase requires that the cameras be used to capture the lecture from positions that do not distract students in lectures.

While initially the framework can assume that this phase is executed manually, it is feasible for this task to be entirely automated. The aforementioned AutoAuditorium project [8] uses an automated camera capture system. Research with a focus on motion tracking in a wide-angle situation has also been undertaken using multiple, precisely-aligned cameras [26]. The images from each camera are 'stitched' together to create a single panoramic image, providing sufficient coverage to ensure the presenter is always in-frame. Computer analysis of the panoramic image is used to produce a cropped ROI (region of interest) from the original wide image, analogous to the creation of a "pan 'n' scan" version of a movie originally filmed in a wide aspect ratio format.

More advanced research related to motion tracking was published by Microsoft which researched the use of multiple computer-controlled cameras (via a 'virtual' director PC) to record presentations [15]. The system required virtually no human intervention, with the editing of video streams from each of the cameras being managed by the virtual director in real time according to pre-programmed editing rules from experts.

Both of the techniques described above could be incorporated into the framework's video capture component by the design of a video capture module which could automatically control cameras. However in the interests of maintaining a low-cost approach, the scope of the framework is presently limited to processing video footage which has already been extracted from the video camera.

An alternative form of input to the system may be a series of still images. Recording snapshots up to four times the resolution of a DV camera, images captured from a still-image camera have an implicit index structure—the time the image was taken may signify an important event, especially if the camera were human operated. Still images may be converted to video by reprinting the images to create a video sequence.

The output from this phase is usually one video file to be processed by the architecture.

4.5.2. Video Preprocessing

The captured video (see section 4.5.1) may often require some preprocessing to synchronise it with a video stream, improve its visual appearance during playback, indexing accuracy, and compression quality. In addition to synchronisation, the preprocessing module may execute one, or all of the following procedures:

- Format normalisation

[3] http://www.adamwilt.com/DV-tech.html#Details

- Noise removal
- Exposure control
- Brightness enhancement
- Image sharpening
- Deinterlacing
- Resizing

Synchronisation with video is important to maintain consistency between what the user sees and hears. Synchronisation may be achieved through a number of simple methods. Used before a lecture commences, a device generating a brief flash (in view of all cameras) combined with an audible tone [19] is an inexpensive method. Histogram analysis on video frames may be used to isolate a frame with a bright signal, and all frame data before this point may be discarded. If a similar process is executed on the audio track(s), the two should be synchronised. See section 4.6.2 for more information on the design of the audio pre-processor.

Since the output from this module is passed directly to the indexer, it must be normalised to ensure the indexer may function independently of the original video format. Specifically, a number of colour schemes are used to record video such as YUV family[4], and the RGB family[5], each differing in their ordering and compression of pixel components. The video indexer component (see section 4.5.3) has been designed to receive RGB32 input, and format conversion may be necessary to ensure all frames output from the pre-processor are in this 32-bit colorspace.

Each of the image manipulation tasks above may be executed as small *filters* within the video preprocessor module, each of which performs specialised transformations on each frame of video. Each filter has configurable options to control the degree to which each video frame is analysed and manipulated before being passed to other modules in the framework for further processing.

Removal of unimportant data before compression significantly increases the performance of video compression algorithms [13]. Noise removal algorithms attenuate spurious noise and small features, reducing the complexity of a video sequence, resulting in video frames that are easier to encode and less prone to errors such as blocking, ringing and temporal flicker. Hence, noise removal is a task which has been incorporated into the design of the video preprocessor module of the framework.

Exposure control and brightness enhancement are luminance control filters that ensure the image is neither too dark, nor too bright. In lecture theatres the light levels are often low, and the use of overhead or data projectors cause a bright source of light resulting in very high contrast areas in video. To ensure contrast levels remain within a reasonable range, these two filters have been included in the preprocessor design to alter levels per-frame as necessary to maintain an acceptable exposure and brightness balance.

Image sharpening filters have been incorporated into the design of this module to enhance image edges which may improve recognition of characters (during indexing), or the general image quality.

---

[4] http://www.fourcc.org/fccyuv.htm
[5] http://www.fourcc.org/fccrgb.htm

Deinterlacing is the process of removing scanlines[6] inherent with video footage generated by many video cameras to reduce artifacts such as crawl, twitter and flickering [27]. While more expensive video cameras are now capable of generating progressively scanned images (non-interlaced), it was felt that deinterlacing support should be incorporated into the video preprocessor to allow for input from less expensive handheld video cameras. Simple field blending or complex motion estimation algorithms [27] may be used to remove the artifacts mentioned above and will assist in any video image analysis performed in the indexer module by producing a clean, undistorted image.

The final feature included in the preprocessor design in video image resizing. The standard video image size from PAL-based cameras is 720x576 pixels, and 720x480 from NTSC-based cameras[7]. Resizing the video to a smaller resolution such as 640x480 or 512x384 is often required in Digital Lecture systems in order to allow other interface items to be displayed on screen with the video.

4.5.3. Video Indexing

Algorithms and techniques specific to *visual* identification of events are outlined in this section. The concepts mentioned are part of the design of the video indexing component of the Indexer (see section 4.4). This component has been designed to receive normalised frames in RGB32 colour space, as this eliminates the need for complex processing of *luma* and *chroma* components[8] of the YUV family of pixel representation.

As discussed in section 2.3, simple analysis of video frames to detect slide changes is a viable indexing approach that could be used in this system. The technique involves one video camera pointed permanently at projector screen, and detection of slide changes using a two-stage process [19]. The first stage uses an inter-frame comparison algorithm to detect whether a slide change has occurred. Frames are thresholded to black and white pixels, and the percentage change in pixels compared between frames—a threshold difference signalling slide change index point. If a change is detected, the second stage is invoked to determine which slide is in view. A general shape-matching algorithm is used to compare video frame images to a set of slides known to the indexer.

Additionally, clustering techniques such as those published by Rui et al. [23] may be incorporated into the video sub-indexer. Clustering is based on the premise that visually-similar frames of video have a high probability of being associated with the same event. A significant change in visual similarity indicates an event. The algorithm involves assessing and assigning a *visual similarity factor* to sequential frames in video and 'clustering' similar frames into groups. Clustering techniques can be used to detect changes in light levels, slides, or significant movement of a lecturer in the frame.

---

[6] *Interlaced scan* has been an essential part of TV and video coding schemes for many years and involves splitting captured frames into two *fields*—one consisting of even-numbered scanlines (0, 2, 4, ...), and the other consisting of odd-numbered scanlines (1, 3, 5, ...), each captured approximately 1/50th of a second apart. Resulting video frames thus contain motion captured at two different instants.

[7] http://www.sonopress.de/spec_eng/vdse003.pdf

[8] In the YUV colorspace there is one component that represent lightness (luma) and two other components that represent color (chroma). As long as the luma is conveyed with full detail, detail in the chroma components can be reduced by subsampling (filtering, or averaging) which can be done in several ways (thus there are multiple formats for storing a picture in YUV colorspace).

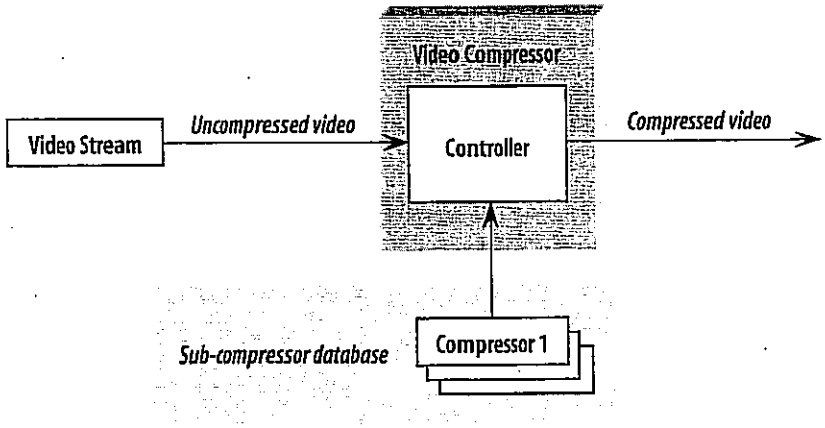


Figure 7: Video Compressor module design

4.5.4. Video Compression

In order to create a manageable video stream, compression is required to reduce the video sizes from over 10 gigabytes to less than 0.5 gigabytes.

The design of the Video Compressor module is illustrated in Figure 7. To allow for multiple compression algorithms to be used, a database was created composed of sub-compressor modules. The *Controller* component in the Video Compressor module uses one of the sub-compressor modules as a dynamic 'plug in' to actually carry out the compression task, and the specific compressor to be used may be defined by the framework (depending on characteristics of the video), or by the user. This design allows new compression algorithms to be used in the future by implementing and adding a new compressor to the database.

The initial design of the framework includes two sub-compressor modules in its database, and both are based on the MPEG-4 compression standard. Standardised in 1999, the MPEG-4 video compression standard incorporates new error resilience components with more efficient compression of images and video [6]. The standard allows for near-DVD image quality at a fraction of the bitrate required to achieve the same quality in older compression schemes such as MPEG-2 or MPEG-1. The DivX compression scheme from DivX Technologies[9] is a commercial implementation of the MPEG-4 standard, and the XviD[10] compression scheme[11] is a free, open-source implementation of the standard. Selected because of their high image quality, each compressor can produce differing file sizes depending on characteristics in the video. Experiments have shown that in shots of high motion, XviD may produce higher quality images, while DivX appears to have a higher tolerance to video noise. Both are capable of producing compressed video sizes of under 500 megabytes at 640x360 resolution[12], and have been included in the initial design of the Video Compressor module to allow for higher flexibility in the selection of the compressor to use based on input video characteristics.

[9] http://www.divx.com

[10] XviD is *DivX* reversed, and is likely a tongue-in-cheek reference to the commercial alternative.

[11] http://xvid.sourceforge.net

[12] Based on 60 minutes of lecture time with 16:9 recording (1.78:1 aspect ratio).

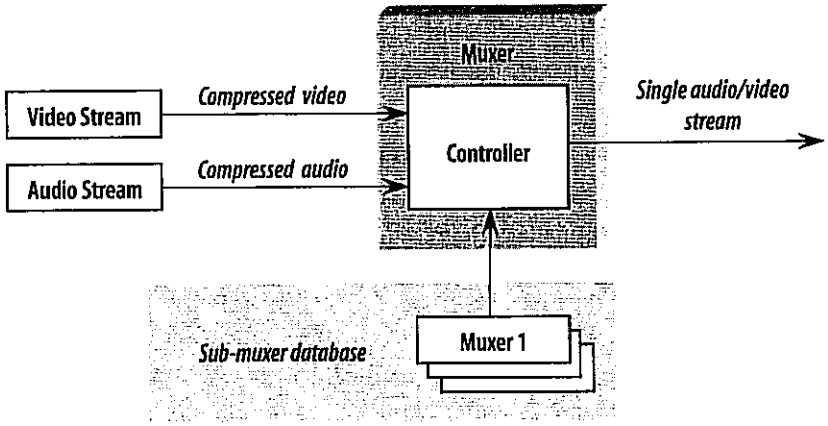


Figure 8: Muxer module design

4.5.5. Muxing

The output from audio and video compressors cannot generally be saved directly to file, and an intermediary process is involved before their output may be written in a format readable by media players. Muxing or *multiplexing* is the process or merging zero or more video streams with zero or more audio streams to create a single file. The file containing these merged streams is called a *container*, and contains data from all streams and also metadata such as an index table and authoring information. Examples of container formats include the Quicktime MOV format, the MP4 format and the Windows AVI format.

The structure of the Muxer Module in the framework is illustrated in Figure 8, and is similar to the Video Compressor module design in its use of a sub-module database. In this instance, the purpose of the database is to allow for different output file formats to be used (such as those listed above). The choice of output file format is important, as the destination platform must be capable of reading and decoding the file. For example, the Quicktime and MP4 formats are relatively platform independent, but the Windows AVI format is not.

The module has been designed to carry out two major tasks

- Multiplexing of one or more streams
- Stream hinting

The multiplexing process involves some dependence between the muxer and both compressors (audio and video) as this module requires a knowledge of the nature of the compressed streams in order to successfully merge samples from each without corruption. Because of this, muxers are developed with support for specific video and audio standards in mind. During playback, the multiplexing process is reversed and the stream is split back into its independent audio and video components for decompression (if necessary) and playback. Thus, the multiplexed file format simply acts as a transport medium for the streams, and hence the name *container*.

In order for the framework to successfully function, each video and audio compressor must have at least one associated muxer capable of reading and multiplexing its output. With this requirement in mind, the MPEG4IP muxer[13] was incorporated into the initial design of the module as a sub-muxer. Created as a standards-complaint, open-source system for creating MPEG-4-based files, the MPEG4IP software supports multiplexing of the output of both video compressors outlined in section 4.5.4, and the audio compressors in section 4.6.4. The output format is standards-compliant MP4—a format that has demultiplexers (for playback) under Windows, MacOS and Linux. The MPEG4IP muxer is also capable of producing files which can be read by Apple's Quicktime 6 player .

It is important to note that the absence of one of the streams (audio or video) will not cause difficultly in the muxing process. Likewise, the presence or more than two streams is also supported by most multiplexers, allowing for multiple audio or video streams to be present in a single file.

The second task carried out by the muxer module is *hinting*. Typically executed during the muxing process, hinting adds additional data to the file to aid in streaming such as precomputed packet boundaries, or video/audio timing and synchronisation data [21]. Hinting is necessary only if the output file may be distributed from a streaming server such as Apple's Darwin Streaming Server[14], which utilises hinting information in multimedia files. The MPEG4IP muxer supports the creation of hint tracks in the output MP4 file.

Support for new file formats is possible by the inclusion of new muxer sub-modules, such as an AVI multiplexer for Windows-specific output, or the platform-independent Matroska format based on EBML[15] (currently under development at http://www.matroska.org).

4.6. Audio Processing

Audio processing is separated into five stages in a similar way to video processing.

1. Audio capture
2. Preprocessing
3. Indexing
4. Compression
5. Muxing

4.6.1. Audio Capture

Digital Lecture audio is usually captured from two main sources: a lecture sound system or portable recording devices.

Use of an embedded theatre sound system with a microphone worn by the speaker has the advantage of receiving the best possible signal of the lecturer's voice, however much of the ambient noise of the lecture is lost because the microphone is positioned on the speaker, and not near the audience. The presence of

[13] http://mpeg4ip.sourceforge.net
[14] http://developer.apple.com/darwin/projects/streaming
[15] A binary derivative of XML

ambient noise in the Digital Lecture system can help in a faithful reproduction of the event being recorded. Further, microphones worn by the speaker typically produce only single-channel (mono) output (for example, audience questions and reactions). Informal tests carried out during this research revealed a mono recording of the lecturer was subjectively unnatural to listen to for the full duration of a lecture (approximately 50 minutes), especially without the presence of ambient noise. This is consistent with results published by Hindus et al. [11] stating that low quality audio created formality and prevented the ear from clearly discerning environmental sounds.

Portable sound devices have the flexibility of being positioned anywhere in the theatre, and since the recording devices are typically very small, they may be effectively placed anywhere without becoming a distraction. Microphones exist for portable devices that are capable of two-channel (stereo) recording, and the use of stereo recording was found to have significant advantages over mono recordings, specifically in the capture and reproduction of ambient noise during the event. However, since the recording device is statically placed in the theatre before the lecture begins, the recorded quality of the speaker's voice may vary depending on their distance from the microphone. Placement of the microphone between the lecturer and the students has been found to give an ideal balance between capture of the speaker's voice and general ambient noise.

Recording from a lecture sound system typically involves using available recording facilities, or connecting a recording device to system line-out jacks. The output from this phase may be a one or two-channel recording of the audio. Sampled at 44.1KHz, uncompressed mono recordings will typically generate files of 220Mb, and stereo recordings 440Mb for 50 minutes.

4.6.2. Audio Preprocessing

The main task of the audio pre-processor module in the framework is to synchronise the audio with the video. As described in section 4.5.2, audio tones may be used to synchronise an audio track with video. The technique has been successfully used by Mukhopadhyay et al. [19]. By searching for a sharp rise in volume at the beginning of the audio stream, the occurrence of a tone may be located and used as a synchronisation point.

4.6.3. Audio Indexing

Audio indexing is executed by the audio sub-indexer component of the framework, as illustrated in Figure 6. The input to the indexer is a raw, uncompressed audio bitstream as captured using the methods described in section 4.6.1.

The track may be indexed using voice recognition applications, however limitations exist with use of conventional software. The need for training is a significant drawback to the use of standard voice recognition tools, as is the limited vocabularies available in the systems. The problem is further complicated by the fact that some lectures may contain non-vocal content such as music, or significant ambient noise. Segmentation of the audio stream into vocal and non-vocal sections may address some of these issues and has been demonstrated in the CueVideo system from IBM [5] where voice recognition was applied only to vocal segments. Application of standard voice recognition software in realistic lecture experiments has yielded 70-80% accuracy, but unreadable transcripts [2].

Indexing techniques presented by Young et al. [34] bypass the problem of limited speech recognition vocabularies by the use of a language-independent phonetic-based indexing using a tree-based segmentation algorithm. However, the technique requires significant processing in order to generate the tree for spoken terms, and also to search indexed audio streams.

Non content-based methods may also prove useful in the audio indexer and may be incorporated into its design as a less-complex alternative. Changes in speaker may be valid index points, such as the occurrence of a student question. Speaker recognition based on audio has been successfully implemented in research published by Saraceno et al. [24]. Detection of silence is also a valid indexing method and requires a simple analysis of volume levels in the audio stream. Silences may indicate changes in topic, a lecturer using a whiteboard, distributing new student handouts, or changing transparencies on an OHP.

4.6.4. Audio Compression

The Audio Compressor module analyses an uncompressed audio stream and uses a selected compression scheme to reduce the audio bitrate while maintaining as much of the original audio quality as possible. The structure of this module is similar to that of the Video Compressor discussed in section 4.5.4. A number of sub-modules are available and specialise in a number of compression schemes.

Compression of audio has progressed significantly over the past few years with the extensive use of compressed audio formats such as MP3, and more recently, AAC. The choice of compression schemes in the design of the Audio Compressor is important to preserve audio fidelity and platform independence.

While one of the most popular sound formats across all platforms, the MP3 compression scheme is nearly a decade old and since its initial development many advances in perceptual audio coding and compression have been achieved[16]. Advanced Audio Coding (AAC) has been standardised as part of the MPEG-4 standard from Dolby Labs, Fraunhofer, AT&T, Sony, and Nokia. The scheme uses new perceptual compression schemes, subjectively achieving higher quality audio at lower bitrates than MP3, as concluded from experiments carried out by Dolby Labs. While still a new compression scheme, AAC compression and playback has been incorporated into Quicktime 6 from Apple[17] and free compressors are available for Windows and Linux from http://faac.sourceforge.net.

Based on the popularity of MP3 and the improved quality and availability of software for AAC, both standards have been included as sub-modules in the design of the Audio Compressor. Incorporation of new or updated algorithms requires only modification of the appropriate sub-module. The relatively immature nature of the AAC standard has resulted in many releases of compression software as new techniques are discovered to implement the standard. Because of this, it is important that the framework cope with constant replacement of audio compression modules with new versions. The current design supports this requirement.

Receiving an input stream of at the rate of approximately 1 megabit per second, the output is typically between 96 and 192 kilobits per second—reduced to 10–20% of the original bitrate. Variable bitrates are supported by both MP3 and AAC, allowing the bitrate to constantly change at the discretion of the

[16] http://www.apple.com/mpeg4/aac/

[17] http://www.apple.com/quicktime/products/qt/

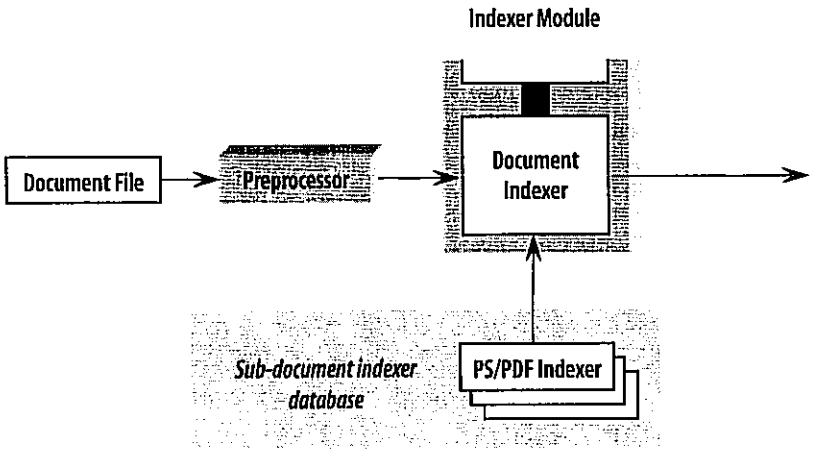


Figure 9: Indexer module design

compression algorithm. Estimation of the size of output is therefore difficult as it is dependent entirely on the nature of the sound file being compressed.

4.6.5. Muxing

The process of muxing is discussed in section 4.5.5. The dependence between the compressor output format and muxer input format applies also to audio formats. The MPEG4IP muxer incorporated into the Muxer Module design supports the multiplexing of both MP3- and AAC-compressed streams.

4.7. Document Processing

Document processing is separated into the stages of preprocessing, indexing and normalisation. For the purposes of the framework a *document* can be loosely defined as a directory of images (such as images of slides or scanned transparencies), or a single file (such as a PDF).

4.7.1. Document Preprocessing

Preprocessing allows portions of a document to be submitted to the framework for processing, such as a range of pages in a PDF, or only specific files in a directory. This feature is required, for example, when a lecture does not cover all pages in a handout and only certain portions need to be submitted to the system for analysis.

While this has been illustrated in Figure 5 to exist as an independent module for clarity, it is foreseeable that this process could be incorporated into the indexing component to simplify implementation.

4.7.2. Document Indexing

The main function of the document indexer is to extract keywords from a document. Because of the great variability in the type of documents that may be submitted to the system for processing, the document indexer component uses an architecture similar to that used in the Video Compressor and Audio Compressor modules discussed in sections 4.5.4 and 4.6.4, respectively. Illustrated in Figure 9, a database of specialised indexer components exists to allow processing and keyword extraction from effectively any type of document. Examples may include PDF, PostScript, Latex, PowerPoint and Image indexers.

Image indexers are required to process scanned images of transparencies, or may even be used if no parser can be developed to process a file. In such a situation, the file may be *rasterized* into pixels (using screenshots, for example) and indexed as if it had been scanned from paper. OCR is an effective technique for keyword extraction from raster documents, and algorithms specialised in recognising characters in non optimal conditions (such as when text is printed against greyscale backgrounds) have been implemented [32]. Indexing scanned transparencies containing handwriting is also possible using word segmentation approaches such as that proposed by Manmatha et al. [16], or shape matching approaches as published by Kameshiro et al. [12] which achieved 88% accuracy in experiments.

Keyword extraction from PostScript and PDF documents is not straightforward as while words of a report usually appear as plain text within a PostScript file, they are intermixed with PostScript language commands and internal data [31]. A robust technique used in the New Zealand Digital Library[18] has been proposed by Nevill-Manning et al. [20] which redefines the *show* PostScript operators to intercept page content, allowing it to be 'rendered' as ASCII text, rather than as pixels. Using some heuristic rules, fragments of text can be reassembled into words and line breaks detected. The technique has been successfully used on over 30,000 documents, working equally well with PDF files—a close cousin to PostScript—without any modifications.

Incorporated into the design of the document indexer is an component capable of keyword extraction from PostScript and PDF documents. These two formats were considered to offer the most flexibility, as many other document formats may be converted into PDF using tools such as Adobe Acrobat Distiller[19], or printed directly to PostScript from the source application. However, since the indexer architecture uses a pluggable structure, indexers for images and native PowerPoint and Latex files may be added in the future.

It is important to note that the absence of an indexer for any particular type of file will not prevent the framework from functioning—the result will simply be that the document passes directly through the module without any processing, and no keywords from the document will be available for the system to use when constructing an interface.

4.7.3. Document Normalisation

The normalisation process involves normalising a document into a format supported by the interface, similar to the normalisation carried out by the video pre-processor to ensure all frames input to the indexer are in RGB32 format (see section 4.5.2). For example, if the interface supports display of slides, the normaliser would be responsible for ensuring all output is in the form of images of a specific size, in a specific format. Documents not already in an image format would require rasterization, and those already rasterized may require format conversion or resizing.

Alternatively, if the interface supports display of slides as a PDF, the normaliser would be responsible for ensuring the output is in the form of a PDF document, possibly through the use of conversion tools such as Adobe Acrobat Distiller, if necessary.

---

[18] http://www.nzdl.org

[19] http://www.adobe.com/products/acrobat/

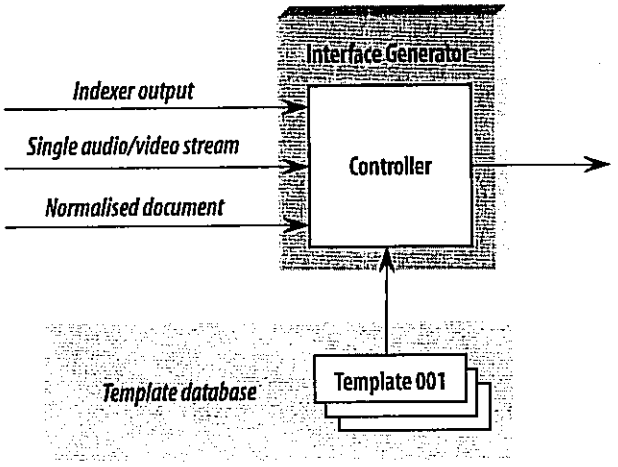


Figure 10: Interface generator design

Because of its reliance on the requirements of the interface, there is some coupling between the Normalisation module and the interface.

The initial design of this module will involve the normalisation of all documents into PNG format in 32-bit colour. PDF and PostScript files will require rasterization to achieve this, and documents submitted to the system as a directory of images may require resizing and conversion to the PNG format. The justification for this approach is to allow slides or scanned transparencies to be output in the same manner, reducing the need for the interface to handle OHP-based lectures any differently to digital slide-based lectures. While all documents could be normalised to PDF, such a design was too complex to implement in the time available.

4.8. Interface Generation

The Interface Generator module uses the output from the indexer, document normaliser, and compressors to assemble an output interface. The interface could feasibly take the form of an applet, standalone executable, or a webpage. As illustrated in Figure 10, the generator uses a database of templates, which may be selected by the framework based on the number of input streams and their characteristics, or manually by the user. For example, a template may exist supporting a configuration of one available video stream, one audio stream and slides in PNG format. Another template may exist for scenarios when slide images are unavailable, or when no video is available.

Templates specify the screen layout of the interface and how each stream will be presented, including the presentation of index points and use of extracted keywords. Different templates are required for each configuration due to differences in screen layout (when one or more streams are absent), and also differences in implementation when certain streams are present. The content of the stream may also influence template selection. For example, if the documents submitted to the system are transparencies (of A4 size), a different template may be used compared to a scenario when standard digital slides are used.

The purpose of the database is to provide flexibility in both visual design and support for as many configuration scenarios as possible. Further, an extensive database can allow the system to use different layouts for lectures captured from different classes, incorporating logos, colour schemes and links.

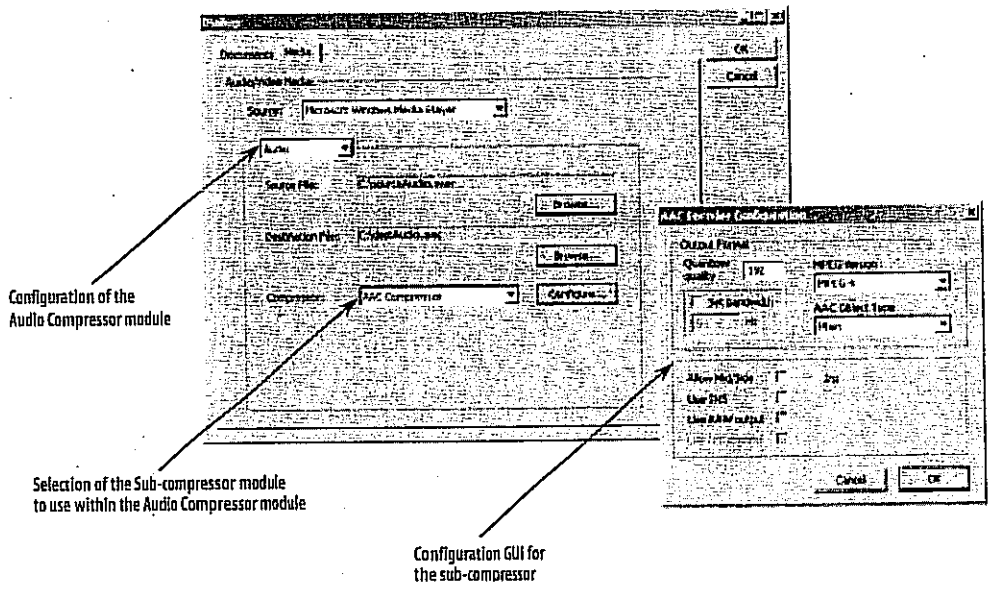


Figure 11: Processing Application Interface

The initial Interface Generator module will incorporate a design for producing HTML-based interfaces, consistent with the use of HTML in interfaces in other systems described in section 2. These interfaces allow web browsers on multiple platforms to use the output produced by the framework and permits the system to be used by the greatest number of students [3]. To display video, embedded media players such as Windows Media Player and Quicktime may be utilised within the webpages.

4.9. Implementation

The framework was implemented in C++ under Windows in order to take advantage of technologies such as DirectShow which provides a wide range of multimedia processing facilities. DirectShow is a subsystem of the DirectX SDK[20] produced by Microsoft, and simplifies the process of accessing and manipulating multimedia streams by the construction of *filter graphs*. DirectShow filters perform specific tasks such as compression, decompression, video rendering, and file parsing and writing[21]. Filters have inputs, processing, and output and when connected together, create a directed, acyclic graph. Custom filters may be written to effectively perform any manipulation task—such as adding of special effects to a sound stream, or in the case of the framework presented in this research, analysis of sound and video frames to facilitate indexing.

All modules are implemented as DLLs (dynamically-linked libraries) to implement the pluggable design. A processor application was built to manage the configuration and execution of the overall framework, including the execution of functions within each of the DLLs. For ease of use, each module has a configuration window (GUI) associated with it, allowing specific settings—such as the bitrate for the compressor, the location of source files, and so on—to be configured with ease. Part of the interface designed for the processing application is illustrated in Figure 11.

[20] http://www.microsoft.com/directx/

[21] http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/dx81_c/directx_cpp/htm/thefiltergraph.asp

Since the system has been written in an object-oriented language, interfaces have been used to allow the separation of behaviour specification and implementation. Interfaces have been created for all framework modules, such as a Video Compressor Interface, and a Muxer Interface, for example. The interfaces contain C++ method specifications such as `execute()` and `setSourceFilename()`. New framework modules may be created simply by extending one of the interfaces creating a concrete Class and implementing the methods to carry out a specialised task. Placement of the new DLL in the `modules/` directory will allow the system to use it.

Some modules in the framework contain sub-components—specifically, the Muxer, Video, and Audio Compressor modules. The processing application allows the user to select which sub module to invoke in the framework, effectively allowing the selection of compressor and muxer to be used from the available database. Modules such as the Document Indexer (see section 4.7.2) are capable of automatically determining which sub module to invoke depending on the extension of the input file(s).

To simplify implementation, parts of the framework execute in stages. The process of indexing runs to completion before the compression/muxing stage, which then leads to the interface generation stage. Executing the framework as a 'pipeline' is not feasible, as some of the streams (specifically, documents) are not temporally based and cannot be processed in parallel with a video or audio stream. The consequence of this is the separation of the indexing into two groups—temporal stream indexing for audio and video, and non-temporal stream indexing for documents.

4.9.1. Video and Audio Processing

Synchronisation between the audio and video streams at the preprocessing stage (as described in sections 4.5.2, and 4.6.2) was carried out manually, however skeleton components were developed to support this feature, minus the source code for detecting synchronisation signals.

Section 4.5.4 described a design initially incorporating two sub-compressors for the Video Compressor module. The DivX and XviD compressors are developed and distributed as DirectShow filters[22], allowing video compression to be handled by the DirectShow filter graph architecture. The development of the MP4 sub-muxer module described in section 4.5.5 required the modification of the MPEG4IP source code to allow the system to function within a DLL, and development of a GUI for the commandline muxer was required to allow the module to have a configuration window consistent with those of other modules in the framework.

The AAC and MP3 audio design specifications (see section 4.6.4) were implemented using the FAAC encoder[23] and the LAME MP3 encoder[24], respectively. Modification of source of these utilities was necessary to allow them to function within the DLL specifications of the framework.

---

[22] The DivX compressor (v5.0.6) is actually a Video for Windows (VfW) component, but a DirectShow 'wrapper' filter is provided to allow its use in filter graphs.
[23] http://faac.sourceforge.net/index.php
[24] http://lame.sourceforge.net/

4.9.2. Document Processing

As discussed in the design of the Document Indexer in section 4.7.2, the analysis of PostScript and PDF files has been included in the framework. The implementation of this component is based on the use of xpdf—an open source viewer for Portable Document Format (PDF) files that includes utilities to convert PDF files to text[25]. The C++ source code for one of these utilities was re-written to function as a Document Indexer sub-module within the framework, and will parse documents of PDF and PostScript format to extract ASCII text.

The Normalisation Module has been implemented using GhostScript[26], which manages the task of PDF and PostScript rasterization to PNG images, and also the CxImage library[27], which manages image resizing and image format conversion to PNG if required.

4.9.3. Interface Generation

While preliminary experiments were successful in producing a basic HTML interface that was cross-compatible with Windows and MacOS, it was felt that current limitations in compatibility between browser implementations on MacOS, Linux and Windows would require the development of effectively three interfaces specific to these operating systems. The technical complexity of the access interface necessary to implement all of the guidelines outlined section 5.2 would require embedded media player support and synchronisation facilities to be present in browsers on all three operating systems in order for one template to suffice. At present, a complete set of standardised, cross-platform facilities does not exist in browsers on all three platforms. For example, embedded media player support varies between operating systems, and synchronization frameworks such as SMIL are implemented in only some of the major browsers.

Hence development of a single, fully functional (but browser-specific) interface was chosen as the goal in order to make best use of available time, and to allow for the evaluation of interface guidelines, further discussed in section 5. It is expected that in the future it may be possible to use a single template to create an HTML interface that is 100% compatible with Linux, MacOS and Windows web browsers.

[25] http://www.foolabs.com/xpdf/
[26] http://www.ghostscript.com/
[27] http://www.codeproject.com/bitmap/cximage.asp

5. Interface Design and Evaluation

5.1. Introduction

Motivated by the research evaluation presented by Abowd [2], this section proposes seven general guidelines that are suggested to be incorporated into any Digital Lecture access interface. The access interface is the part of the system used by students to access the captured material, and its design is important in ensuring it supports student tasks. A detailed analysis of the design, implementation and subsequent evaluation of one access interface template is presented in this section. As discussed in section 4.9.3, the interface presented has been designed to represent a fully featured interface to allow for a complete evaluation, and a demonstration of the use of all three streams in the framework. The guidelines are presented in section 5.2, and a discussion of their incorporation into the design of the access interface in section 5.3. An evaluation focused on examining how students use the interface is presented in section 5.4, including a discussion on the validation of the proposed guidelines.

5.2. Guidelines

Seven general guidelines are suggested in this section for incorporation into any interface created by the Interface Generator module of the framework. These have been developed by examination and exploration of the interfaces of some of the Digital Lecture systems presented in section 2. The aspects that were felt to be the most innovative and useful were identified for each, and elements have been incorporated into the guidelines presented. Validation of these guidelines is one of the goals of the research.

5.2.1. Flexible delivery medium

The interface should be sufficiently flexible to allow students to view it on any platform such as MacOS, Linux, Windows, and potentially mobile devices such as PDAs or cellphones. Flexibility in delivery allows the greatest distribution coverage.

5.2.2. Support for audio, video, document presentation

These streams are seen to cover the basic three elements of lecture artefacts—what is seen, what is said and what is handed out or projected. Documents may be classified as slides, scanned transparencies, handouts or even a book. The interface should allow support for these three streams in order to provide the best reproduction of the captured event.

5.2.3. Synchronisation between all time-based streams

Synchronisation is important to keep the interface consistent, and to allow users flexibility to use any part of the interface to navigate. For example, changing the position of the video should result in appropriate changes in other parts of the interface, such as the document displayed according to the new video position.

5.2.4. Support seeking

Seeking—the behaviour of jumping to specific points within the recorded lecture—should be supported in the interface to allow users familiar with the captured material to have rapid access to a point of interest.

Further, the interface should allow users to use the navigation method most comfortable to them. Some users may prefer to use documents (such as slides) to navigate the captured material. Alternatively if users are aware that a particular piece of information exists near the end or start of the captured footage (for example), navigation by time should also be supported.

5.2.5. Support elimination of any or all streams

Incorporation of this guideline is recommended based on the premise that users should be in control of the interface, and should be able to eliminate streams that are not of interest. For example, elimination of video, audio, and document streams should be supported. Interface designers should not impose streams on users, and this is one of the noted limitations of the systems presented in section 2.

5.2.6. Support student annotations

Annotations are an important aspect of lectures for many students, and existing systems often do not support this feature [2]. While not all students may prefer an electronic version of note-taking, the facility should be available for students comfortable with the concept. Where possible, the note taking process should be as close as possible to the paper-based approach, where notes may be written anywhere on the slide.

5.2.7. Search support

Effectively seeking by keyword, searches are important to allow students to find information rapidly and easily. Searches can be based on documents, audio or video streams, and any others indexed by the system. Search facilities are important when dozens of index points have been identified or when a visual search is not sufficient to find the desired information.

5.3. Interface Design

An interface has been designed and implemented, incorporating the guidelines discussed in section 5.2. Aspects of its design are presented and justified in this section, with a specific focus on how each of the guidelines has been incorporated.

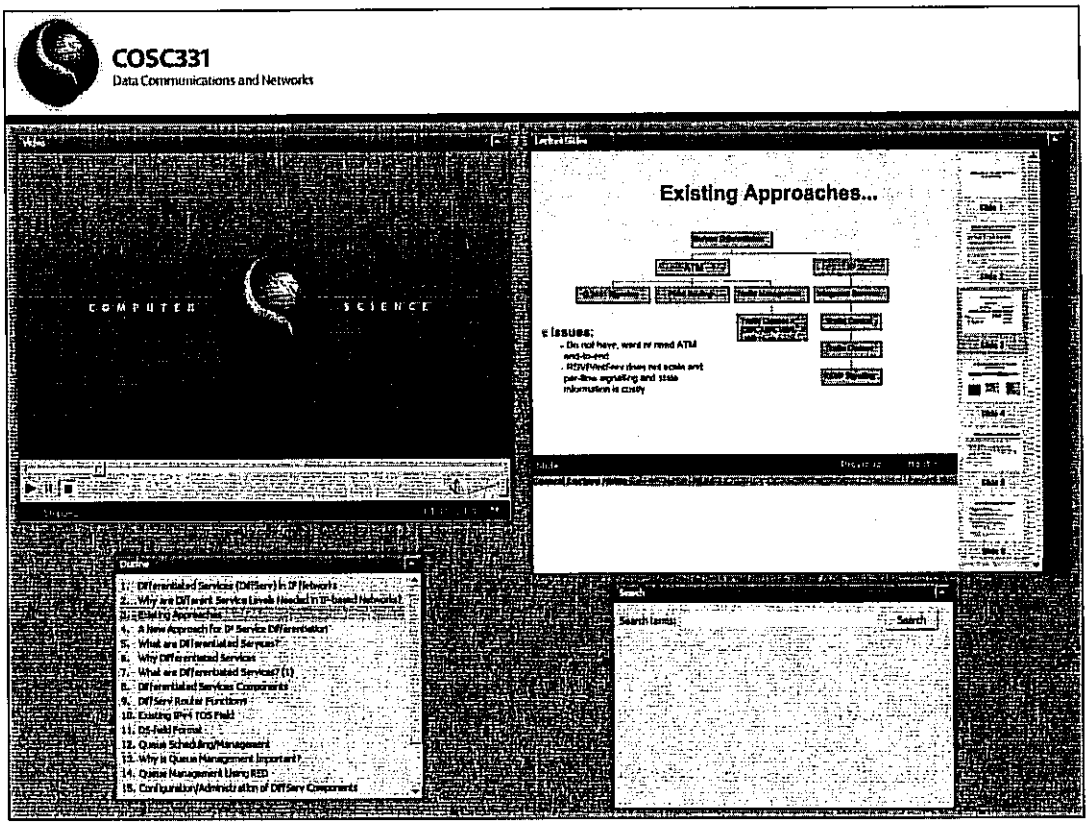


Figure 12: HTML-based access interface showing a captured COSC331 lecture

5.3.1. General Design

Illustrated in Figure 12, the interface has been designed to incorporate four major windows:

1. Video
2. Lecture Slides
3. Outline
4. Search

The specific details of each window is discussed in sections 5.3.2 to 5.3.5, and the presence of the Video and Slides window satisfies the guideline stipulating that the interface support audio, video and document presentation[28]. Each window has a title bar and has been implemented as a CSS (Cascading Style Sheet) layer that allows movability. Clicking and dragging the title bar allows repositioning of windows to anywhere on the screen, but since the windows are CSS layers, desktop clutter is avoided as the layers cannot leave the browser window. The design also incorporates the use of tooltips, and a visually appealing design that attempts to make optimal use of screen space. The implementation of the functionality is based on JavaScript, allowing all functionality of the interface to be contained within a single HTML file.

[28] Rendering of audio is managed as part of the video window.

The HTML-based design satisfies the guideline that the delivery medium be flexible (see section 5.2.1). Web browsers are available on many platforms, PDAs and cellphones. While for reasons of technical complexity and time (see section 4.9.3) the current implementation is browser-specific, the motivation for using HTML has been due to the flexibility offered. Further, this delivery medium allows the interface to be made available from a webserver, or saved as a file on CD for local access by students.

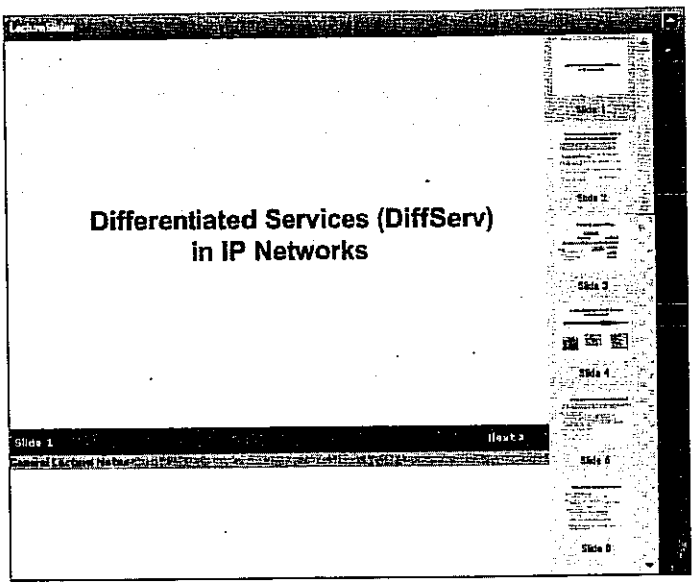


Figure 13: Lecture Slides Window

5.3.2. Video Window

The video window utilises an embedded media player (Windows Media Player 6.4 in this implementation) to display the video stream of the lecture, and renders the audio stream. The typical controls of *play*, *stop*, and *pause* are provided, and a volume control is available on the bottom right of the window. A slider control is available to allow seeking to any part of the lecture. Fullscreen video playback is also supported by the embedded media player.

Preliminary tests with the interface revealed a delay of approximately one second (on a Pentium 4) for the video to seek to the desired position if the slider was moved. To improve the response time, the framework was adjusted to produce a video stream with keyframes spaced at 12-frame intervals instead of 30. Keyframes the basis from which seek operations function, and the interval reduction resulted in the seek time decreasing to less than 0.5 seconds.

5.3.3. Lecture Slides Window

The most complex of the onscreen windows, this window is designed to display PNG images of slides or scanned transparencies. Presented in Figure 13, it represents the output of the document processing components of the framework.

As mentioned, this template has been specifically designed for display of slide or transparency-based presentations, and a scrollable list containing thumbnails of each of the slides presented in the lecture is illustrated in the *sidebar* in Figure 13. Slide synchronisation with the audio and video streams is maintained (see section 5.3.8)—even when the user repositions the media player slider widget. The sidebar highlights the current slide in red (currently *slide 1* in the figure), which dynamically changes as the lecture progresses. Immediate seeking to slides is facilitated if the user clicks one of the slides in the list, which brings that slide into full view in the window, and adjusts the audio and video position to the time index when that slide was presented. Basic mouse-over events have been incorporated into the sidebar to provide immediate feedback, and the thumbnail list is collapsible (out of view) and expandable

(into view) by clicking the ■ icon at the top right of the window. This feature was incorporated to allow user flexibility in controlling the information presented onscreen. For example, students wishing to view a lecture from start to finish are unlikely to be interested in such a sidebar and will most likely wish it to be collapsed.

Provisions for student notes have also been incorporated into this window. See section 5.3.6 for additional design details.

5.3.4. Outline

The Outline window presents a list of index points identified for the lecture. For the COSC331 lecture presented in Figure 12, the outline contains a list of index points related to slide change events. Clicking any of these will seek the Video and Slides window to that index point. While the window currently contains only slide change events, additional events—such as the activation of an overhead projector or a student question—could also be included as index items in the list presented in this window.

While this window also appears to contain similar information to the sidebar with slide thumbnails, the two are differentiated. The Outline window can contain other event entries (as discussed above), and was also designed to allow students to seek to a slide change event by *title*, while the side bar allows seeking based on *visual features* (such as the colour or layout of a slide). Observations concerning this differentiation are discussed in the evaluation in section 5.4.

5.3.5. Search

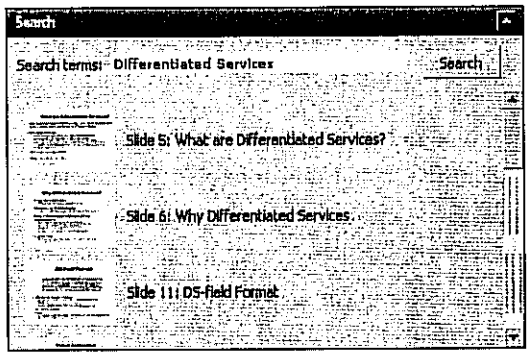


Figure 14: Search Window

The Search window is the visual representation of search facilities incorporated into the interface, based on the guideline introduced in section 5.2.7. Illustrated in Figure 14, the mechanisms behind the window incorporate the Indexer module's keyword list. Users are permitted to enter keywords into the window, and a search is executed on the list with ranked results returned according to documents that match one or all keywords. In Figure 14, the search is carried out on keywords extracted by the Document Indexer from the slides submitted to the processing framework. The returned results list matching slides, and each result entry may be clicked to seek the interface to the presentation of that slide.

5.3.6. Student Annotations

Incorporated into the Lecture Slides window described in section 5.3.3 are two facilities enabling students to add notes to slides. The motivation for the incorporation of these features lies in the guideline established in section 5.2.6 suggesting inclusion for support of student annotations.

The lower half of the window in Figure 13 reveals a space for students to add non slide-specific notes—similar to writing notes on a separate piece of paper the notes are persistent even when the slide changes.

Further, the interface supports recording of slide specific notes, as illustrated in Figure 15. Clicking on the enlarged slide area prompts the interface to display a floating textbox (or popup note) where a text may

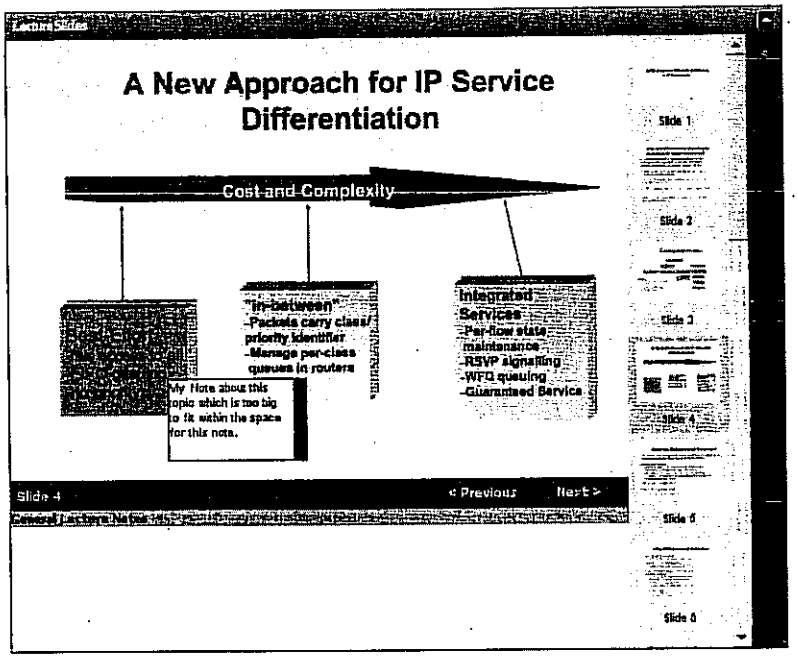


Figure 15: Entry of notes in the Lecture Slides Window

be entered which is attached to the slide, similar to a *post-it* note. The note box expands as the message is typed, and may be moved around the slide by dragging the red bar on the right of the note. Once the entry of the text is complete, the note becomes partially transparent in order to allow content of the slide underneath to be partially visible. Long notes are also collapsed to two lines only followed by an ellipsis indicating hidden text (see Figure 16).

The notes are implemented using CSS layers (similar to the windows themselves), and this implementation decision allows notes to reside anywhere on the screen, while remaining 'attached' to a slide. Hence, the pixel area onscreen for the slide is not a limitation on the number of notes a student may make, as each may be dragged and positioned outside the Lecture Slide window if the user desires.

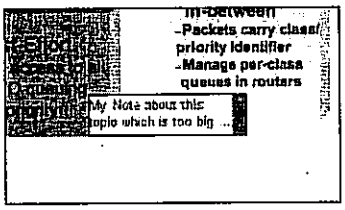


Figure 16: Collapsed slide note

5.3.7. Collapsible Windows

Elimination of streams is an important guideline introduced in section 5.2.5, and has been incorporated into the interface design through the *minimise* or *collapse* feature on all windows. This is illustrated in Figure 12, where all four windows have a collapse icon in their top right corner. Clicking this icon collapses the window to the title bar only[29], similar to behaviour on the MacOS 8 and 9 windowing system. Effectively, this allows the elimination of the slides, outline and even video if the user wishes. Windows are restored by clicking the icon again. Combined with the movability of the windows, the minimise function not only allows streams to be collapsed, but moved out of view to the top of bottom of the screen. The elimination of some streams may improve performance, especially if the captured material is streamed from a remote server. For example, collapsing the slides window will eliminate the

[29] The same effect may also be achieved through a double-click on the title bar.

need for the interface to load and display images, and eliminating video stream will reduce system load as video decompression will be unnecessary.

5.3.8. Synchronisation

The SMIL (Synchronized Multimedia Integration Language) specification[30] standardised by the W3C[31] allows authoring of synchronised multimedia presentations and is "typically used for 'rich media'/multimedia presentations which integrate streaming audio and video with images, text or any other media type." Investigated due to the synchronisation guideline stipulated in section 5.2.3, the SMIL standard has been implemented in Microsoft Internet Explorer 5.5 and 6.0, and allows independent objects on a webpage to function based on a common timeline. Changes in the timeline (seek, start or stop operations) cause all objects on the timeline to respond accordingly. Based on successful preliminary prototypes, the use of SMIL was incorporated into the interface allowing changes in one window to immediately cause synchronisation across all windows onscreen. For example, selecting a thumbnail in the sidebar immediately causes the video to seek to the correct position, and the outline window to outline the appropriate index item—all of which is managed internally by the web browser. Likewise, the action of the user moving the video slider widget causes similar updates in the Lecture Slides and Outline window. Since output from the framework Indexer is a list of event times and descriptions, conversion of these times into SMIL syntax is straightforward.

In addition to Internet Explorer, the SMIL standard has also been implemented in some PDAs and also cellular phones. Thus the use of this technology was justified not only because it provided a prebuilt framework for synchronisation of streams, but also because of the future flexibility offered, demonstrated by the fact the standard has been implemented on small devices as well as desktop applications. While not confirmed, the implementation of this standard in browsers such as Mozilla is expected in the near future which would allow the SMIL-based interface templates to be platform independent.

5.4. Evaluation

This section presents the informal evaluation of the interface introduced in section 5.3 with the guidelines proposed in section 5.2. Motivations and objectives of the evaluation are presented, along with its design, execution and results.

5.4.1. Motivations

The developed interface presented in previous sections is very much of a prototype nature, and is the result of the initial phase of an iterative development process. Informal evaluations are one method of establishing usability of the design of any interface [25], and have been a motivating factor behind the evaluation presented.

The activity of users with the interface drives the development of the framework, and this is a motivating factor in establishing guidelines and evaluating the interface against them. As discussed in section 5.3.2, changes to the framework were necessary in order to correct a usability issue with the interface. Further

[30] http://www.w3.org/AudioVideo/
[31] http://www.w3.org/

examples of interface issues influencing the framework may include students demanding the availability of whiteboard notes in the interface, written transcripts of lectures, or for the system to visibly indicate sections of the lecture when exam hints are mentioned. Each of these demands on the interface inherently drives changes in the framework to accommodate them. Because of this, evaluation of the interface is an indirect evaluation of the functions being performed by the framework and the degree to which these support the needs students might have for the system.

Further, the experiment is motivated by a need to validate the guidelines established in 5.2. If students find the interface difficult to use, it may be due to the existence of a guideline that is hindering student use of the interface, rather than helping it.

5.4.2. Objectives

The experiment has been developed based on two objectives spawned by motivations outlined in section 5.4.1. The experiment aims to:

- Construct a list of observations based on how users interact with the system based on student comments, behaviour and questionnaire responses. The observations can be used in further iterations of the interface development.
- Validate the guidelines for the access interface.

5.4.3. Design

The evaluation was designed to use a captured lecture from one course, and to use students from the three groups:

1. Students who were present at the recorded lecture.
2. Students who were not present at the recorded lecture, but are enrolled in the course.
3. Students who were not present, and are not enrolled in the course.

It is expected that students from each of the groups would use the interface differently depending on their motivations for watching the captured lecture, knowledge of the topics covered, and whether the students have handouts of the lecture or are relying on the system provide screen versions. Due to time constraints, an exhaustive evaluation of precisely how each of the three groups of students use the system was not possible. However, the inherent differences between groups are acknowledged, and aspects of this have been taken into consideration in the design of the evaluation.

Since the evaluations are of an informal nature for the purposes of further design refinement, a list of predefined tasks was compiled for participants to work through with the interface, specifically testing interface aspects such as:

- The methods students use to navigate the captured material, specifically the method used to navigate to a particular slide pointed out from a handout, and a slide of a particular number (for example, slide 12).
- The methods students use to find whether a certain topic was covered in a lecture.

- Whether students use screen or paper-based annotations when asked to record some important fact.
- Whether students were aware that streams could be eliminated if not useful.

The tasks involved using the interface to find a particular slide based on one in a handout, one by topic and one by number. Tasks related to the determination of whether the lecturer spoke about certain topics were included, as well as tasks to determine whether students used the movable and collapsible features of the system and annotation facilities. In total, participants were asked to carry out ten individual tasks.

During the evaluation, observations were noted specifically regarding the interface facilities used to carry out tasks, and anything the student said or was unsure of. A number of oral questions were asked at the end of the evaluation regarding

- The student's preferred method of note taking (screen, paper, or neither) and why.
- Any general comments or suggestions the student had about the interface.

A questionnaire was also given to students covering aspects such as how often they might expect to use such a system[32], subjective questions related to the size and quality of the audio and video streams, and whether they felt the Search and Lecture Slide windows were useful. The questionnaire is included in Appendix A.

5.4.4. Execution

One lecture from a 2003 semester two course COSC331 was recorded using one tripod-mounted DV camera and sound captured with a minidisc recorder connected to a high quality stereo microphone. PowerPoint slides from the lecture were converted to PDF and submitted to the framework for processing.

A number of students from this course were selected from volunteers. These students were categorised into groups one and two[33] as appropriate depending on their presence during the captured lecture used in the experiment. Ten students from this course participated in the experiment—nine in group one, and one student in group two. Eight students not enrolled in COSC331 were used as group three participants, giving a total of 18 students for the overall evaluation.

Participants were initially given as much time as needed to become familiar with the interface—typically 5–10 minutes. A printed copy of the handout given at the lecture was given to all students (including those from group two and three)[34], along with a pen for the student to write on the handout at any stage.

The evaluation was carried out using a computer equipped with headphones, and tasks were explained to students orally. Participants were given assistance when necessary and any difficulties noted. Unexpected responses, such as the use of parts of the interface that were not anticipated, were noted and participants

---

[32] This question has relevance to an interface evaluation, and repeat users of the system are likely to require a well-designed indexing system in order to allow rapid access to target aspects of the lecture. Further, the interface should not hinder experienced users.

[33] Volunteers were selected at the lecture after the one recorded.

[34] Lecture notes for the captured course (COSC331) are readily available online, and it is not unrealistic for students not present in the lecture to have a copy of this handout material.

asked to explain how they thought components of the interface functioned—specifically the search. This information was recorded to aid in an understanding of how participants interpreted the model of the interface.

5.4.5. Results

As discussed, the results of the experiment are a list of observations, comments, and some questionnaire-based statistics.

When asked to locate a slide in the interface given one identified in a corresponding handout for the lecture, 50% of participants used the Outline window to locate the slide based on its title. The other half of participants used the Sidebar thumbnails of slides to find a match by visual comparison. A number of students used both methods to locate the target slide. Contrastingly, when asked to locate a slide based on *number*, 85% participants used the Outline window, as it contains a numbered list of slide change events.

When asked to seek to any slide on "Random Early Detection" (a topic covered in the captured lecture), all participants used the search facility. A number of students enrolled in COSC331 recognised the acronym for this topic is 'RED' and instinctively entered this into the search. Further, when asked to determine whether the IPv6 topic had been covered at any stage during the lecture, 85% of participants again utilised the search facility, with 15% being able to answer this question immediately based on recall that the topic was covered.

A task was incorporated into the experiment to test how students would use the interface if the search failed to reveal any results. Specifically, when asked whether the lecturer mentioned labs or the exam in the lecture, 82% of participants attempted to use the search which did not reveal any results. Other participants watched the beginning and end portions of the lecture, based on experience that these are the periods when this lecturer is most likely to mention such topics. While the topics of labs were mentioned in the lecture, none of the participants isolated the segment that contained the reference eight minutes into the lecture.

Approximately 83% of participants preferred to use paper notes when asked to make a note to themselves about a comment made by the lecturer that was not covered in handouts or slides. When asked to explain their decision to use paper notes, these participants acknowledged that they were unaware the system supported electronic annotations. The remainder of participants (17%) opted to use the General Notes text field to enter the note, and none of the participants used (or were initially aware of) the interface's capability to record slide-specific notes. Once pointed out, 50% of participants said the 'pop-up' notes were a feature they might likely use, with 33% preferring paper-based annotations, and 17% using the non-slide specific General Notes feature of the system. Participant views regarding the annotation facilities fell into three general groups:

- those with a strong preference for paper-based notes, and would not use the annotation facilities on the interface (33%).
- those who prefer paper-based notes, but may use annotation facilities in the future (47%).
- those with a strong preference for the annotation facilities of the interface (20%).

The preference for the annotation facilities of the interface was justified by some participants due to a preference to type notes rather than write them.

All participants recognised the movability of windows, and approximately 90% discovered the use of the collapse/minimise buttons to eliminate streams. Further, when asked to identify the most important windows on screen, the majority of participants selected the Video and Slide windows. However, one participant noted that audio alone would often be sufficient, and several stated that the Search window was more important than the other three. Synchronisation between windows was a feature all participants agreed was important, with several noting that the interface would become confusing if the slides did not match the video.

Results from student responses to the questionnaire reveal that all participants would use the system if available, with 40% preferring to view the captured lectures at home only, and 60% preferring to have the system available both at home and on campus. Participants generally acknowledged the video quality was above average, and of an appropriate size on screen. Approximately 70% of students found the audio to be very useful in performing the tasks in the evaluation with many remarking on the quality of the stereo track. As confirmed by evaluation observations, the popularity of the search window was confirmed by the participants with 80% rating the search facility to be extremely useful.

5.4.6. Participant Comments

Comments regarding the system were recorded, with participants suggesting a range of improvements and features that could be incorporated into the interface. These are briefly outlined below:

- More obvious indication of slide changes. When typing, a slide change may go unnoticed.
- Use of the delete key to remove popup notes.
- Entry of time (in minutes) as another form of seeking
- A facility to allow thumbnails (from the side bar) to be dragged out to any position on the screen, like a floating window. This would allow users to create a collection of important slides which could be clicked to seek the system to the point of their delivery.
- Inclusion of whiteboard notes in another window, sourced from a second video camera.
- The separation of the notes and slides window. Currently the General Notes (not slide-specific) are in the same window as the slides.
- Inclusion of links to the course homepage.
- A facility to allow users to create their own outline including only events they deem significant. Further, indexing of annotations could be incorporated, allowing a user to seek the system to the point when the note was made (specifically for notes made in the General Notes area).
- Indication in the sidebar of thumbnail slides that contain annotations. Currently the system does not indicate which slides are augmented with popup notes unless the slide is in view.
- Resizability of windows. The system does not currently support this.

5.4.7. Discussion

The results from this experiment raise a number of important points. It was noted during the experiment that students associated numbering of entries in the Outline list with actual slide numbers. Specifically, when asked to seek to slide 14, many participants associated this with entry 14 in the Outline. While in this experiment such an assumption holds (the outline contains only slide change events), in the future this assumption may not be valid, as other events could also be listed in this window, such as changes in speaker, use of the whiteboard, and other events as outlined in section 4.4.

The use of acronyms in the search was not anticipated, and illustrates how users enrolled in the course use the system differently to those who are not. This observation affects the framework, as indexing techniques must be designed not to exclude acronyms. Specifically, voice recognition software may ignore acronyms that don't evaluate to valid English words.

One of the more significant observations from this experiment is the reliance of users on the indexing facilities in the interface—specifically the Search and Outline windows. The task involving the search for references to the exam or lab was designed to test the participants' reliance on these interface features, and the results have shown that when the search returns no results, participants often assumed the topic was not covered in the lecture. However, a small number of participants were aware of the methods used to construct the index used in the interface Search window, and were aware of its limitations. These users acknowledged that a failed search did not necessarily indicate the topic was not covered in the lecture—it meant only that the input terms did not match any of the keywords extracted by framework indexers. This observation indicates that many students rely on the framework to extract index points and keywords of sufficient granularity to allow a topic-based search based on non-document-based streams (such as video or audio). To facilitate this, there is a need in the framework for voice-based indexers potentially capable of producing readable transcripts of lectures.

Within minutes of being introduced to the system, it was observed that nearly all participants deduced that windows in the interface could be moved, and participants were observed to use this feature to shift windows in and out of view as required. However, while 90% of participants discovered the use of the collapse/minimise buttons on the interface windows, this was only after being asked how they might go about eliminating a window if it was of no use. Given the short duration of the experiment, it was not expected that many participants would eliminate windows unless prompted, as this was intended to be a feature used only during long sessions with the system. Follow-up questions did indicate the majority of participants would find such a feature useful in some circumstances. Specifically, if a printed set of handouts were available, some participants indicated the onscreen version of the slides would not be of use—even less so if the printed notes already contained annotations.

Comments from participants as listed in section 5.4.6 included some ideas and suggestions for tweaking the interface to improve usability such as the incorporation of resize features on windows, and the separation of the General Notes area from the Slides window. However, a number of important suggestions for new features were proposed which deserve further explanation.

The ability to drag individual thumbnails from the sidebar—effectively converting them into a floating bookmark—could allow students to select those slides that they deem important. Combined with a 'save' feature, this could be used by students to return to captured lectures at a later date and immediately view the slides deemed important from the last session. Further, a suggested feature is a customisable Outline.

The system currently presents all indexed points as a numbered list in the Outline, and the idea proposed by an evaluation participant involved allowing students to create their own index events during the lecture. Points when examinable concepts are covered, or portions which are most likely to require revision in the future—each of which is specific to each student—could be created using such a facility. Similar in concept to the floating thumbnails suggestion, a customisable outline provides a much finer level of granularity over the slide-based granularity of thumbnails. A potential problem with a Digital Lecture system such is this is an *overload* of information, and facilities such as these can allow students to filter and organise it according to their own style.

5.4.8. Guidelines validation

One of the objectives of the evaluation outlined in section 5.4.2 is the validation of the proposed guidelines. The guideline stipulating that the interface be delivered on a flexible medium has been validated by all participants indicating a preference to have the system available at home. In order to facilitate this, a flexible medium is necessary to adapt to delivery on home PC setups, possibly through the use of internet streaming, or CD distribution.

Inclusion for the guideline suggesting that interfaces support audio, video and document presentation has been justified by observations and participant responses to questions regarding which interface aspects they found to be the most useful. As discussed, overall participants expressed an opinion that all three were useful, though individual preferences varied. To support the widest range of students, the inclusion of all three is justified.

The need for synchronisation across streams (windows) has been confirmed by all participants indicating that consistency between information displayed between windows is important. Further, interface seeking facilities were used extensively by all participants in carrying out the tasks in this evaluation, with some indicating the Outline window to be one of the most important on screen. While suggestions have been made to further improve the implementation of seeking in the interface used in this experiment, its inclusion as a general guideline has been justified by the evaluation results.

As discussed in section 5.4.7, the evaluation period was too short to realistically examine how participants used the stream elimination features of the interface. However, some participants did indicate that elimination of the slides window/stream would be useful in some circumstances. Although the results presented indicate the potential use of such elimination facilities, further experiments are necessary to effectively establish the validity of the stream elimination guideline.

Annotation facilities have been established as a feature requested by students of other Digital Lecture systems [2], and the results of this evaluation coincide with these findings. Based on this, its inclusion as a guideline has been justified though not all participants indicate it is a feature they would immediately use. Based on participant suggestions and observations, it is recommended that the guideline specified in section 5.2.6 be amended to also include specifications of save and print facilities for annotations.

Inclusion of search support has been justified as a guideline by the results of this evaluation. As presented in section 5.4.5, the search facility was used by all participants to carry out the tasks and was identified by some as the most important window in the interface.

5.4.9. Conclusion

The evaluation presented has validated six of the seven proposed guidelines, resulted in a number of positive suggestions from participants for improvement, and has permitted the observation of a number of important interactions users have with the system including reliance on the indexing facilities of the interface. A number of the identified interactions have impact on the framework as a whole, specifically the type and granularity of indexing carried out.

Several design issues in the interface require addressing, including more a obvious indication of the popup note facility, and incorporating resizing features to windows. It is recommended that the results and student comments from this evaluation be taken into account during future development iterations of the interface, and further experiments are recommended to establish the validity of the guideline stipulating the inclusion of stream elimination in the interface.

Two significant suggestions from participants are also an important result of this evaluation. The ability to drag thumbnails to identify important slides and the use of the system to create a personal outline are uses of the system which differ significantly from those initially considered at the beginning of this evaluation. The implications of these suggestions are substantial, and will have impact on the behaviour of the framework as a whole. Details are discussed further in section 7.

6. Discussion

This section presents a general discussion of the research presented in this report. Applications of the framework outside a lecture environment is presented in section 6.1, and the process of incorporating a new stream is discussed in section 6.2. Limitations and recommended future work are outlined in sections 6.3 and 6.4, respectively.

6.1. Framework Generalisation

The framework design is sufficiently flexible to allow its use in the recording of other 'non-lecture' events, such as training or demonstration events. For example, the system could be used to record a demonstration of a piece of music to tutor a music student. Input video to the system could be footage of the pianist from multiple angles (using an extension to the system suggested in section 6.4.2), and the document input could be the notes of the music being played. Indexed points could be different pieces of music demonstrated, with index points even being created by the system to the granularity of individual notes. While the Indexer components of the framework may need alteration, the overall architecture would support the capture of such an event.

6.2. New Stream Integration

The integration of new streams has been designed to be as streamlined as possible in the framework. The structural changes resulting from the addition of a new stream to the framework is illustrated in Figure 17. Using an example of an electronic whiteboard stream, some preprocessing may be required to convert the output from a list of pen strokes to a format usable in the indexer. Once preprocessed the data would be passed to the indexer which would be equipped with a new sub-indexer specifically designed to parse the output from the pre-processor. The indexer may group events such as pen-up/down motions to extract events. Once processed the output may be forwarded to the Interface Generator which may include images from the whiteboard in a new window. Alternatively, the generator may choose to ignore this output, and simply include index points stating when the whiteboard was used, and images from existing video could suffice in illustrating what was written.

In this scenario, one new module was required, and one new indexer sub module. Alterations to interface templates *may* be required, although situations such as that illustrated above may not require it. Because the framework modules are based on Object Oriented Class interfaces, implementation of a new indexer sub-module simply requires the extension and concrete implementation of a generic sub-indexer

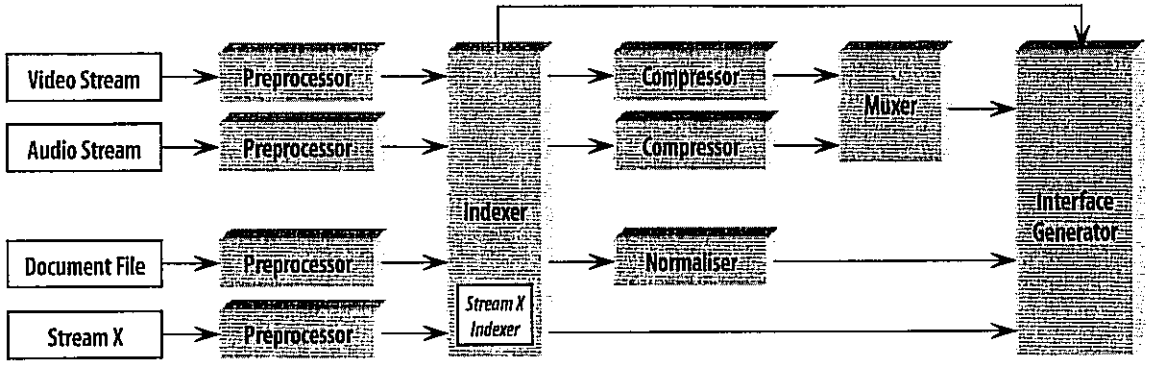


Figure 17: Integration of a new stream

interface. Excluding those already mentioned, it is envisioned that no other components of the framework would require alteration.

6.3. Limitations

The system is currently limited to processing the input of one video and one audio stream. This limitation prevents the capture of lectures using multiple cameras, each focused on recording one lecture feature—such as the lecturer, the whiteboard, or even the audience. Further, multiple audio input devices are not supported by the current design and would be useful in theatres with multiple embedded microphones. Section 6.4 identifies some solutions to these limitations.

The process of indexing is currently a manual task, and its automation is one of the important features remaining to be implemented in the framework. The manual task currently involves several hours of work per lecture, and this is not realistic in the long term. Selected techniques for the video, audio and document streams have been included in this report, establishing that the challenges involved in implementing such a module are not insurmountable.

The browser-specific nature of the developed interface is an acknowledged limitation of the system at present. Justification for its Internet Explorer-specific design have been outlined in other parts of this report. It should be noted that this is purely an implementation limitation, and is not an inherent problem in the framework.

The evaluation carried out and reported in section 5 was informal in nature and was designed to primarily gather feedback on the initial design of the interface and proposed guidelines. This approach is limited by its informality, and as such only general conclusions may be drawn. This limitation may be addressed by the design of a formal experiment involving an extensive set of tasks, timed execution and potentially evaluation with respect to the interfaces of other systems.

6.4. Future Work

A number of directions of research have been identified for the framework proposed in this research. Specifically, the ideas presented are related to the incorporation of additional streams to the framework, the establishment of a Digital Lecture library, and a number of recommended enhancements to the interface prototype.

6.4.1. Slide change detection software

Detection of slide change events by visual analysis of video frames could be alleviated by the use of software installed on the machine controlling the slide presentation. Such software could capture keystrokes or mouse clicks which trigger changes in slides and record the times when these events occur. The output from the software would be an additional stream which could be used in the framework. However it should be noted that the introduction of such software may prove intrusive to some lecturers, and the process of downloading the captured data may also prove inconvenient. For willing lecturers, however, the use of such software could result in slide change event generation with effectively 100% accuracy.

6.4.2. Multiple video sources

Multiple cameras could be used to film a single lecture and all sources could be input into the processing architecture for editing. Integration of such a feature would require the design and implementation of a virtual director to determine which video streams are of interest at any given point. The output from the virtual director module would be a single video stream which would be processed in the same manner as is presently possible. Implementation of this design would require replacing only one module of the framework (specifically, the Video Stream input component) with one designed to produce one stream based on multiple video source streams. Research regarding the implementation of such a system has been carried out by Microsoft [15].

6.4.3. Pointer movement capture

Capture of mouse movements or laser pointer activity may greatly assist students viewing seminars from lecturers using gesture movements with pointers to identify key points on their slides. Pointer (mouse or laser) could easily be captured by a video camera fixed on a projection screen. Use of software to capture mouse pointer movements is possible, but with the consequence of intrusiveness to a lecturer (as discussed in section 6.4.1).

6.4.4. Construction of a Digital Lecture Library

All processed lectures should be placed in an online digital library with search facilities to allow students to search by keyword across multiple lectures, much the same as is currently possible within a single lecture. The data to support such a search is available in the system in the form of index points and keyword lists. Centralisation of these lists to a server could allow sophisticated searches to be carried out across all lectures presented for a course, and potentially across multiple courses.

6.4.5. Intelligent Interface Search

Current search algorithm implemented in the interface prototype is not sophisticated, and ranks slides according to the number of keywords matched, and the frequency within that slide. The search could be extended to give weighting to matches within the slide header text over matches within the body of a slide. Further support for boolean search operators would allow a complex search and could be incorporated into the algorithm.

6.4.6. Saving and Printing of Interface Notes

The interface does not currently save notes placed on slides, and any annotations created are lost when the browser window is closed. Incorporation of a note-persistence feature would require a server with authentication for students using the system, and facilities for storing notes in a database. Further support for printing of notes is an important feature for which further research is recommended. The system may collate annotations to produce sorted output of notes, or may simply print the slides verbatim as they appear on screen with popup notes.

7. Conclusion

Three research goals were established in section 3.2. The construction of a low-cost, flexible media processing framework based on the concepts of modularity, multi-platform output, capture location independence, and a non-invasive approach, has been presented in this research. The architecture of the system is currently based around three basic streams of video, audio and documents, each of which has its own preprocessor and indexer. Support for technology evolution has been incorporated into the framework with number of modules in the system containing databases of supporting components, effectively allowing the system to use an unlimited number of replaceable compression algorithms, muxing techniques, document parsers and stream muxers. The architecture design has been successfully implemented, and while the Indexing component of the system has not been fully implemented due to complexity and time limitations, elements of the module were successfully implemented including a functional document indexer capable of extracting keywords from PDF and PostScript files.

The extensibility of the framework has been discussed, and the integration of new data streams has been illustrated in a process designed to require modification to only one or two framework components.

The second goal of establishing a set of general interface guidelines has been achieved, and this research has proposed seven guidelines covering requirements of synchronised streams, availability of search facilities, and the use of a flexible delivery medium.

The design and implementation of an interface incorporating these guidelines was the third goal, and has been carried out. The system incorporated emerging technologies such as CSS layers and SMIL. The validity of six of the seven guidelines has been established through an informal evaluation involving the observation of the behaviour of 18 participants. The evaluation has resulted in a number of suggestions for improving the interface, some of which will require modifications to framework modules to implement. Observations from participants included a reliance on the indexing facilities of the system and lack of clarity in the availability of the annotation features in the interface. These issues should be addressed in the next development iteration of this prototype.

The suggestion of the inclusion of a student-created index in the interface will have a significant impact on the framework. In the initial stages of this research, it was felt that the functions performed by the indexer were some of the most important in the framework, and that the creation of a comprehensive set of index points would be the defining aspect of the system. However, the suggestion of an outline created by students (and the ability to drag thumbnails out of the sidebar) could have important learning implications, as it would encourage students to watch the lecture and formulate their own list of entries that would support revision during a future session. Further, it is foreseeable that some lecturers may be reluctant to have their lectures indexed, and the provision of a comprehensively indexed lecture may defeat some deliberate teaching methods such as the use of "skeleton notes." Incorporation of a customisable outline could allow the Digital Lecture system to be used as a course learning tool, where, by default, the Outline window contains only those entries created by students. Thus, the implementation of a comprehensive indexer in the framework may be unnecessary if the system is used as a course tool, rather than as a pure revision tool as was the intent at the outset of this research. To support a customisable outline, the 'save' feature of the interface becomes of equal importance, and the system would require a method of differentiating between individual students. Automatic indexing is a feature

that could be disabled in the framework according to how lecturers would like it to be used in their course. While it was noted that the social implications of this system were not within the score of the research, further evaluation in this area is strongly recommended to assess the effectiveness of an outline created by students on learning.

In addition to the evaluation suggested above, a number of future directions for research are recommended. Specifically, the complete implementation of the framework Indexer and the incorporation of improved search and persistence capabilities in the interface.

References

1. Visibly Superior Video Quality White Paper, Domino FX, http://www.dominofx.com/technology/DFX_White_Paper.pdf.

2. Abowd, G. Classroom 2000: An experiment with the instrumentation of a living educational environment. *IBM Systems Journal, 38* (4). 508-530.

3. Abowd, G., Atkeson, C., Feinstein, A., Hmelo, C., Kooper, R., Long, S., Sawhney, N., Tani, M., Teaching and Learning as Multimedia Authoring: The Classroom 2000 Project. in *ACM Multimedia '96*, (1996), 187-198.

4. Adjeroh, D., Bell, T., Cockburn, A., McKenzie, B., Vargo, J. Flexible delivery using Digital Lectures, University of Canterbury, Christchurch, 2001.

5. Amir, A., Srinivasan, S., Ponceleon, D., Petkovic, D., CueVideo: Automated video/audio indexing and browsing. in *SIGR '99*, (1999), ACM Press.

6. Battista, S., Casalino, F., Lande, C. MPEG-4: A Multimedia Standard for the Third Millenium. *IEEE Multimedia, 6* (4).

7. Bell, T., Cockburn, A., McKenzie, B., Vargo, J. Flexible Delivery Damaging to Learning? Lessons from the Canterbury Digital Lectures Project.

8. Bianchi, M., AutoAuditorium: a Fully Automatic, Multi-Camera System to Televise Auditorium Presentations. in *Proceedings of the Joint DARPA/NIST Smart Spaces Workshop*, (Gaithersburg, MD, 1998).

9. Gonzalez, J., Zhang, H., Greenberg, O. Lecture Browser: A Webcast Lecture Capture and Indexing System, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2000.

10. Hill, R., Cruz, G., Capturing and Playing Multimedia Events with STREAMS. in *ACM Multimedia '94*, (San Francisco, CA, 1994), 193-200.

11. Hindus, D., Ackerman, M., Mainwaring, S., Starr, B., Thunderwire: A Field Study of an Audio-Only Media Space. in *ACM Conference on Computer-Supported Cooperative Work (CSCW'96)*, (1996), 238-247.

12. Kameshiro, T., Hirano, T., Okada, Y., Yoda F., A Document Retrieval Method from Handwritten Characters Based on OCR and Character Shape Information. in *Sixth International Conference on Document Analysis and Recognition (ICDAR '01)*, (Seattle, Washington, 2001).

13. Karunaratne, P., Segall, C., Katsaggelos A., A Rate-Distortion Optimal Video Pre-Processing Algorithm. in *ICIP 2001*, (2001), IEEE.

14. Li, H., Doermann, D., Kia, O. Automatic text detection and tracking in digital videos. *IEEE Transactions on Image Processing, 9* (1). 147-156.

15. Liu, Q., Rui, Y., Gupta, A., Cadiz, J. Automating camera management for lecture room environments, Microsoft Research, Redmond, 2000.

16. Manmatha, R., Han, C., Riseman, E., Croft W., Indexing Handwriting Using Word Matching. in *1st ACM International Conference on Digital Libraries*, (1996).

17. Mu, X., Marchionini, G. Interactive Shared Educational Environment (ISEE): Design, Architecture, and User Interface, School of Information and Library Science, University of North Carolina, Chapel Hill, 2002.

18. Mu, X., Marchionini, G., Pattee, A., The Interactive Shared Educational Environment: User interface, system architecture and field study. in *JCDL'03*, (2003).

19. Mukhopadhyay, S., Smith, B., Passive capture and structuring of lectures. in *ACM Multimedia '99*, (Orlando, Florida, 1999), 477-487.

20. Nevill-Manning, C., Reed, T., Witten, I. Extracting text from PostScript. *Software Practice and Experience, 28* (5). 481-491.

21. Raj, A. Automated Deployment of RTSP (Real Time Streaming Protocol) Video Files *Computer Science*, University of Houston, 2000.

22. Roccetti, M., Salomoni, P., A Web-based Synchronized Multimedia System for Distance Education. in *2001 ACM Symposium on Applied Computing*, (Las Vegas, NV, 2001), ACM Press, 94-98.

23. Rui, Y., Huang, T., Mehrotra S. Constructing Table-of-Content for Videos. *ACM Multimedia Systems, Special Issue Multimedia Systems on Video Libraries, 7* (5). 359-368.

24. Saraceno, C., Leonardi, R., Speaker Dependent Video Indexing based on Audio-Visual Interaction. in *Proceedings of ICIP'98*, (1998), 358-362.

25. Shneiderman, B. *Designing the user interface.* Addison-Wesley, Menlo Park, 1998.

26. Sun, X., Foote, J., Kimber, D., Manjunath, B., Panoramic video capturing and compressed domain virtual camera control. in *Proceedings of the ninth ACM international conference on Multimedia*, (2001), ACM Press, 329-347.

27. Tourapis, A., Au, O., Liou, M., DeInterlacing techniques with the use of Zonal Based Algorithms. in *Visual Communication and Image Processing 2001 (VCIP-2001)*, (San Jose, CA, 2001).

28. Tsekeridou, S., Pitas, I., Audio-Visual Content Analysis for Content-Based Video Indexing. in *ICMCS*, (1999), 667-672.

29. Tsekeridou, S., Pitas, I. Content-based video parsing and indexing based on audio-visual interaction. *IEEE Transactions on Circuits and Systems for Video Technology, 11* (4). 522-535.

30. Wactlar, H., Christel, M., Hauptmann, A., Gong, Y. Informedia Experience-on-Demand: capturing, integrating and communicating experiences across people, time and space. *ACM Computing Surveys (CSUR), 31* (2es).

31. Whitten, I., Moffat, A., Bell, T. *Managing Gigabytes.* Morgan Kaufmann Publishers, Inc., 1999.

32. Wu, V., Manmatha, R., Riseman, E., Finding Text in Images. in *International Conference on Digital Libraries*, (Philadelphia, PA, 1997), ACM Press, 1-10.

33. Xie, W., Shi, Y., Xu, G., Xie, D. Smart Classroom: An Intelligent Environment for Tele-education. *Lecture Notes in Computer Science, 2195.*

34. Young, S., Foote, J., Jones, G., Sparck, K., Brown, M., Acoustic Indexing for Multimedia Retrieval and Browsing. in *ICASSP*, (1997).

Appendix A

When would you expect to use this system? (tick all that apply)

- ❑ If I missed a lecture
- ❑ Immediately following a lecture I attended
- ❑ For exam and test revision
- ❑ For general interest in the lectures of other courses
- ❑ I wouldn't use this system

How often would you watch the same lecture with this system? (tick one)

❑ Never ❑ One viewing only ❑ A few viewings ❑ Many viewings

Where would you most prefer to use this system? (tick one)

❑ On campus ❑ At home ❑ Both ❑ I wouldn't use this system

How useful did you find the **video** in performing the tasks? (tick one)

❑ Very useful ❑ Moderately useful ❑ Not really useful

How would you rate the quality of the **video**? (tick one)

❑ Excellent ❑ Good ❑ Average ❑ Fair ❑ Poor

Would you prefer the video to be (tick one)

❑ Larger? ❑ Smaller? ❑ The same size it is now?

How useful did you find the **audio** in performing the tasks? (tick one)

❑ Extremely useful ❑ Useful ❑ Occasionally useful ❑ Not at all useful

How useful did you find the large screen version of the PowerPoint slides? (tick one)

❑ Extremely useful ❑ Useful ❑ Occasionally useful ❑ Not at all useful

How often would you expect to make **notes** with this system (instead of paper versions)?

❑ All the time ❑ Frequently ❑ Occasionally ❑ Never

How useful did you find the search facility?

❑ Extremely useful ❑ Very useful ❑ Occasionally useful ❑ Not at all useful