LecTix: A Lecture-Multimedia Player

by

Timothy D. Olsen

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2005

Author .....
Department of Electrical Engineering and Computer Science
June 7, 2005

Certified by .....
Charles E. Leiserson
Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

LecTix: A Lecture-Multimedia Player

by

Timothy D. Olsen

Submitted to the Department of Electrical Engineering and Computer Science
on June 7, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

LecTix 2.0 is a multimedia player designed specifically for the playback of recorded classroom lectures. LecTix 2.0 plays multimedia consisting of synchronized audio, video, and PowerPoint-style slides. In addition to controls commonly found in multimedia players, LecTix 2.0 features controls designed specifically for lecture-multimedia playback such as customizable skip, variable-speed playback with pitch-normalization, and a browsable timeline of slides.

LecTix 2.0's features contribute to its being usable, widely available, and extensible. LecTix 2.0's automatic media synchronization and large, externally consistent controls for navigation make for a learnable, memorable, and efficient user interface. LecTix 2.0's open-source implementation using the Java Media Framework allows it to be freely distributable, portable, and convenient to use without a network connection. LecTix 2.0's media class hierarchy, events, and lecture description result in a modular player that can be extended to support new media types without recompilation of the player's core.

In addition to presenting LecTix 2.0, this thesis reviews seven players in use today. I compare them to LecTix 2.0 in terms of usability, availability, and extensibility. I also present a case study of the production of lecture multimedia and the use of an early version of LecTix in an introductory algorithms course.

Thesis Supervisor: Charles E. Leiserson
Title: Professor

Acknowledgments

I have taken a round-about journey to get to this point, but I have finally made it. Without people to support and encourage me, I wouldn't have made it this far.

I thank my advisor, Charles Leiserson, for his patience, advice, and insight. He's made a better writer out of me. Professors like him are what make MIT so great.

I thank my parents and my sister for their never ending love and support, and for knowing not to bother me while I wrote "the thesis."

Members of the Supercomputing Technologies Group provided feedback on LecTix and friendship. I thank Kunal Agrawal, Elizabeth Basha, Michael Bender, John Danaher, Leigh Deacon, Jeremy Fineman, Zardosht Kasheff, Angelina Lee, Bradley Kuszmaul, Sid Sen, Gideon Stupp, and Jim Sukha.

I thank the people who have worked on LecTix and those who continue to do so. In particular, I thank Kai Huang and Luis F. G. Sarmenta. Luis also provided well thought-out advice on the thesis document.

I thank Charles Leiseron, Tomas Lozano-Perez, and the Singapore-MIT Alliance for funding my research. This project never would have gotten done without the support.

Rob Miller's course on user interface design and implementation was essential to LecTix 2.0's implementation. I thank him and Min Wu for imparting their knowledge to me and offering their invaluable feedback. I also thank the students who performed a heuristic evaluation of what was an atrocious prototype of LecTix: Alex Faaborg, Albert Leung, Andrew Perelson, and Anson Tsai.

Numerous other people provided feedback on LecTix. I especially thank Tom Lasko, Larry Rudolph, and Bill Thies. Bill Thies also took slide timings for the course *6.046: Introduction to Algorithms*.

I thank the people from MIT's Academic Production Media Services for capturing lecture videos and providing copies for me. I especially thank Joanne Flood, Craig Milanesi, and Peter Hess.

I thank Alejandro Caro for being a great and patient manager while I worked at

Akamai Technologies. He helped me to become a better person.

I thank my friends who have stuck with me through thick and thin: Greg Anderson, Michael Collier, Rachel Craig, Alex Dzindolet, Gene Fierro, Lizzy Hickey, Effie Hios, Oliver James, Rhiannon Jordan-Woodbury, Matt Lalime, Gabe Law, Betty Li, Celeste Loetz, Mike McGlothin, Vicky Medina, Erick Medina, Diego Medina, Becca Nathan, Ernesto Pascaul, Kenny Perry, Ian Scott, Jaime Tayag, Kat Troncellito, Brett Wiesner, and James Williams. These folks have always kept it real. They're my family.

I thank Xiaolu Hsi, Marcia Yousik, and Kim McGlothin for their support. I never would have made it this far without them.

And I thank MIT for generous financial aid and for teaching me how to use my brain. I graduate smarter and more knowledgeable than I was ten years ago.

Images of PowerPoint slides shown in Figures 1-2, 2-1, 2-2, 4-4, 5-4, and 5-7 are reproduced by permission from Charles E. Leiserson.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Distance education and e-learning hold the promise of anytime, anywhere education. These technologies can be essential for learners place-bound by factors such as employment, child-care demands, disability, or remoteness of the location where they live [39]. Other learners may simply prefer not to have their schedules constrained by a class or a tutor. In this context, the lecture-multimedia player has arisen as a fundamental e-learning tool.

From Multimedia Players to Lecture-Multimedia Players

To understand what a lecture-multimedia player is, we must first understand what a multimedia player is. Figure 1-1 shows RealPlayer [38], a popular multimedia player. Like most multimedia players, RealPlayer is designed primarily for listening to music or online radio, and for watching movie trailers, music videos, news clips, and the like.

RealPlayer's user interface reflects this design. A large portion of space is devoted solely to the video. Controls lie along the bottom of the player, allowing the user to play, pause, seek to any point in time, and adjust the volume.

Lecture-multimedia players extend the capabilities of regular multimedia players by offering additional features geared toward viewing lectures. They play audio and video like other multimedia players; however, they also present additional multimedia relevant to the lecture (such as PowerPoint-style slides), and extra controls for quick

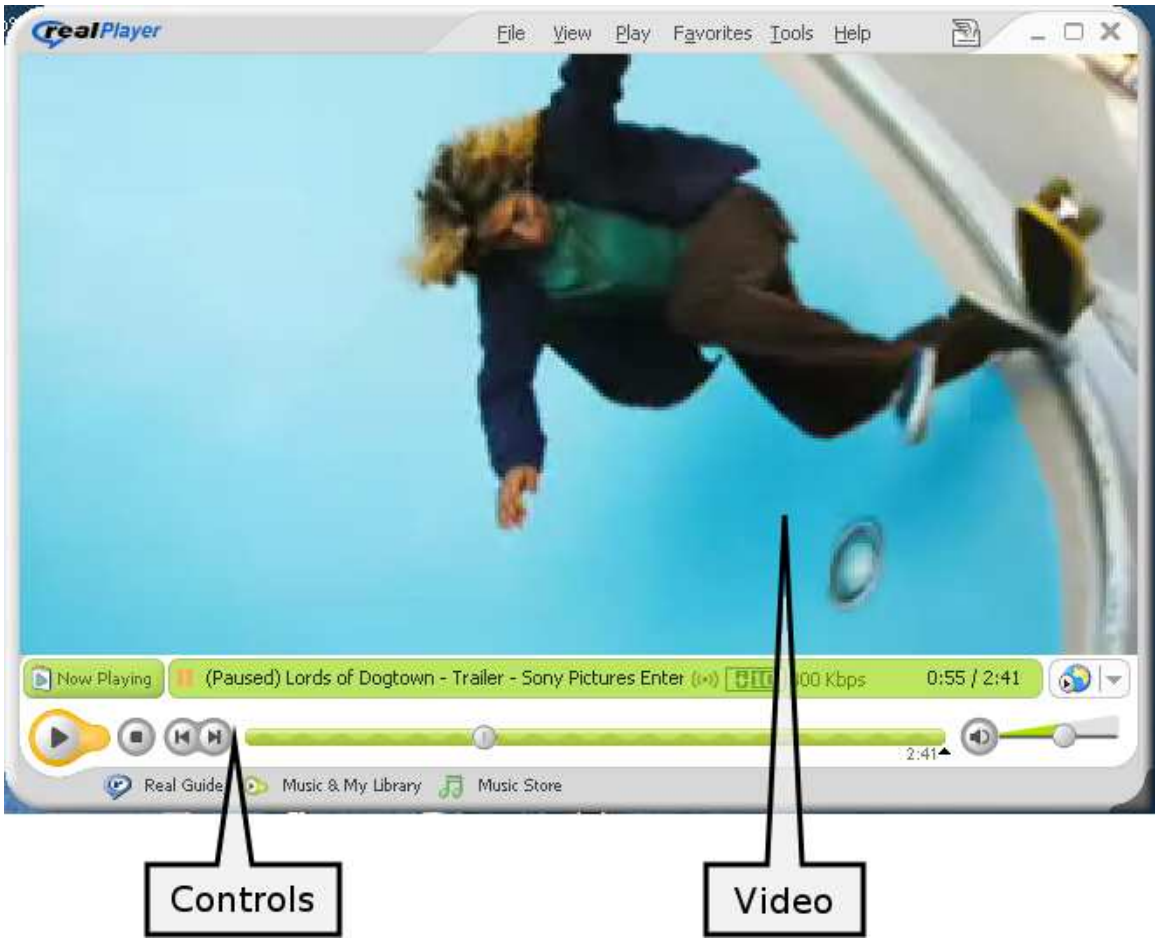

Figure 1-1: RealPlayer: a multimedia player by RealNetworks.

navigation. Lecture-multimedia players feature extra navigation controls because students often want to watch a specific part of a lecture.

1.1 LecTix 2.0: A Lecture-Multimedia Player

This thesis presents LecTix 2.0, a lecture-multimedia player I designed and implemented to be usable, "available," and extensible.

LecTix 2.0's user interface is designed specifically for interactive viewing of lecture-multimedia. Shown in Figure 1-2, LecTix 2.0 plays audio, video, and accompanying PowerPoint-style slides. The user interface also includes a control panel and a browsable time line of the slides. In addition to the controls common in regular multimedia players for playing, pausing, seeking, and adjusting volume, LecTix 2.0 has controls for skipping forward or backward a customizable number of seconds, browsing through the slides, and adjusting playback speed.

LecTix 2.0 is a product of the LecTix project, a research effort to design and implement a lecture-multimedia player. The LecTix project identifies three properties that a lecture-multimedia player should have:

- *Usability*: The lecture-multimedia player should be easy, practical, and pleasant to use.

- *Availability*: Students should be able to obtain the player at a low cost (ideally, free), and use it to view a lecture anywhere, anytime, in any format, and on the computing platform of their choice.

- *Extensibility*: It should be possible to add new features—in particular, support for new media types. Ideally, adding new media types should not require recompilation (a new release) of the player.

This thesis shows how the design and implementation of LecTix 2.0 attempts to attain each of these three properties. Overall, it successfully does so; LecTix 2.0 is usable, extensible, and moderately available.

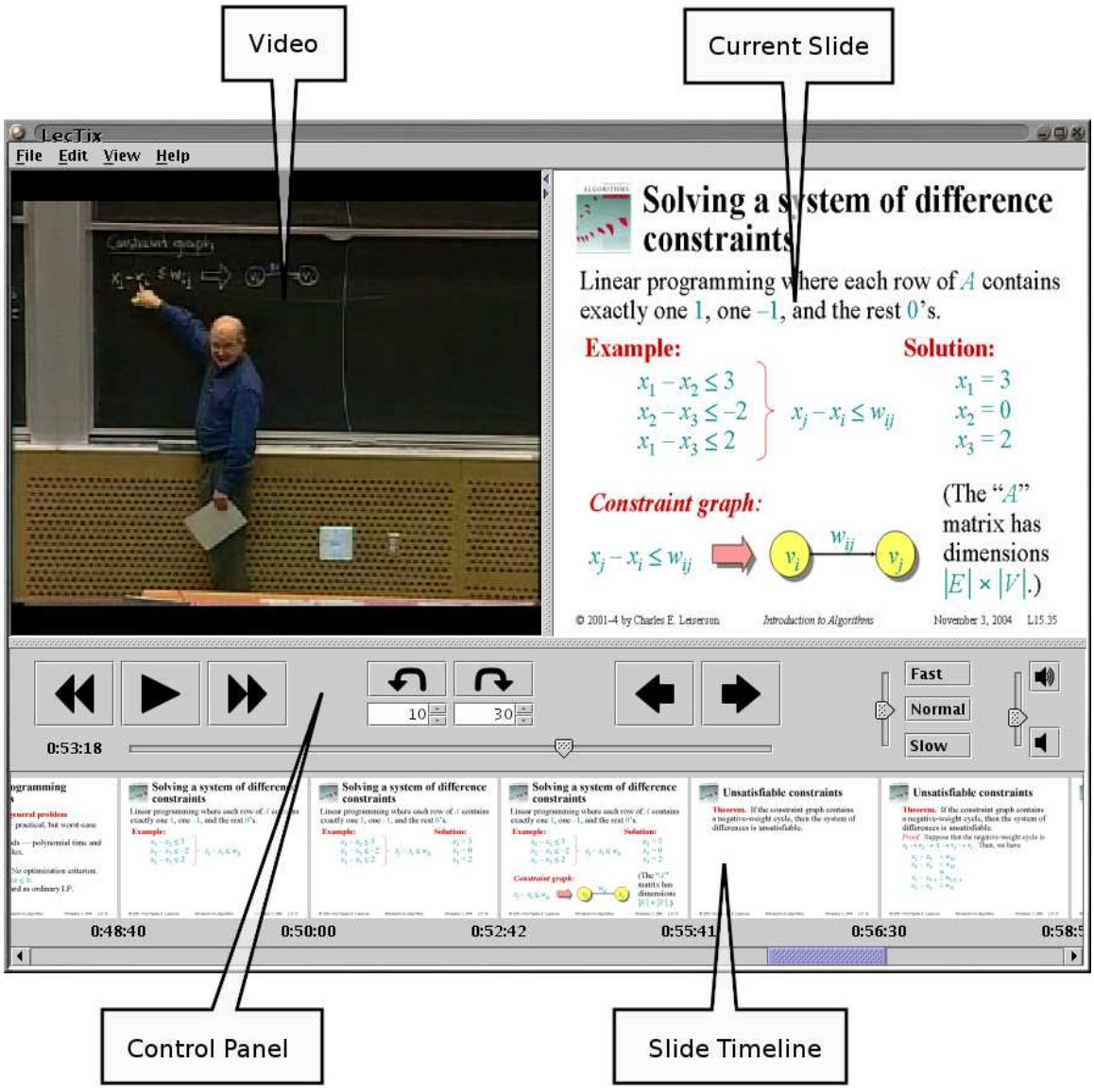

Figure 1-2: LecTix 2.0's main components.

LecTix 2.0 successfully achieves usability and extensibility. LecTix 2.0 achieves usability by offering an automatically synchronized presentation and large controls for navigation that are consistent with the real world. LecTix 2.0 achieves extensibility by being open-source and featuring a decoupled, modular class hierarchy with support of the addition of new media types without recompilation of the LecTix core.

With regard to availability, however, LecTix 2.0 is only moderately successful. In the United States and other countries where ideas in software can be patented, a conflict of availability arises that pits the distributability and portability of a player against its compatibility with contemporary patented codecs. LecTix 2.0 gives up some compatibility so that it can be a free, open-source, portable player.

1.2 Organization of the Thesis

This thesis contains seven chapters, a glossary, and a bibliography.

Chapter 2: Usability This chapter describes the features of LecTix 2.0 that contribute to its usability. Usability is broken down into five attributes as identified by Nielsen [31]. Each attribute is shown to be addressed by a feature in LecTix 2.0. The chapter also presents LecTix 2.0's controls and compares different methods of media synchronization.

Chapter 3: Availability This chapter describes the features of LecTix 2.0 that affect its availability. Availability is broken down into four attributes, and the features of LecTix 2.0 that affect each of the four attributes are discussed. The chapter also gives an overview of the restrictions and fees associated with the implementation and distribution of state-of-the-art codecs (MPEG-4, RealVideo, Sorenson, and Windows Media Video) and why these restrictions and fees pose problems for open-source players.

Chapter 4: Extensibility This chapter describes the features of LecTix 2.0 that contribute to its extensibility. Extensibility is shown to consist of three attributes,

each of which are addressed by features of LecTix 2.0. Features that are discussed include the media-class hierarchy, event mechanism, and lecture description.

Chapter 5: Related Work This chapter reviews seven lecture-multimedia players. It compares the features of the seven players and LecTix 2.0, and discusses how those features affect the usability, availability, and extensibility of each player.

Chapter 6: LecTix 1.3 Case Study This chapter discusses the use of the prior version of LecTix, LecTix 1.3, in the course, *6.046: Introduction to Algorithms*. The chapter also describes the methods used to produce lecture multimedia for the class.

Chapter 7: Conclusion This chapter concludes with comments on the contributions of LecTix 2.0 and the LecTix project. Ideas for future work are also presented.

Glossary The glossary defines terms that might be unknown to readers unfamiliar with distance education or multimedia.

Bibliography The bibliography lists works that have enabled me to "stand on the shoulders of giants" [7].

Chapter 2

Usability

While usability is important in any software that interacts with a user, lecture-multimedia players place a greater than usual emphasis on the user due to its focus on the user interface. Other features such as reliability and performance directly impact the usability of a player. This chapter looks at the five attributes that compose usability as defined by Nielsen [31] and demonstrate how LecTix 2.0 contributes to each of these five attributes, hence making it a usable lecture-multimedia player.

2.1 Attributes of Usability

Overall, usability is the degree to which a system is easy, practical, and pleasant to use. The attributes of usability as defined by Nielsen [31] are:

- *Learnability*: The degree to which the system is easy to use.
- *Efficiency*: The degree to which the system can be used efficiently, once the user has learned the system.
- *Memorability*: The degree to which it's easy to remember how to use the system, even if used infrequently.
- *Few and Noncatastrophic Errors*: The degree to which there are few errors, the degree to which those errors are discovered by the user, and the degree to which

Table 2.1: LecTix 2.0 features that contribute to usability

Feature	Learnability	Efficiency	Memorability	Few Errors	Satisfaction
Large Controls	—	✓	—	✓	—
Real-World Controls	✓	—	✓	✓	—
Navigation	—	✓	—	—	✓
Synchronization	—	✓	—	✓	✓
Variable-Speed	—	✓	—	✓	✓

those errors do not destroy the user's work.

- *Satisfaction*: The degree to which the system is pleasant to use.

These five attributes compose usability. A player that addresses all five issues is considered usable.

2.2 Features of LecTix 2.0 that Contribute to Usability

Five features of LecTix 2.0 contribute to the five attributes of usability. These five features are (1) large controls, (2) controls that match the real world, (3) controls for navigation, (4) media synchronization, and (5) variable-speed playback. Table 2.1 shows which usability attributes each feature contributes to.

2.2.1 Large Controls

Large controls contribute to high efficiency and few errors. Large controls are more quickly hit than small controls. Furthermore, users are less likely to accidentally miss hitting them.

LecTix 2.0's large controls take advantage of Fitts's Law [12] so that they can be more quickly hit, thereby improving efficiency. Fitts's Law states that the time to move the pointer to a target is proportional to the logarithm of the ratio of the width of the object to its distance from the pointer. MacKenzie proposed the slightly

different but more accurate Shannon formulation of Fitts's Law [26]:

$$T = a + b \log_2 \left(\frac{D}{W} + 1 \right), \tag{2.1}$$

T is the average time to move to the target, a and b are empirically determined constants, D is the distance to the center of the target, and W is the width of the target measured along the axis of motion. As we can see, increasing W, the width of the target, decreases T, the time to hit the target.

Large icons also reduce the probability of an error occurring. An error occurs when the user, thinking he or she has hit the target, actually misses the target and clicks the mouse anyway. In Equation 2.1, W is the maximum distance (along the axis of motion) from the center of the target that the user can position the pointer to correctly hit the target. Card, Moran, and Newell showed Fitts's Law could be derived by modeling the movement to the target as a series of successive movements until the pointer hits the target [5]. Each successive movement carries with it a probability that the user misses the target. Once the user thinks he or she has hit the target, movement can end. Therefore, as W increases, the user needs fewer successive movements on average to hit the target. With fewer chances for the user to mistakenly think he or she has hit the target, the probability of an error occurring decreases.

2.2.2 Externally Consistent Controls

Controls that match the real world, or externally consistent controls, contribute to high learnability, high memorability, and few errors. Users can quickly learn, easily remember, and correctly interpret the purpose of these controls.

The labeling of LecTix 2.0's controls matches the user's intuition well, allowing the user to quickly learn and remember their functions. Figure 2-1 shows how LecTix 2.0's rewind and fast-forward buttons compare to those of RealPlayer. LecTix 2.0 labels the rewind and fast-forward buttons consistent with the VCR. RealPlayer, on the other hand, relegates the *rewind* and *fast-forward* to be secondary functions of the previous-clip and next-clip buttons. To rewind or fast-forward, users must hold down

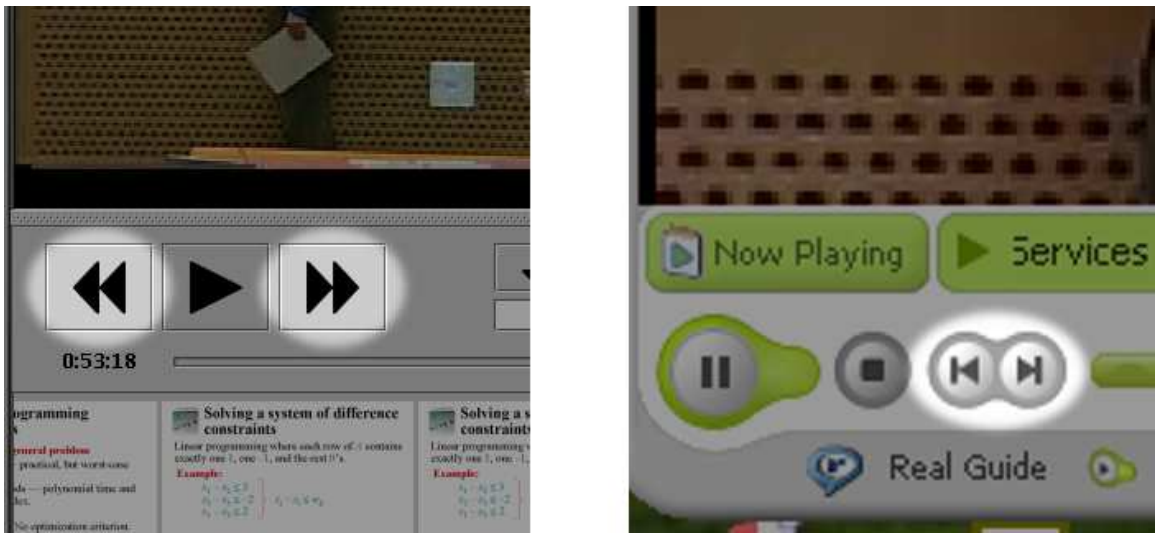

Figure 2-1: LecTix 2.0's rewind and fast-forward buttons (left) better match the real world than RealPlayer's (right). Despite rewind and fast-forward being useful functions for lecture viewing, RealPlayer maps the buttons' primary functions to *previous-clip* and *next-clip*. To rewind or fast-forward, the user must hold down one of the buttons—a procedure not readily apparent. Many users may not even realize that RealPlayer can rewind or fast-forward.

the previous-clip or next-clip button, respectively. It may not be apparent to do so, potentially leaving many users unknowledgeable that the player can rewind or fast-forward.

2.2.3 Navigation Controls

Controls for navigation contribute to high efficiency and satisfaction. A variety of controls for navigation ensures an appropriate control is used for the task at hand. Using the appropriate control increases efficiency and decreases frustration which leads to increased user satisfaction.

To get an idea of the range and versatility of LecTix 2.0's controls, I describe each control in turn and then present scenarios for which a particular control shows most useful.

Shown in Figure 2-2, LecTix 2.0 provides controls typical of multimedia players, as well as controls designed specifically for lecture viewing. The controls of LecTix 2.0 typically found in multimedia players include play, pause, rewind, fast-forward buttons, and seek and volume sliders. The controls designed specifically for

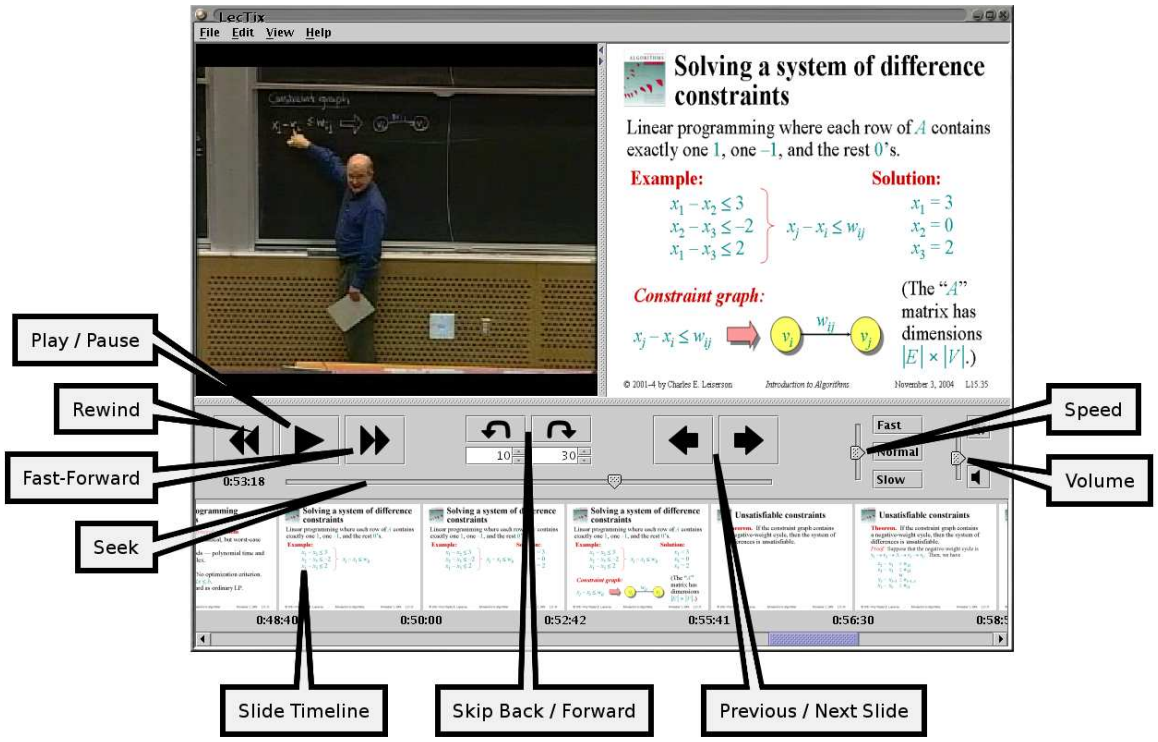

Figure 2-2: LecTix 2.0 controls.

lecture-viewing are customizable skip-back and skip-forward buttons; previous- and next-slide buttons; an interactive slide timeline; and a speed slider with shortcuts for slow, normal, and fast settings.

As is common in many media players, there is one button to play and pause. This button toggles between a *play* state and a *pause* state.

Unlike many media players (but similar to many VCRs), the rewind and fast-forward buttons are also toggle buttons. Clicking on the rewind button, for example, begins rewinding. The user can then click on rewind button or play button to stop rewinding. This interface removes the need for the user to hold down a button while rewinding or fast-forwarding.

While rewinding or fast-forwarding, LecTix 2.0 moves the video along accordingly. This feature is an improvement over RealPlayer (Figure 1-1, page 14) which pauses the video until rewinding or fast-forwarding is completed.

The seek slider allows the user to quickly jump to any point in the lecture, but

with low accuracy.

The skip-back and skip-forward buttons enable the user to skip back or forward a specific amount of time. Underneath the skip-back and skip-forward-buttons are two dials for customizing the number of seconds to skip.

To the right of the skip buttons are the previous- and next-slide buttons. These allow the user to quickly scan the topics in a lecture.

Another control related to the slides is the slide timeline which sits along the bottom of the user interface. Users can click on a slide to go to the relevant point in time in the lecture. The slide timeline also serves as media, presenting a view of the slide through thumbnails interspersed with the times at which slide transitions occur.

The speed slider allows the user to alter the playback speed of the lecture multimedia. To prevent the audio from sounding too high or low, LecTix 2.0 shifts the pitch of the audio back to normal. Three buttons serve as shortcuts for *slow*, *normal*, and *fast* settings.

The slider to the very right controls the volume. LecTix 2.0 provides a shortcut button for muting the audio, and another for setting it to full volume.

Most of LecTix 2.0's controls' main purpose is navigation. These controls include play, pause, rewind, fast-forward, skip-back, skip-forward, previous-slide, next-slide, the slide timeline, and the speed slider. Nonetheless, each of these controls have a somewhat different purpose, and they are used appropriately in different situations. Listed below are some scenarios for which different controls may be used.

- Start watching the lecture: play button.
- Take a break: pause button.
- Course-grained scan: seek slider or skip-forward button.
- Fine-grained scan: rewind and fast-forward buttons.
- Repeat something that was unclear (instant replay): skip-back button.
- Skip-over uninteresting events: skip-forward button.

- Scan the topics: previous-slide and next-slide buttons.
- Select a topic: slide timeline.
- Quickly watch or review material: speed slider (fast setting).
- The lecturer is speaking too quickly in one of the user's non-native languages: speed slider (slow setting).

By having controls specialized for common tasks, users can efficiently watch lectures and learn. In turn, users experience less frustration, leading to greater overall satisfaction with LecTix 2.0.

2.2.4 Media Synchronization

A feature typical of lecture-multimedia players is the synchronization of multimedia. Synchronization ties together individual media streams, ensuring each one continually presents content relevant to the other media streams' content.

Automatic synchronization of media contributes to high efficiency and few errors. It relieves the user of the burden of synchronizing the multimedia on their own, allowing them to devote more attention to the lecture and eliminating errors that could occur during manual synchronization.

In general, lecture-multimedia consist of two types of media: continuous and discrete. Continuous media, such as video and audio, frequently change with time. Discrete media, such as slides, change infrequently with time. Typically, a lecture-multimedia player links to a library that automatically synchronizes continuous media to a clock. In contrast, the player, if it supports synchronization, must directly synchronize discrete media. Therefore, the problem of synchronization from the point of view of the player is to synchronize discrete media and any clocks that continuous media synchronize to. (See Section 4.2.3, page 47, for a discussion on how LecTix 2.0 implements continuous and discrete media.)

Many lecture-multimedia players available today (see Chapter 5 for a review and comparison of several such players) feature some form of synchronization. Not all

players, however, provide automatic synchronization that operates correctly during user interaction. Some players do not provide any automatic synchronization at all, leaving synchronization to be performed manually by the user.

LecTix 2.0 implements a form of media synchronization I call segment synchronization. Segment synchronization correctly maintains synchronization in the face of user interaction. An inferior form of synchronization that may momentarily leave media unsynchronized during or after user interaction is called trigger synchronization. A complete lack of automatic synchronization (except for audio and video which are synchronized not by the player, but by their containing format) is called manual synchronization.

Manual synchronization

Other than possibly starting all media streams from the beginning when a multimedia presentation is loaded, a player that offers only manual synchronization makes no effort to synchronize media streams. It is therefore left to the user to synchronize media streams.

There is an advantage to leaving synchronization to the user: he or she can browse through media without jumping to that time in the lecture. The disadvantage, of course, is that watching the lecture requires constant maintenance on behalf of the user to ensure all media streams remain relevant to each other.

Another option is to allow synchronization—trigger or segment—to be turned off, enabling manual synchronization. An earlier version of LecTix, LecTix 1.3 (page 67), provides this option. The user can turn on synchronization when watching the lecture, and turn synchronization off while browsing through media. LecTix 2.0 does not have this option, but adds a slide time line for independent browsing.

Trigger synchronization

Trigger synchronization is the synchronization of multimedia only when the video plays *through* a certain point in time. In the example shown in Figure 2-3(a), a user plays a lecture from video frame 1 to video frame 7, during which the user does not

interact with the player. Under these circumstances, the transition from frame 3 to frame 4 triggers the transition from slide A to slide B. Likewise, the transition from frame 6 to frame 7 triggers the transition from slide B to slide C.

If the user navigates through the lecture while the lecture is playing (for example, by dragging the seek slider), then trigger synchronization may fail to keep the lecture synchronized. For instance, in Figure 2-3(b), if the user seeks directly from frame 2 to frame 5, then the player misses the transition to slide B. The player should show slide B during frames 5 and 6, but does not because it never transitions from frame 3 to frame 4. The lecture resynchronizes, however, once the player transitions frame 6 to frame 7, causing a transition to slide C.

Segment synchronization

To stay synchronized in the face of user interaction, a player must support segment synchronization. Figure 2-3(c) illustrates the advantage of using segment synchronization. In this example, the current slide remains synchronized despite the user seeking from frame 2 to frame 5.

One way to implement segment synchronization uses a modified form of trigger synchronization. If, in addition to synchronizing at trigger points, the player also forces a synchronization every time the player makes a discrete jump in time—for example, the user seeks to a different point in the video—then the presentation stays correctly synchronized.

Another way to implement segment synchronization is by periodically forcing a synchronization of the video and slide streams. As long as the period between synchronizations is small enough, then the user sees correct synchronization of the multimedia.

We can define how small this time interval should be by comparing it to the cycle time of the model human's perceptual processor as developed by Card, Moran, and Newell [5]. If a corresponding video frame and slide appear within one cycle of the perceptual processor, then the model human perceives the two events as happening at the same time. Card et al. give a range for the duration of one cycle of the human

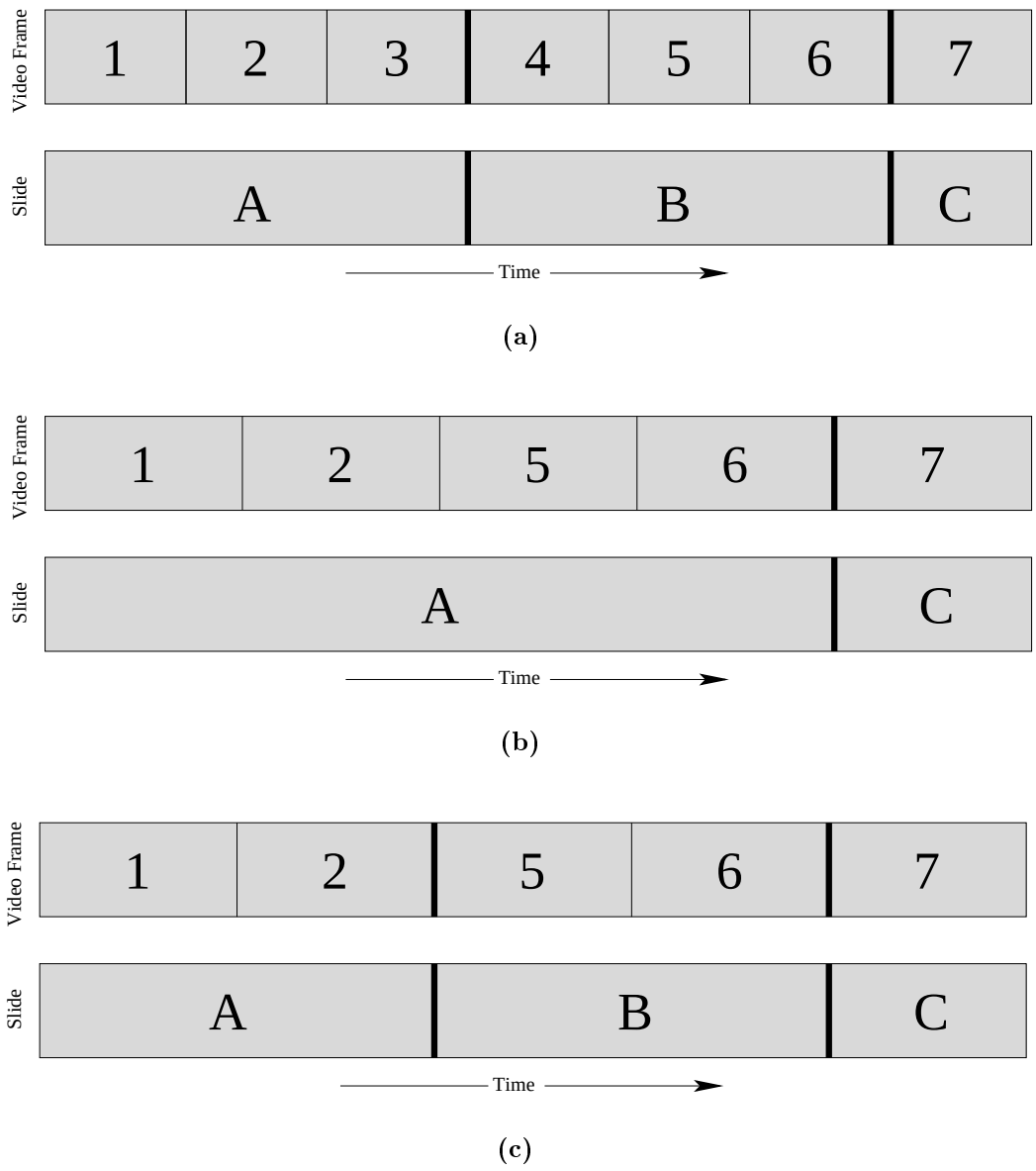

Figure 2-3: Segment synchronization succeeds where trigger synchronization fails. Each sub-figure depicts two concurrent media streams: a video stream and a sequence of slides. A sequence of numbered frames represents the video stream. Each slide is assigned a capital letter. Thin dividing lines represent transitions between video frames that do not cause a slide transition. Thick dividing lines represent video frame transitions that cause a slide transition. **(a)** Trigger synchronization properly transitions slides during normal playback without user navigation. The transition from video frame 3 to video frame 4 triggers a transition from slide A to slide B. **(b)** If the user drags the seek slider—jumping the video from frame 2 to frame 5—then trigger synchronization misses the transition to slide B. The lecture later resynchronizes when the transition from video frame 6 to video frame 7 triggers the transition to slide C. **(c)** Despite the user seeking to video frame 5, segment synchronization properly transitions to slide B.

perceptual processor as being between 50 and 200 milliseconds with an average of 100 milliseconds. The correct time of transition, however, may be any time from within a range of several seconds. Longer transition times are particularly well tolerated for solely blackboard-based lectures. Lectures with slides presented in the recorded video, however, have a narrower range of correct synchronization times such that the cycle time of the perceptual processor may become significant.

Although periodically forcing synchronization has some overhead, developers can implement it more easily and cleanly than a modified trigger synchronization. Trigger synchronization requires support from the video player to send notification when it reaches a trigger point. Periodically forcing synchronization, on the other hand, requires the operating system to send notifications (alarms) periodically—a feature found in most operating systems—and that synchronization calls are cheap, especially in the case where no transition should occur.

LecTix 2.0 implements segment synchronization by the latter method. Discrete media and the continuous media's clocks synchronize every 300 milliseconds. While outside of the range given by Card et al. for the period of the cycle time of the human perceptual processor, infrequent calls to synchronize present a low load to the computer's processor.

2.2.5 Variable-Speed Playback

Variable-speed playback contributes to high efficiency, few errors, and high satisfaction of use of LecTix 2.0. Whether users wish to review a lecture at fast pace, or slow down the lecture so that they can understand the lecturer clearly, variable-speed playback allows users to watch the lecture at a pace that is natural for them.

Both playing a lecture fast or slow have advantages that contribute to usability. Users playing a lecture fast can finish watching the lecture more quickly than if they had watched it at a normal pace, thereby increasing efficiency. For users whose native language is the one spoken by the lecturer, playing the lecture slow decreases errors in listening.

Furthermore, informal feedback from students shows that the control for variable-

speed playback is fun to use, thereby increasing user satisfaction.

Variable-speed playback is usually accompanied by pitch-shifting of the audio back to its original pitch, otherwise known as pitch-normalization. LecTix 2.0 borrows Andrew J. Leiserson and Luis F. G. Sarmenta's implementation of pitch-normalization found in Lecture Viewer [18], a predecessor to LecTix. This implementation uses the synchronized overlap-add algorithm for time-scale modification of speech proposed by Roucus and Wilgus [40].

Chapter 3

Availability

A lecture-multimedia player is of limited use if it cannot be acquired at a low cost (after all, students tend to have a limited budget), and if it cannot play any lecture multimedia, anywhere, anytime, on the student's computing platform of choice. This thesis uses the term "availability" to refer to how successfully a user can access and effectively use a player in a variety of scenarios.

This chapter decomposes the property of availability into four attributes, and show how features in LecTix 2.0 affect each attribute. Because of an inherent conflict of availability between a player's price and compatibility with contemporary codecs, this chapter also gives an overview of the restrictions and fees associated with the implementation and distribution of contemporary codecs in countries where software can be patented, and discusses how these restrictions and fees pose problems for open-source players.

3.1 Attributes of Availability

Availability is the degree to which a user can acquire a player cheaply, and use it to play any lecture-multimedia, anywhere, anytime, and on any platform. Availability consists of four attributes:

- *Distributability*: The degree to which a player may be distributed cheaply and without restriction.

Table 3.1: LecTix 2.0 features that affect availability

Feature	Distributability	Portability	Convenience	Compatibility
GPL	✓	✓	—	—
Java	—	✓	—	—
JMF	✓	*	✓	×

* On one hand, the Java Media Framework (JMF) contributes to portability by providing playback for the Cinepak, MJPEG, and H.263 codecs across all Java-supported platforms. On the other hand, the JMF only supports playback for the MPEG-1 codec on select platforms.

- *Portability*: The number of platforms the player runs on, and how easily the software can be ported to other platforms.

- *Convenience*: The degree to which the player can play a lecture anywhere, anytime.

- *Compatibility*: The degree to which the player can play all lecture multimedia, encoded in any format or codec.

These four attributes compose availability. A player is available if it addresses all four issues.

3.2 Features of LecTix 2.0 that Contribute to Availability

As shown in Table 3.1, three features of LecTix 2.0 affect the four attributes of availability. These features are (1) its open-source license, the GNU General Public License; (2) the Java language, in which it's implemented; and (3) its use of the Java Media Framework. All three features contribute in some way to availability. Some facets of the Java Media Framework, however, detract from LecTix 2.0's portability and compatibility.

3.2.1 The GNU General Public License

LecTix 2.0's open-source license, the GNU General Public License (GPL) [13] contributes to LecTix 2.0's distributability and portability. As an open-source license, the GPL allows LecTix 2.0, its source code, and derived works to be freely redistributed. Furthermore, the GPL *requires* that the source code to any derived works be made available upon distribution. With access to the source code, developers can port LecTix 2.0 to the platform of their choice.

3.2.2 The Java Programming Language

LecTix 2.0's language of implementation, Java [17], contributes to portability. Programs written in the Java language can run on a wide variety of platforms, from cell phones to high-end servers.

Java owes its high portability to the Java Virtual Machine (JVM). The JVM is an abstract computing machine, emulated on real computing platforms by JVM implementations. Programs written in Java compile to JVM instructions, also known as bytecodes. The JVM is designed to be efficiently emulated, allowing it run efficiently on any platform.

Table 3.2 shows the wide range of desktop and server operating systems ported to by various JVM implementations. Table 3.2 lists 18 operating systems that can run a Java program. Many of the operating systems run on wide variety of hardware, further increasing the number of platforms with JVM implementations.

Not all of the operating systems listed in Table 3.2, however, can run LecTix 2.0. The Graphical User Interface (GUI) toolkit that LecTix 2.0 uses is not ported as widely as JVMs for the Java language. The Java Swing library is included with the JVM implementations represented by the *Vendor*, *IBM*, and *Sun* columns. Twelve of the operating systems listed have a port from one of those JVM implementations, and are expected to be able to run LecTix 2.0.

Table 3.2: Java Virtual Machine ports to desktop and server operating systems

Operating System	JVM Implementation						
	Vendor[a]	Blackdown[b]	GCJ[b]	IBM[c]	Jikes[b]	Kaffe[b]	Sun[c]
AIX	✓[d]	—	✓	✓[d]	✓	✓	
AmigaOS	—	—	—	—	—	✓	—
BeOS	—	—	—	—	—	✓	—
BSDi	✓	—	—	—	—	✓	—
FreeBSD	—	—	✓	—	✓	✓	✓[e]
Hurd	—	—	—	—	✓	✓	—
HP/UX	✓	—	✓	—	✓	✓	—
IRIX	✓	—	✓	—	—	✓	—
Linux	—	✓	✓	✓	✓	✓	✓
Mac OS X	✓	—	✓	—	✓	—	—
NetBSD	—	—	✓	—	✓	✓	✓[f]
NeXTStep	—	—	—	—	—	✓	—
OpenBSD	—	—	✓	—	✓	✓	✓[f]
Plan9	—	—	—	—	—	✓	—
Solaris	✓[g]	—	✓	—	✓	✓	✓[g]
SunOS	—	—	—	—	—	✓	—
Tru64	✓	—	✓	—	✓	—	—
Windows	✓	—	✓	✓	✓	✓	✓

Sources: Jikes [22], Pick [33], Schmidt [41], the GCC Team [15].

[a] Vendor of the operating system.

[b] Open-source project.

[c] Vendors that distribute JVMs for operating systems besides their own.

[d] Vendor and IBM are the same port.

[e] Ported by the FreeBSD Team.

[f] Using Linux binary emulation.

[g] Vendor and Sun are the same port.

3.2.3 Java Media Framework

The Java Media Framework (JMF) is a set of libraries that provide a multimedia framework for Java programs to work with. It provides implementations of various multimedia formats and codecs. LecTix 2.0 relies on the JMF for multimedia playback.

The Java Media Framework (JMF) is a mixed bag when it comes to availability. On one hand, the JMF contributes to distributability and convenience. But on the other hand, the JMF detracts from compatibility. When it comes to portability, various aspects of the JMF contribute to portability, while other aspects detract from it.

Overall, the JMF contributes to LecTix 2.0's distributability. Despite the source code to the JMF being publicly available, the JMF is not technically open-source because it imposes restrictions and legal liabilities upon its distribution. Such legal liabilities might discourage its distribution. Nonetheless, the JMF benefits the distributability of LecTix 2.0 because it is distributed separately by Sun Microsystems, a company able to negotiate, asses the risk of patent litigation, and cross-license patents if necessary.[1]

The JMF contributes to convenience by allowing multimedia to be played from the client machine. After downloading lecture multimedia from a server, a student can then watch the lecture without a network connection. The lifetime of lecture multimedia on a server (as well as the server's stability) may be limited, and a network connection may not always be available from a student's laptop. Storing the lecture-multimedia locally means it can be played anywhere, anytime.

The JMF both contributes to and detracts from portability. Table 3.3 shows a selection of video codecs supported by the JMF. On one hand, the JMF supports playback of the Cinepak, MJPEG, and H.263 [20] video codecs on any platform with an implementation of the Java Swing toolkit. On the other hand, the JMF only

[1]The Java Virtual Machine shipped by Sun is licensed in a similar fashion to the JMF. Of course, this can be distributed separately as well. Stallman [43] gives an excellent explanation for why patented ideas in software tend to only be legally usable by corporations with large patent portfolios.

Table 3.3: Java Media Framework video-codec support by platform

Video Codec	Windows/x86, Linux/x86, Solaris/Sparc	Mac OS X/PowerPC and Others[a]
Cinepak	✓	✓
MJPEG	✓	✓
MPEG-1	✓	—
H.263	✓	✓

Source: Sun Microsystems [45].

[a] That is, any platform with Java Virtual Machine and Java Swing toolkit implementations.

supports MPEG-1 [21] playback on the Windows/x86, Linux/x86, and Solaris/Sparc platforms. In practice, only the H.263 and MPEG-1 video codecs are of high enough quality for lecture videos,[2] leaving H.263 as the best codec for portability.

The JMF's limited options for the encoding of lecture videos (H.263 and MPEG-1) greatly detracts from LecTix 2.0's compatibility. State-of-the-art codecs found in contemporary multimedia players—such as RealVideo, MPEG-4, and Windows Media—are not supported by the JMF. Therefore, LecTix 2.0 cannot play lecture videos encoded in those codecs.

How Software Patents Cause a Conflict in Availability

LecTix 2.0's lack of compatibility with many contemporary codecs is unfortunate, but unavoidable in countries such as the United States where ideas in software can be patented.[3] Compatibility with patented codecs is often at odds with two other attributes of availability: distributability and portability.

Patented codecs often restrict the distributability of players that implement them. Use of a patented codec requires a license which may not necessarily be available. If a license is not available, a lecture-multimedia player must use the inventor's Application Programming Interface (API) or be based on the inventor's player. For example, a likely reason for the Singapore-MIT Alliance (SMA) distance education

[2] MJPEG can be of high quality, but it does not perform any inter-frame compression, resulting in very large file sizes.

[3] As of May 2005, bills for software patents are currently under legislation in the European Union and India.

program decision to base their player (see Section 5.4, page 60) on RealPlayer—as opposed to building their own player that can play the RealVideo codec—is that Real did not offer a license for the codec at the time. And in the case where a license is available, royalties are often required upon the codec's distribution. Furthermore—whether through the inventor's API, player, or license—restrictions are imposed on distribution. Such restrictions, coupled with royalties that may be required, detract from a lecture-multimedia player's distributability.

Patented codecs detract from a player's portability in the case when a license for the codec is not available. Often the inventor's API or player is ported to only select platforms, limiting the portability of the lecture-multimedia player that uses the inventor's API or player.

Nonetheless, despite incompatibility with many contemporary codecs, LecTix 2.0 achieves moderate availability by being distributable, portable, and convenient. The result is a free, open-source, and portable lecture-multimedia player.

3.3 Restrictions on Contemporary, State-of-the-Art Codecs

This section details the restrictions and fees associated with the use of contemporary codecs in the United States, and shows how these restrictions and fees directly impact distributability and portability. I present five formats and five codecs, and I discuss their restrictions with regard to royalties, open specifications, and API's. I also discuss four multimedia players, and show how restrictions on contemporary codecs limit their distributability and portability.

While contemporary formats and codecs go hand in hand, this section focuses mainly on contemporary codecs because they are the most restricted. Nevertheless, this section presents information on formats for completeness and because one of the formats requires a royalty.

I consider five pairs of contemporary formats and codecs. These pairs are presented

below. The format is listed first, followed by the codec, and then a description of the two.

Ogg, **Theora** The Xiph.Org foundation develops the Ogg container format and Theora [47] video codec. Theora is based on On2's VP3 video codec, which On2 has patented but irrevocably licensed to the public for free. Theora development is currently in a late alpha stage and is soon to go beta.

RealMedia, **RealVideo** RealNetworks develops the RealMedia container format and RealVideo video codec. The RealMedia format has an open specification and may be used for free; but the RealVideo codec is only available as a binary API and requires a royalty upon distribution.

MPEG-4, **MPEG-4** The MPEG-4 standard consists of several parts, two of which are a container format and a video codec. Both require royalties upon distribution, but the video codec does not require any royalties on the first 50,000 players distributed in a year.

QuickTime, **Sorenson 3** The QuickTime container format developed by Apple is similar to the MPEG-4 container format. QuickTime is an open specification like MPEG-4, but can be licensed for no charge [9]. Sorenson Communications develops the Sorenson 3 video codec. Sorenson 3 does not have an open specification, but Apple licenses binary API's for it at no charge.

ASF, **Windows Media Video 9** Microsoft develops the Advanced Systems Format (ASF) and the Windows Media Video 9 codec. ASF is an open format and can be licensed at no charge. Windows Media Video is not open, but a binary API is available at no charge for the Windows operating system.

3.3.1 Specifications, API's, and Royalties

Closed specifications (documentation on how to implement the format or codec), limited ports of API's, and royalties are the three features commonly found in contemporary codecs that detract from the distributability and portability of any player

Table 3.4: Openness of state-of-the-art formats and codecs

Format Codec	Open Specification	API available Linux[b]	Mac OS X[c]	Windows[b]	Royalty[a] ($)
ASF	✓	✓	✓	✓	0
Windows Media Video 9	×	×	×	✓	0
MPEG-4	✓	✓	✓	✓	0.15[d]
MPEG-4	✓	✓	✓	✓	0.25[e]
Ogg	✓	✓	✓	✓	0
Theora	✓	✓	✓	✓	0
QuickTime	✓	✓	✓	✓	0
Sorenson 3	×	×	✓	✓	0
RealMedia	✓	✓	✓	✓	0
RealVideo	×	✓	✓	✓	0.25[f]

[a] Per decoder distributed.
[b] Intel x86.
[c] PowerPC.
[d] $100,000 annual cap.
[e] Only payable after 50,000 units annually.
[f] $1,000,000 annual cap for non-Windows platforms.

that implements them. I show here which of these restrictions affect contemporary, state-of-the-art formats and codecs.

Table 3.4 shows the degree to which contemporary, state-of-the-art formats and codecs are restricted. The *Open Specification* column signals whether a specification is publicly available. The next three columns tell us whether an API is available for the Linux/x86, Mac OS X, and Windows/x86 platforms. The last column, *Royalty*, gives the royalty due per player distributed.

Table 3.4 tells us two unsurprising things. First, all formats and codecs have an API available for the Windows platform—not surprising given its desktop-market dominance. Second, any format or codec with an open specification has API's available for all three platforms. As with open-source software, an open specification for a format or codec naturally lends itself to be implemented on many platforms.

But one interesting thing Table 3.4 shows is that Ogg Theora is the only[4] contemporary, state-of-the-art codec available that has an open specification and is royalty

[4]Dirac [4] and the Snow codec (developed as part of the FFmpeg project [11]) are state-of-the-art, open-source, and royalty-free codecs that were only in their infancy at the time of this writing.

free. The Windows Media Video 9 and Sorenson 3 codecs do not charge royalties, but their specifications are closed, and their API ports are limited. RealVideo has API's [35] available for all three platforms, but its specification is closed and the API's require a royalties. MPEG-4, which has an open specification and API ports to all three platforms, charges royalties for both the container format and codec.

As we can see, most contemporary video codecs either restrict distributability due to royalties, portability due to closed specifications and limited ports of the API, or both.

3.3.2 Portability of Vendors' Players

Even though codec vendors may not publish the codec's specification or offer a developer API for every platform, they often offer their own players, usually at no cost. For the most part, these players are closed-source, which restricts outside developers from modifying the code, and porting it to new platforms. I show here how closed specification and closed source-code has affected the portability of vendors' players and the codecs they can play.

Table 3.5 shows the players[5] available from codec vendors for playing contemporary formats and codecs on the Linux/x86, Mac OS X/PowerPC, and Windows/x86 platforms. In the left-most column, the table lists these platforms and the players that run on them. The remaining columns indicate the format-codec pairs that the players can play.

None of the players can play all five formats and codecs across all three platforms. Only RealPlayer on the Windows platform can play all five format-codec pairs. In addition, RealPlayer is the only player ported to all three platforms.

Helix Player [37], the only open-source player listed, fares worse than any other player on Table 3.5: it plays only one of the format-codec pairs, and it runs on only one of the platforms listed. The conflict between distributability and compatibility, discussed on page 36, explains Helix Player's limited codec support. On the other

[5]Table 3.5 does not consider players such as MPlayer [34] or VLC [6] because they are distributed from Europe where software patents are not enforced.

Table 3.5: State-of-the-art formats and codecs: players and ports

	Format / Codec				
Platform / Player	Ogg / Theora	RealMedia / RealVideo	MPEG-4 / MPEG-4	QuickTime / Sorenson 3	ASF / WMV9
Linux / x86					
Helix Player	✓	×	×	×	×
RealPlayer	✓	✓	×	×	×
Mac OS X / PowerPC					
RealPlayer	×	✓	✓	✓	×
QuickTime Player	×	×	✓	✓	×
Windows Media Player	×	×	×	×	✓
Windows / x86					
RealPlayer	✓[a]	✓	✓	✓	✓
QuickTime Player	×	×	✓	✓	×
Windows Media Player	✓[b]	×	×	×	✓

[a] Requires Xiph Player Plugin [36]
[b] Requires Ogg Directshow Filters [23]

hand, Helix Player's apparent limited portability seems to contradict the notion that an open-source license contributes to portability.

Nothing could be further from the truth. Helix Player is an off-shoot of RealPlayer which initially targeted the Linux/x86 platform. Helix Player and RealPlayer share the same playback engine; the only difference is that RealPlayer can play patent-encumbered codecs with tricky licenses. The existence of RealPlayer on the Mac OS X/PowerPC and Windows/x86 platforms serves Helix Player's niche on those platforms, for now.

Hence, Helix Player developers have been concentrating their efforts on other platforms. Besides Linux/x86, Helix Player also supports the Symbian cell-phone platform. Ports are in progress to the Solaris/Sparc, Solaris/x86, HP-UX/PA-RISC, Linux/PowerPC, Linux/MIPS, Linux/Sparc, Linux/ia64, FreeBSD/x86, and AIX/PowerPC platforms. Given time, the number of platforms Helix Player supports will outnumber the platforms supported by the other players that Table 3.5 lists.

Chapter 4

Extensibility

Great technology is said to perform well not only the tasks it was designed for, but tasks never envisioned by the original designers as well.[1] An extensible system facilitates the inclusion of new features never dreamed of by the original designers. Just as in research where researchers build on top of each others' works, extensible systems allow developers to do the same and "stand on the shoulders of giants" [7].

This chapter presents the three attributes that make up extensibility, and discusses the features of LecTix 2.0's design and implementation that contribute to each attribute, thereby making it an extensible lecture-multimedia player.

4.1 Attributes of Extensibility

Extensibility is the degree to which any developer can add new features to a system. Extensibility consists of three attributes:

- *Modifiability*: The ease with which a system can be modified by any developer.
- *Modularity*: The degree to which a system is organized into cleanly separated, decoupled parts.
- *Interfaceability*: The degree to which a player can be externally interfaced to, allowing it to be extended without recompilation (i.e. a new release) of the

[1] This idea is not my own, but I have been unable to find its source.

Table 4.1: LecTix 2.0 features that contribute to extensibility

Feature	Modifiability	Modularity	Interfaceability
GPL	✓	—	—
Java	✓	✓	✓
Class Hierarchy	—	✓	✓
Events	—	✓	✓
Lecture Description	—	—	✓

player.

These three attributes compose extensibility. A player that has all three attributes is considered extensible.

4.2 Features of LecTix 2.0 that Contribute to Extensibility

LecTix 2.0 has five features that contribute to extensibility: (1) its open-source license, the GNU General Public License (GPL); (2) the Java language, in which it's implemented; (3) a hierarchy of media classes; (4) a system for media classes to notify each other of events; and (5) a file format for describing lecture-multimedia presentations. Table 4.1 shows which attributes of extensibility each feature contributes to.

As is usually the case with extensible systems, these features build on top of each other to provide several layers of extensibility. Figure 4-1 depicts these layers. The GPL ensures that the source code to the LecTix 2.0 core remains available. Java, a programming language that supports abstraction and garbage collection, facilitates the modification of LecTix 2.0's open source code. Java's object-oriented features of classes, encapsulation, and inheritance enable the organization of a hierarchy of media classes (media types). The media classes that sit at the top of this hierarchy provide a uniform interface for manipulation by the LecTix 2.0 core. A media-event notification system builds on top of the media classes' uniform interface to enable media objects to broadcast *events* without explicit knowledge of the recipients. The

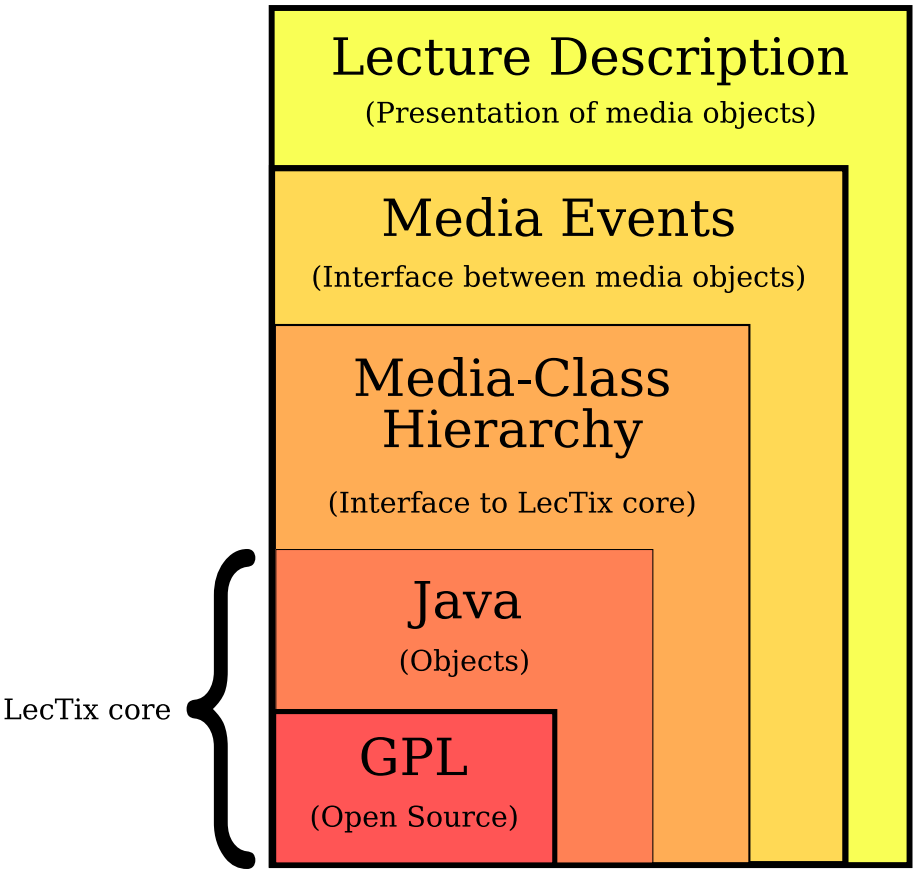

Figure 4-1: Layers of LecTix 2.0's extensibility.

lecture description file pulls it all together, describing the media objects to display and how to synchronize them.

Now I describe each of the five features and how they contribute to LecTix 2.0's extensibility.

4.2.1 The GNU General Public License

LecTix 2.0's open-source license, the GNU General Public License (GPL) [13] contributes to LecTix 2.0's modifiability. As discussed in Section 3.2.1 on page 33, the GPL permits distribution of derived works and ensures that source code remains available. With permission to distribute derived works, and access to the source code, developers can modify LecTix 2.0 to their liking.

4.2.2 The Java Programming Language

LecTix 2.0's language of implementation, Java [17], contributes to LecTix 2.0's modifiability, modularity and interfaceability. These contributions stem from Java's design as an object-oriented programming language [44] with support for abstraction and garbage collection.

Java's support for abstraction and garbage collection facilitates the modification of LecTix 2.0. Support for abstraction enables developers to think of problems at a high level. Garbage collection provides automatic memory management for developers, allowing them to devote more attention to the problem at hand.

Java's support for abstraction can be broken into three parts: classes, encapsulation, and inheritance. Classes and encapsulation contribute to the attribute of modularity. Inheritance contributes to the attribute of interfaceability.

Classes and encapsulation enable LecTix 2.0 to be broken up into separate modules. A *class* is a data type representing a set of variables and methods that can operate on those variables. An instance of a class is called an *object*. *Encapsulation*, also known as *data hiding*, allows objects to hide state from each other, effectively decoupling them. Using classes and encapsulation, a programmer can organize a system into separate, decoupled parts.

Inheritance builds on classes and encapsulation to provide a consistent interface among related classes. Inheritance is the ability for classes to share behavior. For example, if a class `A` inherits from a class `B`, then `A` will have the variables and methods that `B` has. In addition, `A` can add its own variables and methods or even override `B`'s methods. `B` is considered a *subclass* of `A`, and `A` is considered to be a *superclass* of `B`. Objects of class `B` can then substitute for objects of class `A` by a behavioral notion of subtyping [24], and newly introduced external modules can communicate in a general fashion through the interface of class `A`, without knowledge of class `B`.

4.2.3 Media-Class Hierarchy

Taking advantage of inheritance, LecTix 2.0 defines a media-class inheritance hierarchy that contributes to the player's modularity and interfaceability. This inheritance hierarchy enables LecTix 2.0 to reuse code and to provide a uniform interface to media objects, leading to concise, elegant code.

Each of the LecTix 2.0's media classes implements either a specific or generic type of media. For example, a specific media type can implement video or slides. A generic media type, on the other hand, covers a broad range of media types. Two examples of generic media are continuous and discrete media. Continuous media change continuously with time, such as video or audio. Discrete media changes only a discrete number of times during a presentation. PowerPoint-style slides are an example of discrete media.

Object-oriented programs can represent these notions of generic and specific media types. A class representing a specific type is a *concrete* class. A class that represents a generic type is an *abstract* class.

Such classification of media types naturally lends itself to a representation by an inheritance hierarchy of media classes. Figure 4-2 shows the media-class hierarchy implemented by LecTix 2.0. Figure 4-2 labels abstract classes (those representing generic types) with italics, and concrete classes (those representing specific types) with roman type. The most general of media classes, `LTMedia`, sits at the top of the hierarchy. `LTMedia` can represent any media type in LecTix 2.0. Two abstract classes subclass `LTMedia`: `ContinuousMedia` and `DiscreteMedia`. As in our example, `ContinuousMedia` represents media that change continuously with time; `DiscreteMedia` represents media that change only a discrete number of times.

Two concrete classes sit at the bottom of the hierarchy. The first, `JMFMedia`, subclasses `ContinuousMedia`. `JMFMedia` represents media playable by the Java Media Framework (see page 35). The second, `SlideMedia`, subclasses `DiscreteMedia`. `SlideMedia` represents PowerPoint-style slides.

The abstract classes (`LTMedia`, `ContinuousMedia`, and `DiscreteMedia`) provide

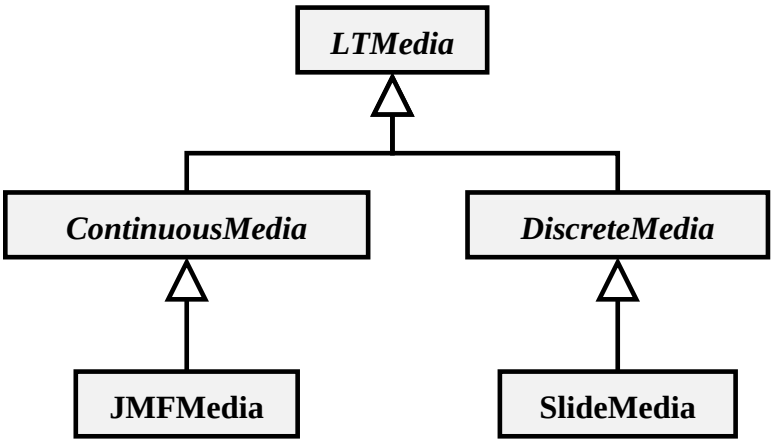

Figure 4-2: LecTix 2.0 media-class hierarchy. Abstract classes are labeled with italics. Concrete classes are labeled with roman type.

a uniform interface for its subclasses to hide behind. Code that manipulates concrete media objects does so in a general fashion, without any explicit references to concrete media classes. For example, the `LTMedia` class declares the method `getVisualComponent()` which returns a `java.awt.Component` object that can display itself visually on the screen. Rather than having to consider whether a media object is video or a sequence of slides, the Graphical User Interface (GUI) of LecTix 2.0 treats the media object as an `LTMedia` object, calls `getVisualComponent()`, and displays the returned object.

The abstract classes facilitate the reuse of code by providing common objects and methods for their subclasses to use. For instance, continuous types of media in LecTix 2.0 keep a running clock that drives the playback of all media. `ContinuousMedia` provides a timer to periodically wake up subclasses so that they can notify other media classes of the time. Discrete types of media maintain a time-ordered list of individual media to present, such as slides. `DiscreteMedia` provides a time-ordered list of general objects and methods to manipulate the list. `SlideMedia` can then specialize the list for slides, and reuse the methods to manipulate the list provided by `SlideMedia`

Table 4.2: LecTix 2.0 media events

Media Event	Description
`RATE_CHANGED`	Change the playback rate
`MEDIA_TIME_CHANGED`	Change the current media time
`STARTED`	Start playback
`STOPPED`	Stop playback
`SKIP_STARTED`	Start rewind or fast-forward
`SKIP_STOPPED`	Stop rewind or fast-forward
`VOLUME_CHANGED`	Change the volume
`PREV_SLIDE`	Show the previous slide
`NEXT_SLIDE`	Show the next slide

4.2.4 Media Events

LecTix 2.0 features a media-event notification system that contributes to LecTix 2.0's modularity and interfaceability. The system provides a mechanism for media objects to broadcast *events* without requiring knowledge of the recipients, thereby decoupling the media objects. The event notification system also serves as a uniform interface for communication among media objects.

For concreteness, Table 4.2 gives an overview of the events used in LecTix 2.0. User interaction with LecTix 2.0's controls results in most sending of the events listed.. In addition, continuous media objects typically send `MEDIA_TIME_CHANGED` events periodically to notify other media of the current running time. The `STOPPED` event may be sent by either the control panel or by a continuous media object when it has reached the end of media time.

Before proceeding to describe the implementation of LecTix 2.0's media-event notification system, we must first define the term *interface*. In object-oriented programming, an *interface* is a label for a set of method *declarations*. Method declarations declare the types of objects that the methods take as arguments and the type of object each method returns. Unlike a method *definition*, an interface does not provide implementations for its methods.

Interfaces are important in languages that do not support *multiple inheritance*. Multiple inheritance is the ability to inherit from more than one superclass. To prevent confusing situations where more than one superclass defines methods with

identical signatures, some languages such as Java and Smalltalk [16] do not allow for multiple inheritance. Instead, Java provides interfaces, for which a class can implement any number of. Having multiple interfaces does not pose the same problem that having multiple inheritance does, because interfaces only declare methods, not define them.

Through the implementation of multiple interfaces, media objects can take on different roles in LecTix 2.0. Media objects can (1) display themselves, (2) send events, (3) receive events. Figure 4-3 shows the hierarchy of classes and interfaces to support these roles.

Starting from the bottom right of the figure, `LTMedia`—the parent of all media classes (see Figure 4-2 on page 48 for the complete media class hierarchy)—provides the `getVisualComponent()` method for displaying media. Media classes usually override this method, because `LTMedia`'s default implementation returns a `null` object. Nonetheless, `LTMedia`'s definition provides a uniform interface across all media objects.

To the left of `LTMedia` is the `ControlPanel` class, responsible for the controls the user interacts with. It inherits from `JPanel`, a class from the Java Swing toolkit.

Above `ControlPanel` and `LTMedia` are three interfaces and one class that make up LecTix 2.0's event system. The top two interfaces in the events system, `LTMediaListener` and `LTMediaEventSource`, model after interfaces suggested by Geary [14, pages 300–309]. `LTMediaListener` serves as a uniform interface to objects that listen to media events; it declares methods for receiving each media-event type. `LTMediaListener` inherits from `EventListener`, an interface from the `java.util` package that declares no methods but exists solely to tag various kinds of event-listener classes.[2] Positioned to the right of the `LTMediaListener` class, `LTMediaEventSource` serves as a uniform interface to objects that send events; it declares methods for adding and removing listeners from its notification list.

In addition to Geary's suggested event-support classes, LecTix 2.0 adds the `LTMediaSocialite` interface and `LTMediaEventBroker` class to complete its media-

[2]Java uses events for all types of classes, particularly in the Java Swing toolkit.

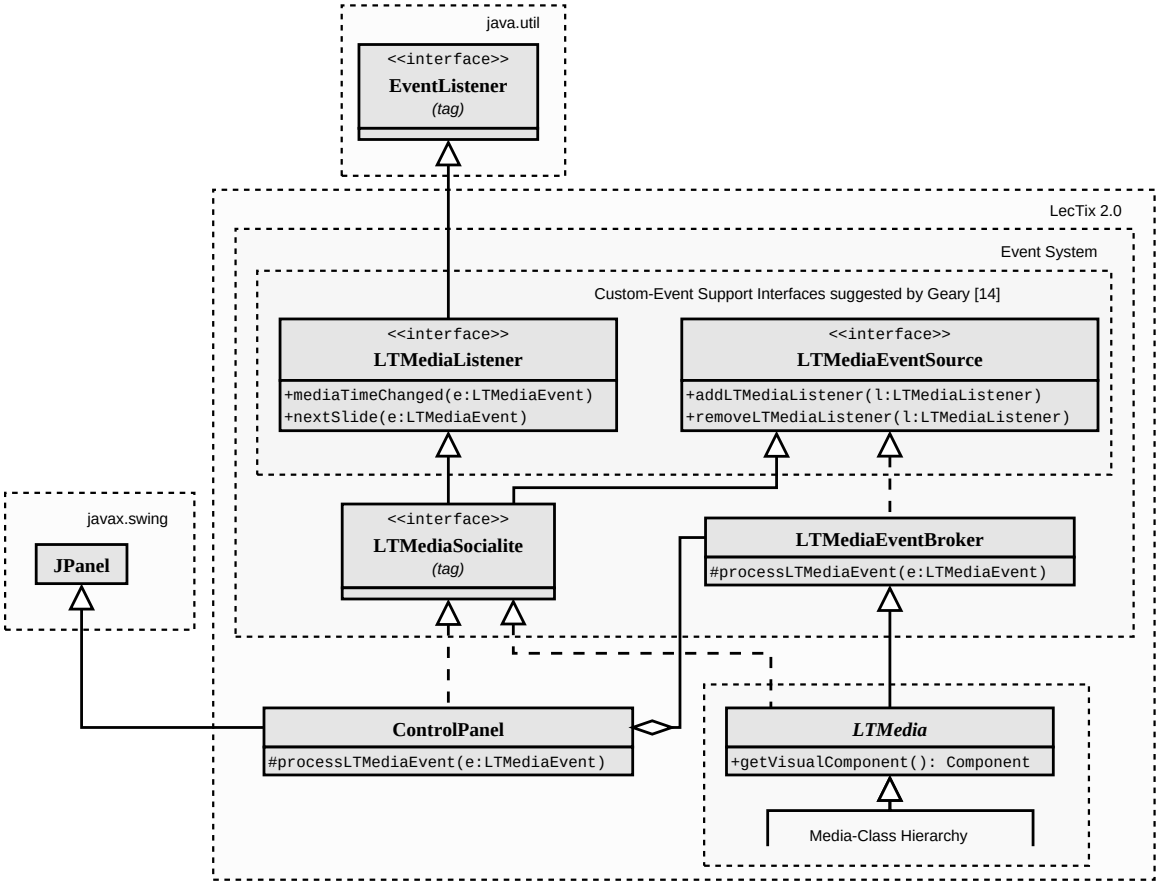

Figure 4-3: Unified Modeling Language (UML) [2] representation of event-support classes and interfaces. Each solid box represents a class or interface. Interfaces and concrete classes are labeled in bold type, with interfaces distinguished by the «interface» stereotype. The abstract class `LTMedia` is labeled in bold italics. Solid lines ending with a triangular arrowhead indicate inheritance. Dashed lines ending with a triangular arrowhead indicate implementation of an interface. The solid line that begins with a diamond-shaped head indicates that `ControlPanel` has an `LTMediaEventBroker` object as a member variable. With the exception of `JPanel`, each class or interface lists a sampling of its method definitions or declarations, respectively, in the bottom half of its box. The note *(tag)* indicates an interface with no method declarations. The symbol + precedes *public* methods. The symbol # precedes *protected* methods.

event system. Sitting directly under `LTMediaListener`, the `LTMediaSocialite` interface serves to tag classes that implement both `LTMediaListener` and `LTMediaEventSource` classes. To the right of the `LTMediaSocialite` interface sits the `LTMediaEventBroker` class; it provides a default implementation of an `LTMediaEventSource` and maintains the event notification list of listeners. It also defines the *protected* (only accessible to it and its subclasses) `processLTMediaEvent()` method for sending events.

So that all media classes can send and receive messages, `LTMedia` implements the `LTMediaSocialite` interface. `LTMedia` inherits from `LTMediaEventBroker`, gaining an implementation for maintaining notification lists and sending messages. To receive messages, `LTMedia` implements the `LTMediaListener` interface with default definitions of the event-receiving methods. The default definitions take no action; media classes override them when they wish to receive an event.

The `ControlPanel` class also implements the `LTMediaSocialite` interface. Because it already inherits from the `JPanel` class, however, `ControlPanel` cannot also inherit from `LTMediaEventBroker` (see discussion of multiple inheritance on page 50). Instead, `ControlPanel` takes `LTMediaEventBroker` as a member variable. `ControlPanel` defines its implementation of `processLTMediaEvent` and `LTMediaEventSource`'s event-notification-list maintenance methods to call those of `LTMediaEventBroker`.

Media-Event Example

To get a better idea of how media events work in practice, let's consider an example where a student advances a slide. Figure 4-4 shows the four steps LecTix 2.0 takes to advance a slide.

First, the user clicks on the next slide button (Figure 4-4(a)). `ControlPanel` then instantiates an object representing the event `NEXT_SLIDE` and calls `processLTMediaEvent()` with the event object as its argument.

Second, `processLTMediaEvent()` calls `nextSlide()` on all of `ControlPanel`'s media-event listeners (Figure 4-4(b)). In this case, the media-event listeners are

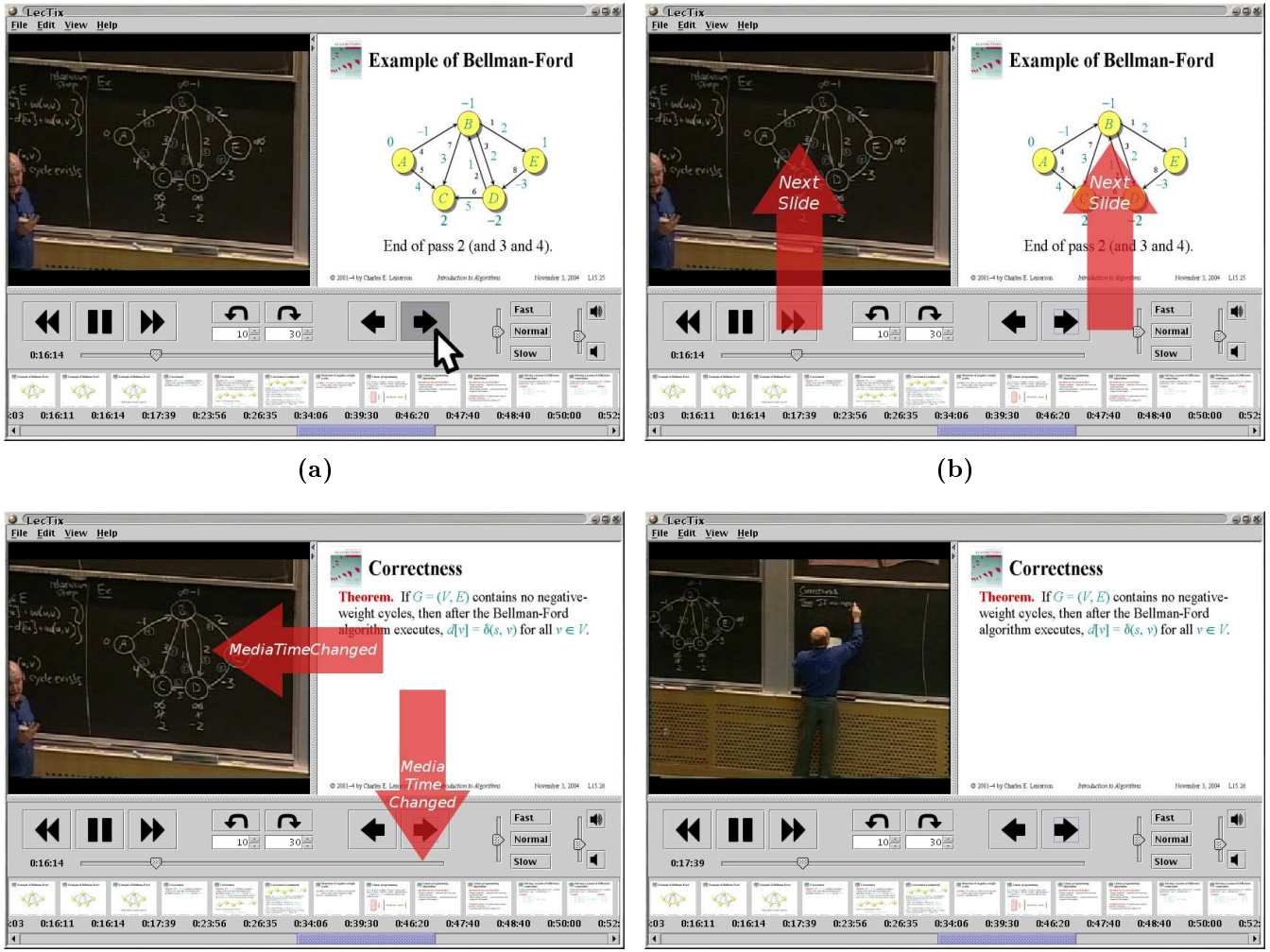

Figure 4-4: Advancing a slide. **(a)** Student clicks the *Next Slide* button. **(b)** `ControlPanel` sends *NextSlide* events to `JMFMedia` (video) and `SlideMedia`. **(c)** `SlideMedia` advances to the next slide and sends *MediaTimeChanged* events to `JMFMedia` and `ControlPanel`. **(d)** `JMFMedia` advances to the new time. `ControlPanel` updates its seek time and slider.

`JMFMedia` (video and audio) and `SlideMedia` (slides). Figure 4-4(b) depicts, with red arrows, `ControlPanel` sending `NEXT_SLIDE` events to `JMFMedia` and `SlideMedia`.

Third, `SlideMedia` advances the slide (Figure 4-4(c)). To maintain synchronization, `SlideMedia` broadcasts a `MEDIA_TIME_CHANGED` event. As shown by red arrows in Figure 4-4(c), `SlideMedia`'s listeners are `JMFMedia` and `ControlPanel`.

Finally, `JMFMedia` and `ControlPanel` receive the `MEDIA_TIME_CHANGED` events (Figure 4-4(d)). `JMFMedia` advances the video to the new time, and `ControlPanel` adjusts the seek slider and time display. The slide advance is now complete, and the lecture presentation remains synchronized.

4.2.5 Lecture Description

LecTix 2.0's lecture description format contributes to interfaceability by providing a textual format for describing lecture-multimedia presentations. The lecture description serves as the interface for lecture-multimedia producers—human users or software—to target. The lecture description interfaces to and builds on top of the extensibility of the media-event system, media-class hierarchy, and Java programming language.

A lecture description file specifies:

- The filenames of individual media files, such as video and image files.
- The media time at which each media file is to be displayed (for synchronization).
- The concrete media classes that should be instantiated to display the media files.

To see what a lecture description looks like, look at the sample one shown in Figure 4-5. The lecture is encoded in the Extensible Markup Language (XML) [3]. In general, XML consists of nested *elements*, each with zero or more *attributes*. For example, the root element of the lecture description shown in Figure 4-5 is `lecture`. It has the attribute `version` which specifies the earliest version of LecTix that can understand this lecture description.

```
<?xml version="1.0"?>

<lecture version="2.0">
  <media type="JMF">
    <file name="lecture15.mpg"/>
  </media>
  <media type="Slide">
    <file time="00:00:00" name="slides/Slide001.png"/>
    <file time="00:01:24" name="slides/Slide002.png"/>
    <file time="00:02:59" name="slides/Slide003.png"/>
                                    ⋮

    <file time="01:16:06" name="slides/Slide045.png"/>
  </media>
</lecture>
```

Figure 4-5: A lecture description file

Two `media` elements nest inside the `lecture` element. Each of the two `media` elements specify a `name` attribute. LecTix 2.0 will append the suffix `-Media` to get the name of the media class to instantiate for each `media` element. In this case, LecTix 2.0 will instantiate a `JMFMedia` object and a `SlideMedia` object.

Nested inside the `media` elements are `file` elements. Each `file` element specifies the `name` of a media file, and optionally, the `time` at which to display it. If no `time` attribute is given, LecTix 2.0 assumes a default time of 0. In this case, the video file, `lecture15.mpg`, will start at 0, the beginning of media time.

The simple encoding of a lecture description in a simple XML text file contributes to LecTix 2.0's interfaceability. Both humans and software can easily modify a lecture-multimedia presentation. Of course, if a developer wishes to add new media types or change the internals of LecTix 2.0, he or she is free to do so. LecTix 2.0 makes this freedom possible by offering five layers of features that contribute to its extensiblity: the GNU GPL, the Java programming language, a media-class hierarchy, media events, and a lecture description file format.

Chapter 5

Related Work

This chapter introduces seven lecture-multimedia players, describes their user interfaces and features, and compares them and LecTix 2.0 in terms of usability, availability, and extensibility.

I only consider players here that can present at least one additional media stream besides audio and video. All the players reviewed here fulfill this requirement by presenting a scheduled stream of static images—typically PowerPoint-style slides or snapshots of the blackboard.[1]

The players reviewed are:

1. Columbia Video Network
2. IIT Online
3. Microsoft Producer
4. Singapore-MIT Alliance
5. Stanford Online
6. UNITE (University of Minnesota)
7. LecTix 1.3

[1] One player I do not review, the .NET Show [30], presents audio, video, and a transcript.

A screenshot of each player is given along with a description of its interface and features.

5.1 Columbia Video Network

The Columbia Video Network (CVN) [8] is Columbia University's distance education program, offering university credit and degree programs online.

Figure 5-1 depicts the player for the Columbia Video Network playing sample lecture available online. As is common in many lecture-multimedia players, the player is embedded in a web page. Shown in the upper left corner, a Windows Media plug-in plays a streaming lecture video. In the upper right, a high-quality close-up of the blackboard displays writing otherwise hidden by the lecturer in the video. The bottom panel contains an index of various points in time of the lecture. Unfortunately, the names given for the index entries, such as *Image 59*, probably do not help the student much.

Directly under the VCR-style controls presented by the Windows Media plug-in are controls for adding custom index entries. To add an entry, the student clicks on the *Time* button when the presentation is at the desired point in time. The player then displays the time in the text box to the right of the *Time* button. Next, the user enters a name for the entry, and clicks on *Add marker*. By delaying entering the entry's name until after capturing the time, the user obtains an accurate timing without much advance notice.

5.2 IIT Online

IIT Online [19] is the Illinois Institute of Technology's distance education program, offering university credit and degree programs online.

Figure 5-2 shows the player for IIT Online [19]. The player consists of Synchronized Multimedia Integration Language (SMIL) [1] presentation inside RealPlayer. The upper-left hand portion of the player shows a video of the lecturer writing notes

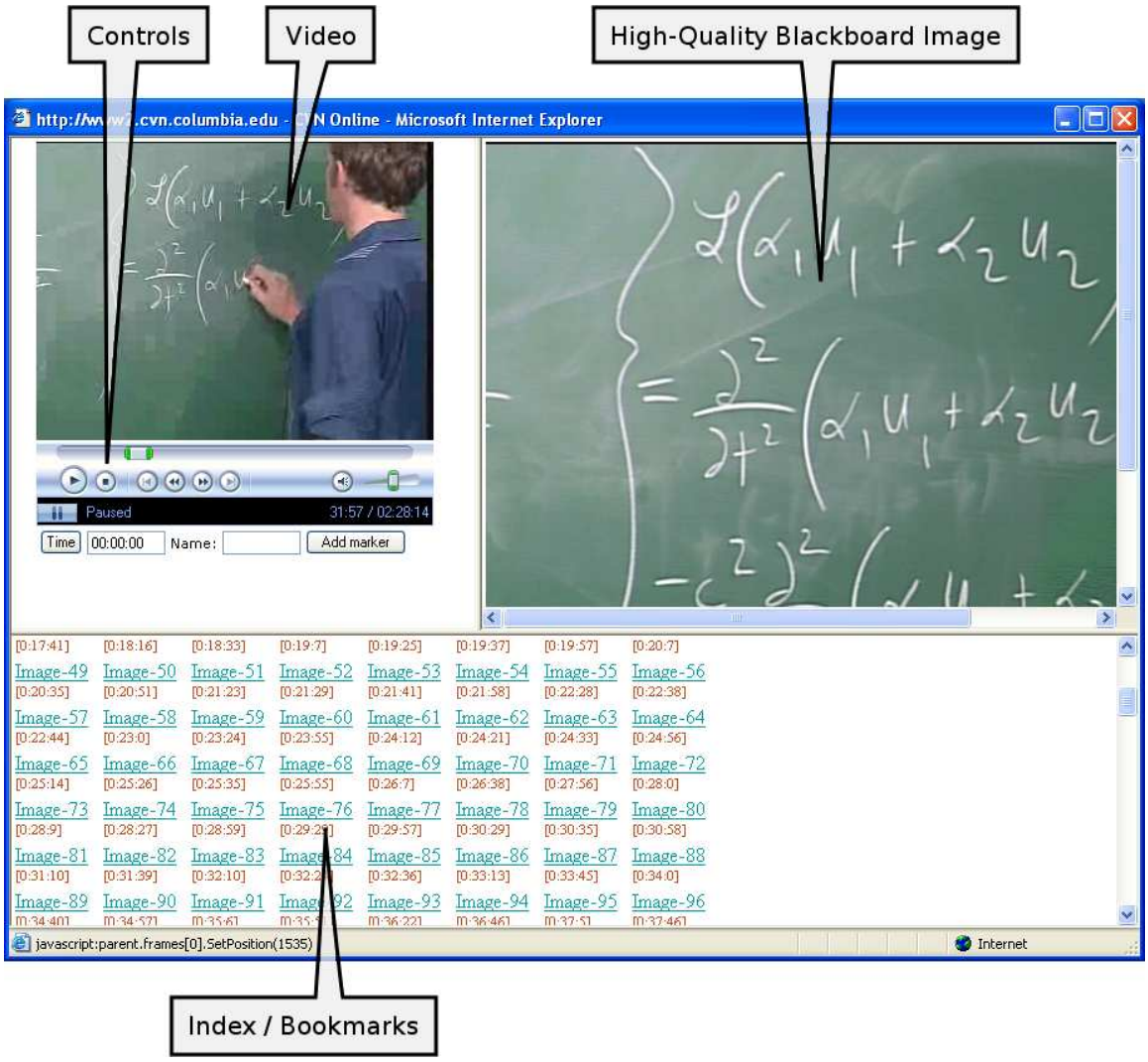

Figure 5-1: Columbia Video Network

on the paper. The right half of the player zooms in on the notes and previews what the lecturer will write. Directly under the video sits an index. Contrary to the index found in the CVN player (Section 5.1), this player's index has meaningful entries that help students find the part of the lecture that interests them.

The controls for the IIT Online player sit along the bottom of the player. These controls consist of typical VCR-style controls. As with the SMA player (Section 5.4), the previous-clip and next-clip buttons (sitting to the right of the stop button) appear useless in the context of a single lecture-multimedia presentation. If held down, however, the buttons rewind or fast-forward the presentation.

5.3 Microsoft Producer

Microsoft Producer [29] is an add-on to Microsoft's PowerPoint software. From an encoded video and PowerPoint slides, Microsoft Producer produces multimedia presentations playable inside Microsoft's Internet Explorer browser. The presentations integrate the video, slides, and an index. While not exclusively for lecture-multimedia presentations, the player shares features commonly found in lecture-multimedia players.

Figure 5-3 depicts a sample multimedia presentation made by Microsoft Producer. The layout is similar to that of IIT Online: video sits in the upper left, an index appears below it, and a slide displays in the right half of the screen. Unlike IIT Online, however, Microsoft Producer's controls sit between the video and index. Also, despite the adequate empty space next to the time display, the player does not provide a seek slider. The player compensates, however, by featuring buttons for skipping back and forward 10 seconds.

5.4 Singapore-MIT Alliance

The Singapore-MIT Alliance (SMA) [42] is a joint educational and research collaboration among three universities: the National University of Singapore (NUS), the

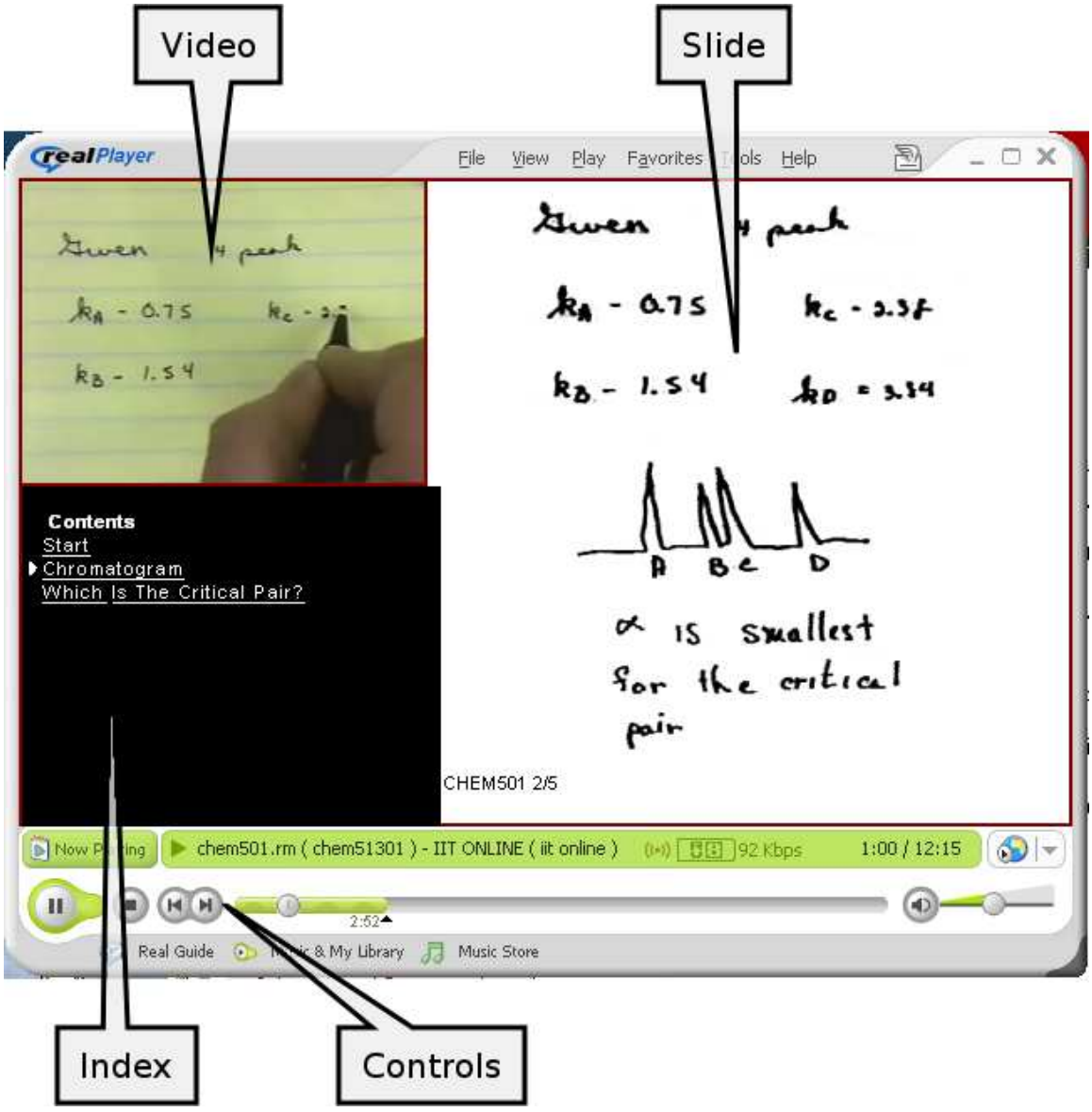

Figure 5-2: IIT Online

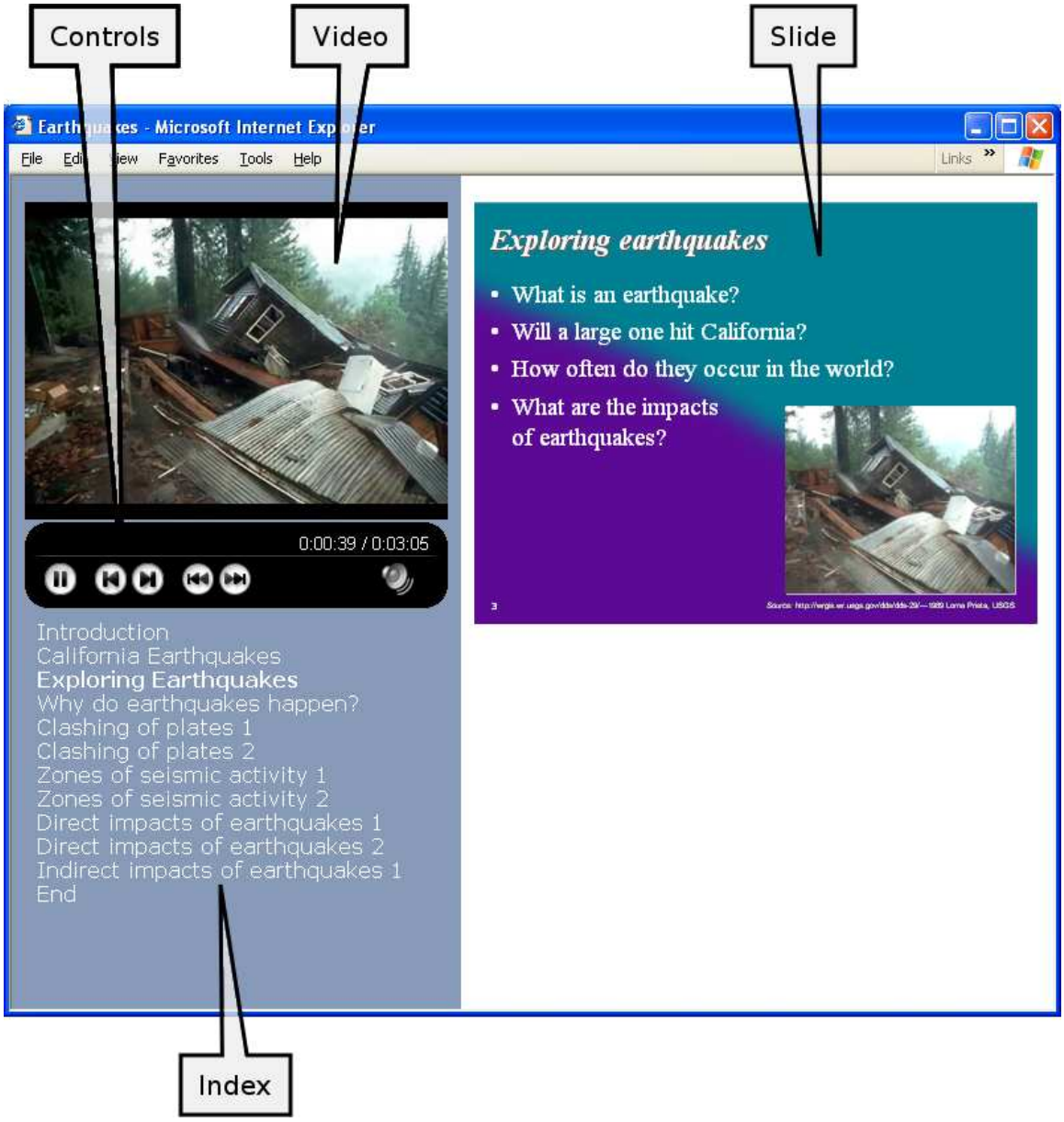

Figure 5-3: Microsoft Producer

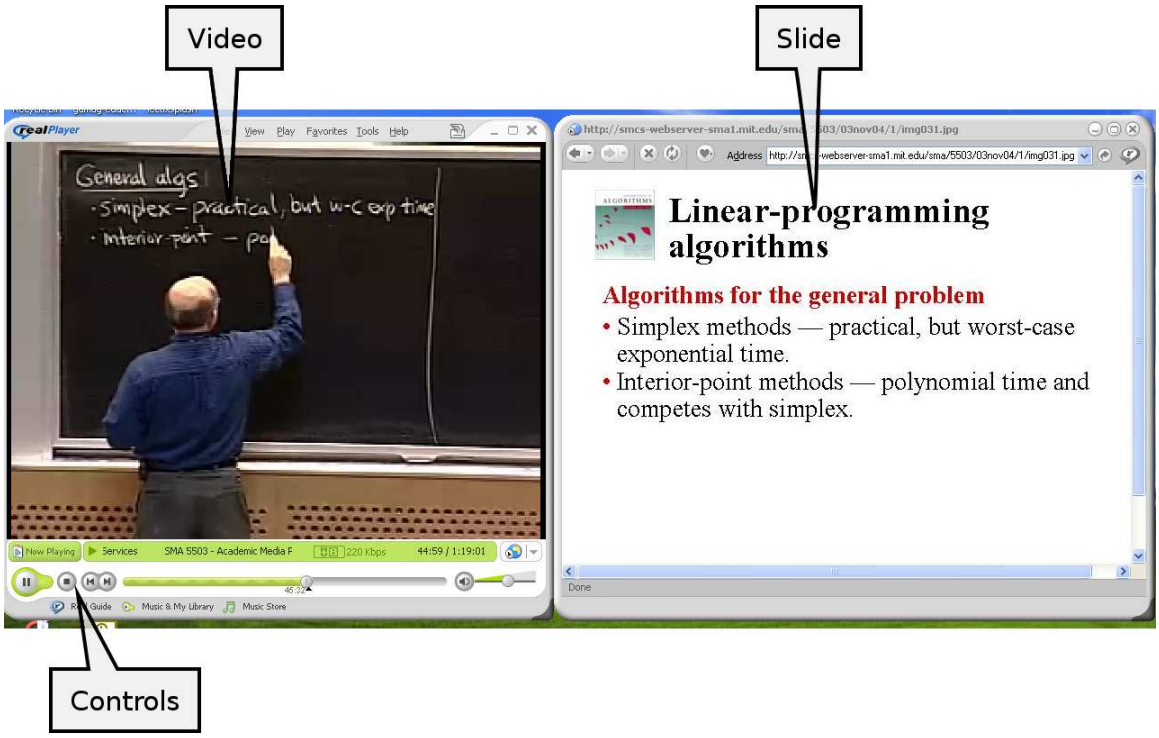

Figure 5-4: Singapore-MIT Alliance

Nanyang Technological University (NTU), and MIT. SMA offers classes in Singapore at NUS or NTU, as well as in Cambridge, Massachusetts at MIT.

Figure 5-4 shows a screenshot of the SMA player. Similar to IIT Online (Section 5.2), the SMA player is based on RealPlayer. The SMA player differs from the IIT Online player, however, in that it does not use SMIL, and in that its interface consists of two windows: a main RealPlayer window on the left, and a window on the right that displays slides. The main RealPlayer window devotes most of its space to the video. Underneath the video are the same controls found in the IIT Online player, including the previous-clip and next-clip buttons that also rewind and fast-forward. The slide window on the right features controls along the top for browsing the history of slides already seen—unlike other lecture-multimedia players which have previous- and next-slide buttons for browsing slides in presentation order.

5.5 Stanford Online

Stanford Online offers Stanford University graduate programs and courses over the Internet.

Figure 5-5 depicts the Stanford Online player. Like the CVN player, the Stanford Online delivers its player through a web page. A Windows Media plug-in sits on the left side of the browser, and a slide displays in the center. Two sets of controls appear in this player. A set of VCR-style controls sits below the video as part of the Windows Media plug-in. Another set of controls for navigating slides sit below the slide.

5.6 University of Minnesota UNITE

Figure 5-6 shows the player offered by the University of Minnesota's UNITE program [46]. The player positions the multimedia like the rest of the players: video on the left and static image on the right. At the point in time shown in Figure 5-6, however, the player has swapped the types of content typically presented by the video and image streams. The image shows a still-frame shot of the whiteboard, and the video shows a PowerPoint-style slide. This technique allows a user to still see the slide while the player presents the whiteboard as a high-quality static image.

Similar to Microsoft Producer and Stanford Online, this player features slide-navigation controls. A user can navigate to the first, previous, next, or last slide. In addition, the UNITE player features a scrollable strip of slide thumbnails (miniature renderings of the slides). Clicking on a thumbnails brings up the full slide. Unfortunately, the video does not synchronize with the new slide.

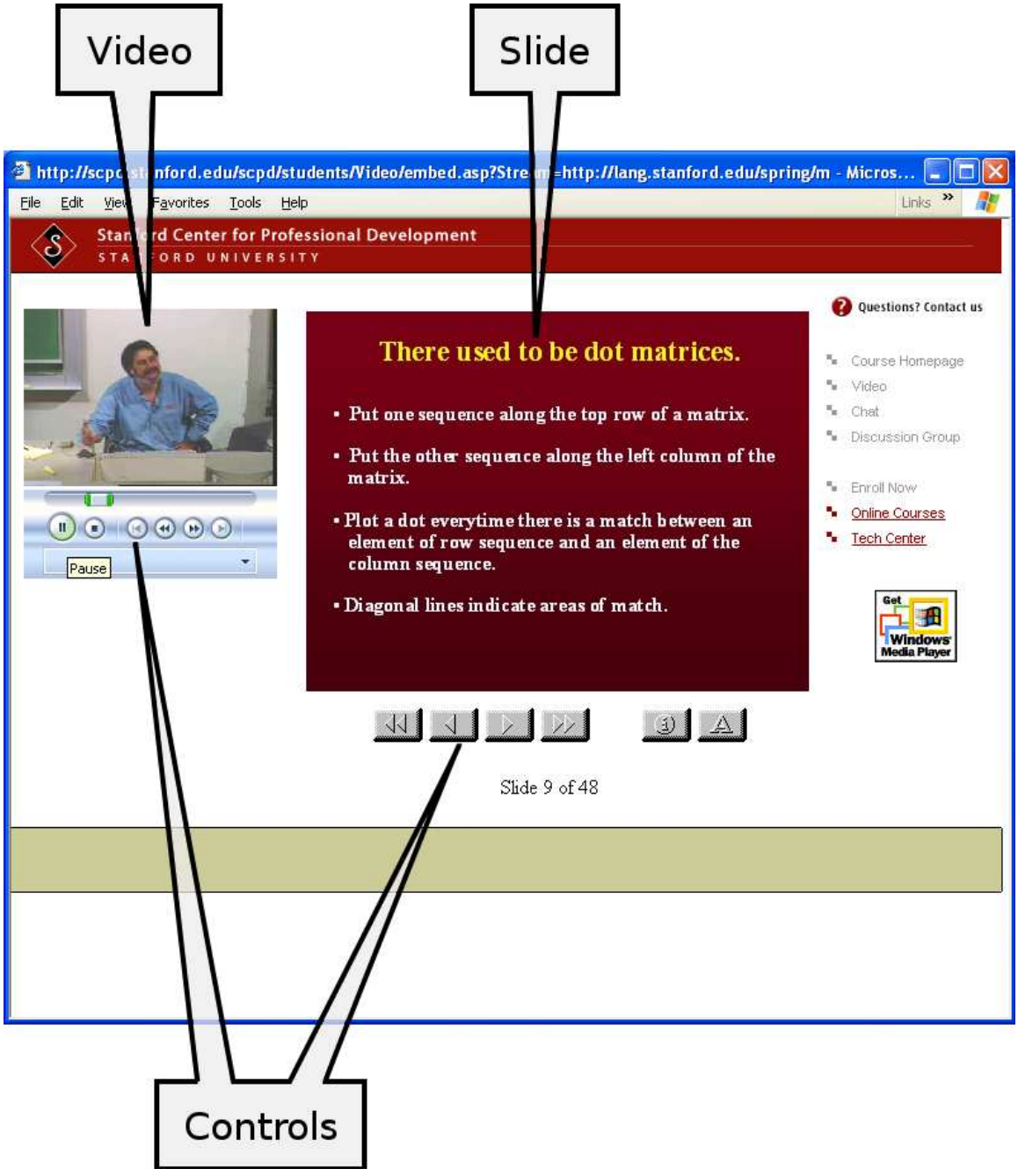

Figure 5-5: Stanford Online

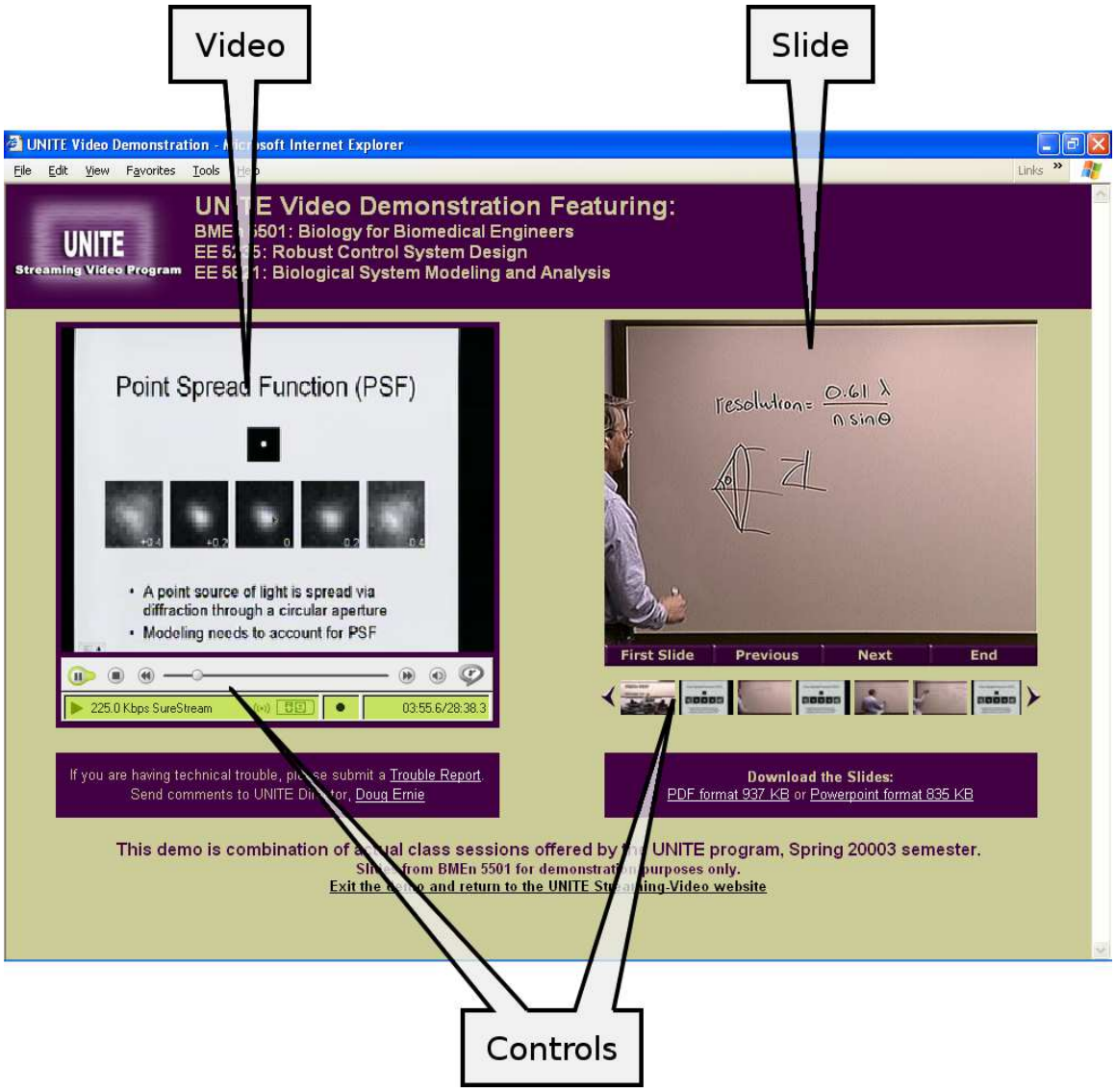

Figure 5-6: UNITE

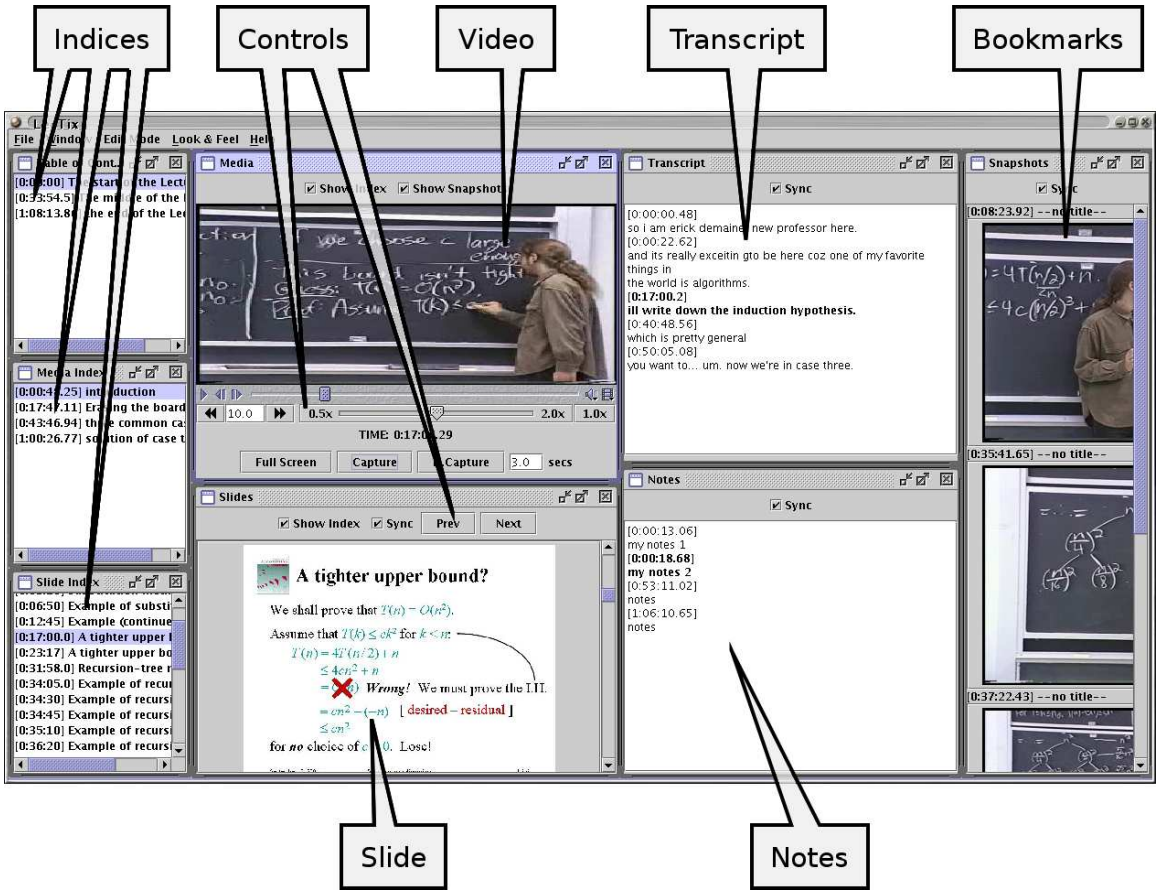

Figure 5-7: LecTix 1.3

5.7 LecTix 1.3

LecTix 1.3 is the predecessor to LecTix 2.0.[2] Shown in Figure 5-7, LecTix 1.3 supports many features not available in LecTix 2.0 such as multiple text indices, transcript, user-editable notes, and bookmarks.

Along the left side of the player lie multiple text indices called the table of contents, media index, and slide index. Selecting an entry in any of the indices jumps the presentation to the point in time relevant to the entry.

To the right of the controls lie the video and slides. A variety of controls sit below the video and above the slides. Besides the usual play, pause, and seek slider controls, LecTix 1.3 has a skip-back, skip-forward, variable-speed slider, full-screen

[2]LecTix has been known by several names. Here they are in time order from earliest to latest: Lecture Viewer [18], EVES [27], LecTix 1.3, and LecTix 2.0.

button, and a capture button for setting bookmarks. LecTix 1.3 calls its bookmarks snapshots because they not only mark a label and time, but snapshots of the video as well.

To the right of the video and slides lie the transcript and user-editable notes. Both appear similar, but have different uses. The transcript serves to help students who have difficulty understanding the lecturer's speech. In the notes area, the student can take down notes that are synchronized to the presentation. The notes can be considered another form of bookmarks.

To the very right lies a scrollable pane of snapshots taken by the student. This feature provides a way for the student to keep parts of the blackboard on display after the video has moved elsewhere.

5.8 Comparison of Players to LecTix 2.0

This section compares the seven players just described to LecTix 2.0. I compare the players in terms of the properties of usability, availability, and extensibility. With the possible exception of LecTix 2.0's incompatibility with contemporary state-of-the-art video codecs and its affect on availability, I show that LecTix 2.0 fares better than the seven other players in all three properties.

Usability Comparison

LecTix 2.0 contains more features that contribute to usability than the other seven players. Of the features I consider here, LecTix 2.0 lacks only bookmarks.

To better compare the features across all eight players, we must first expand the list of features found in Table 2.1 (page 20) that we identified contributed to LecTix 2.0's usability. Table 5.1 lists the expanded set of features and which of the five attributes of usability they contribute to. The navigation feature from Table 2.1 has been broken up into bookmarks, index, previous and next slide, and seek slider. Table 5.1 also adds a new feature, tooltips.

Table 5.8 shows which of the features from the expanded list that each of the eight

Table 5.1: Expanded list of features that contribute to usability

Feature	Learnability	Efficiency	Memorability	Few Errors	Satisfaction
Controls					
Bookmarks	—	✓	—	—	✓
Index	—	✓	—	—	✓
Prev/Next Slide	—	✓	—	—	✓
Seek Slider	—	✓	—	—	✓
Variable Speed	—	✓	—	✓	✓
Controls' Properties					
Large	—	✓	—	✓	—
Real World	✓	—	✓	✓	—
Tooltips	✓	—	✓	✓	—
Visible	✓	✓	✓	—	—
Synchronization	—	✓	—	✓	✓

players implement. LecTix 2.0 comes out ahead with 8 of the 9 features implemented. Columbia Video Network and LecTix 1.3 tie for second place with 6 of 9 features implemented.

Availability Comparison

In terms of availability, LecTix 1.3 and LecTix 2.0 compare favorably to the other six players due to their open-source code base and high portability. The other six players, however, can play contemporary formats such as Windows Media (ASF) and RealMedia.

Table 5.8 shows which of the features that affect availability each player implements. While not shown here, LecTix 1.3 and LecTix 2.0 can run on more platforms than just the ones listed in Table 5.8. See Section 3.2.2 on page 33 for a discussion on the high portability of the Java language.

Extensibility Comparison

In terms of the three attributes of extensibility—modifiability, modularity, and interfaceability—LecTix 2.0 rates better than any of the other seven players by contributing to all three attributes (see Chapter 4).

LecTix 1.3 is modifiable (open-source), but lacks somewhat in interfaceability,

Table 5.2: Reviewed players' features that affect usability

Feature	Columbia Video Network	IIT Online	Microsoft Producer	Singapore-MIT Alliance	Stanford Online	UNITE	LecTix 1.3	LecTix 2.0
Controls								
Bookmarks	✓	×	×	×	×	×	✓	×
Index	✓[a]	✓	✓	×	×[b]	✓	✓	✓
Prev/Next Slide	×	×	✓	×[c]	✓[d]	✓[d]	✓	✓
Seek Slider	✓	✓	×	✓	✓	✓	✓	✓
Variable Speed	✓	×	✓	×	✓	×	✓	✓
Controls' Properties								
Large	×	×	×	×	×	×	×	✓
Real World	✓	✓	×	×	✓	✓	×	✓
Tooltips	✓	✓	✓	✓	✓	×	×	✓
Visible[e]	×	×	×	×	×	✓	✓	✓
Synchronization	Segment	Segment	Segment	Segment	Trigger	Manual	Segment	Segment

[a] All of the sample lectures available on CVN's website had useless index entries of the form *Image n* where n is a positive integer.
[b] Stanford Online provides an index, but the user cannot click on the entries, and scrolling is broken.
[c] SMA provides previous- and next-slide buttons, but they only serve to browse through the history of the lecture as it has already been played.
[d] Not synchronized.
[e] A player fails here if it has hidden controls other than the volume slider.

Table 5.3: Reviewed players' availability

Feature	Columbia Video Network	IIT Online	Microsoft Producer	Singapore-MIT Alliance	Stanford Online	UNITE	LecTix 1.3	LecTix 2.0
Open Source	×	×	×	×	×	×	✓	✓
Video Format[a]	WM	Real	WM	Real	WM	Real	MPEG-1[b]	MPEG-1[b]
Streaming/**L**ocal	S	S	S	S	S	S	L	L
Platform								
Windows[c]	✓	✓	✓	✓	✓	✓	✓	✓
Mac OS X[d]	[e]	✓	×	✓	[e]	✓[f]	✓[g]	✓[g]
Linux[c]	×	✓[h]	×	✓	×	✓[f]	✓	✓

[a] WM stands for Windows Media. Real stands for RealMedia.
[b] Both LecTix versions can also play Cinepak and H.263 codecs.
[c] Intel x86.
[d] PowerPC.
[e] Audio and video only.
[f] Slide index not available.
[g] Cannot play MPEG-1, but can play Cinepak and H.263 codecs.
[h] Using the slide index crashes RealPlayer.

and especially in modularity. Media objects in LecTix 1.3 make explicit calls to each other. To add a new media type requires careful consideration of many lines of code. LecTix 1.3 does, however, have a lecture description format which contributes somewhat to its interfaceability.

While Windows Media has an API available for it, the players based on it—such as CVN, Microsoft Producer, and Stanford Online—do not reexport the interface. Furthermore, Windows Media scores poorly on modifiability because of its proprietary code.

The rest of players—IIT Online, SMA, and UNITE—which are based on RealPlayer, are not extensible. All have proprietary code, and no API is available. Even if they may be designed in modular fashion, external developers cannot modify or interface to the code.

Chapter 6

LecTix 1.3 Case Study

In the fall of 2004, the course *6.046: Introduction to Algorithms*, taught at the Massachusetts Institute of Technology, made LecTix 1.3 (see Section 5.7, page 67) available for student use. This chapter looks at the methods used for producing the lecture multimedia for the course, and the students' overall reaction to LecTix 1.3.

6.1 Production of Lecture Multimedia

To try out LecTix 1.3, it was offered as an experimental player in addition to the Singapore-MIT Alliance (SMA) player (see Section 5.4 on page 60). Production costs were kept down by only preparing slides, a slide index, and a lecture video for each lecture. A table of contents, media index, and transcript were not included.

The production of the lecture-multimedia for *6.046: Introduction to Algorithms* involved three parties: the lecturer, teaching assistant (TA), and MIT's Academic Media Production Services (AMPS). The lecturer took responsibility for preparing the lecture slides and giving the lecture. The TA was responsible for capturing slide timings, and encoding the lecture video to MPEG-1, and fine-tuning the timings to produce a lecture multimedia presentation. Before giving the lecture video to the TA, AMPS took responsibility for the lecture's recording.

Besides recording, AMPS also encoded and hosted lecture media for the SMA player. Their lecture media consisted of two media streams: lecture video and Pow-

erPoint slides.

Due to similar content, some of the production work involved in producing lecture multimedia for the SMA player and LecTix 1.3 was shared. Both sets of lecture multimedia required the production of slides, slide timings, and lecture video.

Figure 6-1 details the production work-flow for LecTix 1.3 lecture multimedia for *6.046: Introduction to Algorithms*. The three participants—lecturer, TA, and AMPS—sit across the top of the diagram.

The work-flow consists of five stages. The first two stages of the work-flow contain the actions shared between the production of SMA and LecTix multimedia. The last three stages contain actions only for the production of LecTix multimedia. Furthermore, the TA is the only participant to participate in these last stages.

Stage 1 The lecturer prepares the slides. Once finished, the lecturer gives a copy of the slides to the TA.

Stage 2 The lecturer gives the lecture. During the lecture, the TA records the times at which the slides should be shown during lecture multimedia playback, otherwise known as the slide timings. Meanwhile, AMPS records the lecture.

Stage 3 The TA encodes AMPS's recording of the lecture into MPEG-1 video. While a computer encodes the lecture, the TA adds slide titles to the slide timings to produce a slide index.

Stage 4 The TA fine-tunes the lecture timings to the lecture video. If the original slide timings are accurate, only a small offset to all the timings is necessary to align each one's start times.

Stage 5 The TA posts the lecture multimedia to the web.

After the TA posts the lecture to the web, students can then download and watch it.

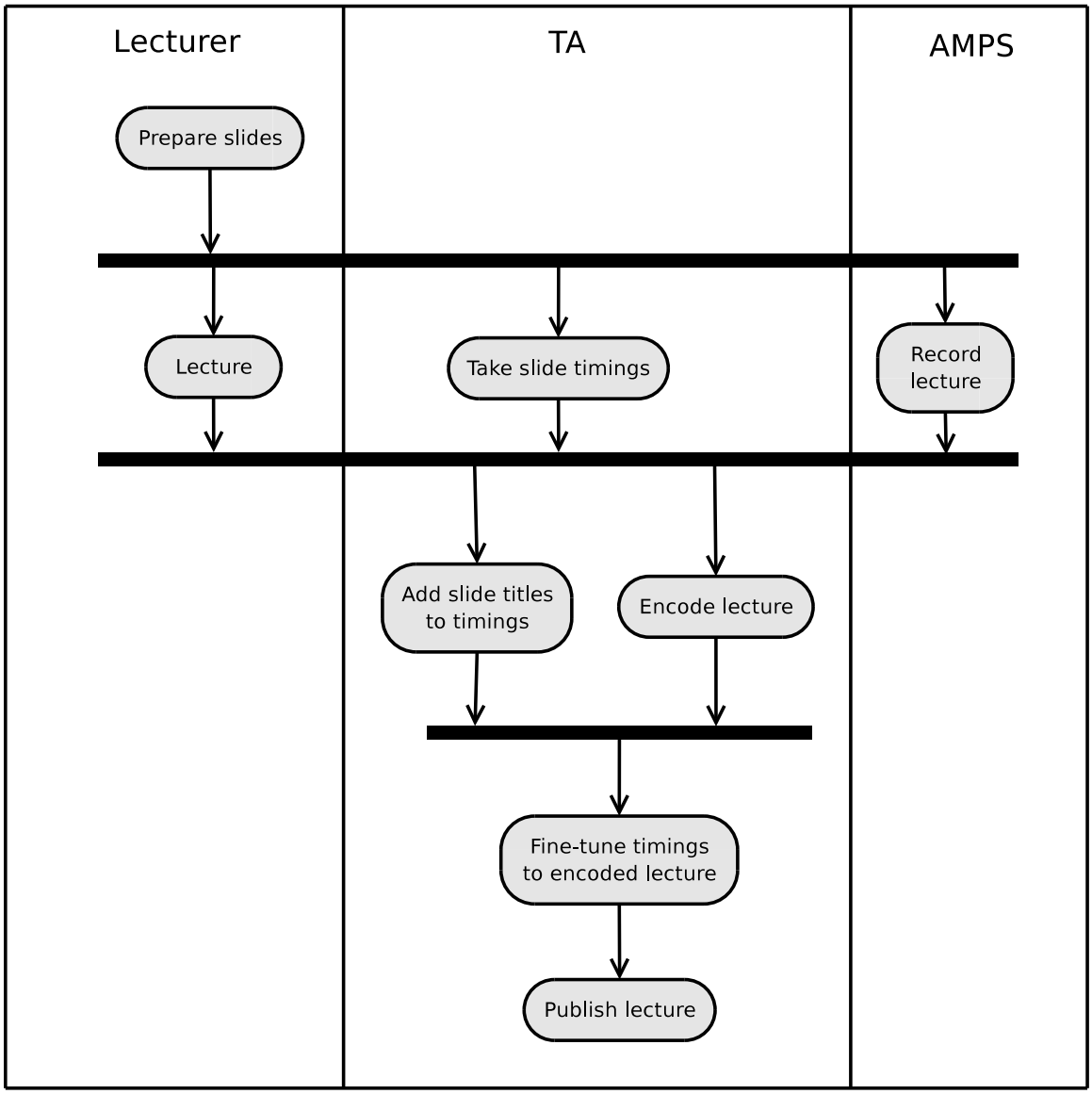

Figure 6-1: Production work-flow for *6.046: Introduction to Algorithms*. Labeled along the top are the participants in the work-flow. Actions they are responsible for lie in shaded ovals under each participant. Arrows show the direction of the work-flow. Thick horizontal bars represent synchronization points. Progress cannot proceed past a synchronization point until all actions leading up to it finish.

Table 6.1: Downloads for LecTix 1.3 and lecture-multimedia

	No. downloads	Average downloads per lecture
LecTix 1.3	156	—
Full Lecture	610	33.9
Just Video	2,366	131.4

6.2 Student Reaction to LecTix 1.3

Overall, the student reaction to LecTix 1.3 was fair. Table 6.1 shows that LecTix 1.3 was downloaded 156 times (class size was 114). Assuming most of the downloads were from students of *6.046*, this number implies that many ,if not most, of the students tried the player.

The remaining two lines of Table 6.1, however, tell more. Students downloaded full lecture-multimedia (video plus slides) 610 times compared to just the video being downloaded 2,366 times. I hypothesize that the students found the convenience of having the lecture file locally (an attribute of availability) to trump the higher quality video streamed by the SMA player.

Informal feedback from some students indicate that I am somewhat correct. But other feedback says that they did not like LecTix 1.3. The main reason they gave was that the video was too small, and that LecTix 1.3 wasted a lot of screen space for features that our production didn't support.

With this knowledge, I decided that LecTix 1.3 suffered a bit from feature overload, and that it would be best to redesign it from scratch. And that is how LecTix 2.0 was born.

Chapter 7

Conclusion

This chapter concludes with comments on the contributions of LecTix 2.0 and with some ideas for future work.

7.1 LecTix 2.0 Contributions

LecTix 2.0 aims to be a usable, available, and extensible lecture-multimedia player. In many aspects, it succeeds. I have shown how its features contribute to the attributes that compose usability, how its features contribute to all but the compatibility attribute of availability, and how its features contribute to extensibility.

While it's great that the features of LecTix 2.0 contributes to something, what exactly does LecTix 2.0 contribute to the world?

In that regard, LecTix 2.0 contributes a free, open-source lecture-multimedia player that students find easy to use and that runs on their computing platform of choice. For students and educators familiar with the art of programming, LecTix 2.0 contributes a player that they can easily extend to better work for them, so that students can become better learners, and educators can become better teachers.

7.2 Future Work

All is not done with LecTix 2.0, however. Of course, such is to be expected from a system designed to be extensible. The two major areas to be addressed in LecTix 2.0 are features missing from LecTix 1.3, and support for a contemporary, state-of-the-art codec.

While LecTix 1.3 does suffer somewhat from feature bloat, the real problem is not that there's too many features, but that LecTix 1.3 presents them all at once. In a lecture-multimedia players, extra media can mean extra production costs. As was the case with the production of lecture-multimedia for *6.046: Introduction to Algorithms* (see Chapter 6), extra production costs often means that the extra media do not get produced. Hence, the students did not care much for a player that devoted over half of its screen space to features that weren't being used.

A better approach would be to carefully limit how much the player shows to the student at once. Students like to watch large video, sometimes to the exclusion of everything else. Other students, however, may find a transcript essential. It is the job of the lecture-multimedia player to accommodate several different kinds of users, possibly including the student that seeks information overload from 10 simultaneous streams of media.

To address this issue, future work on LecTix 2.0 could include looking at ways to make the user interface *easily* customizable. I stress the word *easily*, because a hard-to-customize interface is just as bad as an uncustomizable interface. Liu [25] has suggested preset views similar to those in Eclipse [10].

As for the matter of finding a contemporary, state-of-the-art codec for LecTix 2.0, we must remember that compatibility in lecture-multimedia players often conflicts with distributability. Fortunately, however, three new codecs are on the horizon that will break that conflict: Ogg Theora [47], Dirac [4], and the Snow codec from the FFmpeg project [11].

With these missing puzzle pieces in place, LecTix 2.0 can become an even more usable, available, and extensible player.

Glossary

codec An acronym for "compressor / decompressor." A codec is a set of algorithms, or implementation thereof for (1) reducing the size of (compressing) and encoding a single media signal (such as video or audio) into a stream of bytes; and for (2) decoding and uncompressing the stream of bytes to reconstruct, if not the original media signal, a media signal similar to the original.

format A file format that acts as a container for byte streams that are the result of codecs compressing media signals. Often the byte streams are multiplexed and synchronized so that they can be presented simultaneously (such as synchronized audio and video).

open source Software, or licenses for software, that conform to the Open Source Definition (OSD) [32] as specified by the Open Source Initiative (OSI). The OSD specifies ten criteria that software must comply with in order to be considered open source. The three criteria important for discussion in this thesis are:

- *Free Redistribution*: The software must be freely redistributable. The license must not require a royalty or fee upon resale or redistribution.
- *Source Code*: The software must include source code, or the source code must be made available upon request.
- *Derived Works*: "The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software" [32].

The remaining seven criteria focus on the redistribution of the license and the

allowance of the software for use by anyone and for any purpose. When hyphenated, "open-source" becomes an adjectival noun, as in "open-source software" or "open-source license."

platform A combination of a specific operating system and hardware architecture. For example, Linux/i386 is a different platform than Linux/Alpha despite them having the same operating system.

Bibliography

[1] Jeff Ayars et al. Synchronized Multimedia Integration Language (SMIL 2.0). http://www.w3.org/TR/2004/PER-SMIL2-20041105/, Nov 2004. W3C proposed edited recommendation.

[2] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Object Technology Series. Addison-Wesley, Reading, Massachusetts, 1999.

[3] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/2004/REC-xml-20040204/, Feb 2004. 3rd edition, W3C recommendation.

[4] British Broadcasting Corporation. Dirac. http://www.bbc.co.uk/rd/projects/dirac/index.shtml.

[5] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, 1983.

[6] Antoine Cellerier et al. VideoLAN Client. http://www.videolan.org/.

[7] Bernard of Chartres, circa 1126. Quoted in [28].

[8] Columbia University. Columbia Video Network. http://www2.cvn.columbia.edu/.

[9] Apple Computer. QuickTime file format license. http://developer.apple.com/softwarelicensing/agreements/pdf/qtfileforma%t.pdf.

[10] Eclipse Foundation. Eclipse. http://www.eclipse.org/.

[11] FFmpeg Team. Ffmpeg multimedia system. http://ffmpeg.sourceforge.net/.

[12] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, June 1954.

[13] Free Software Foundation. GNU General Public License. http://www.gnu.org/licenses/gpl.html.

[14] David M. Geary. *Graphic Java 1.2: Mastering the JFC*, volume 1. Sun Microsystems Press, Palo Alto, CA, 3rd edition, 1999.

[15] the GNU Compiler Collection Team. Host/Target specific installation notes for GCC. http://gcc.gnu.org/install/specific.html.

[16] Adele Goldberg and David Robson. *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley Series in Computer Science. Addison-Wesley Professional, Reading, Massachusetts, 1983.

[17] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. The Java Series. Addison-Wesley Professional, 3rd edition, June 2005.

[18] Kai Huang, Charles E. Leiserson, and Luis F. G. Sarmenta. A media player for use in distance education. http://hdl.handle.net/1721.1/3864, January 2003. Poster presented at the Singapore-MIT Alliance Third Annual Symposium, Singapore.

[19] Illinois Institute of Technology. IIT Online. http://www.iit-online.iit.edu/.

[20] International Telecommunication Union. H.263: Video coding for low bit rate communication. http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-%REC-H.263, Jan 2005.

[21] ISO/IEC. ISO/IEC 11172-1993: MPEG-1 coding of moving pictures and associated audio at up to about 1.5 mbits/second, 1993.

[22] Jikes. `http://jikes.sourceforge.net/index.shtml`.

[23] Zentaro Kavanagh. Ogg Directshow Filters. `http://www.illiminable.com/ogg/`.

[24] Barbara H. Liskov and Jeannette M. Wing. Behavioral subtyping using invariants and constraints. In Howard Bowman and John Derrick, editors, *Formal Methods for Distributed Processing: A Survey of Object-Oriented Approaches*, chapter 12. Cambridge University Press, Cambridge, 2001.

[25] Vicky Liu. Personal communication, Nov 2004.

[26] I. Scott MacKenzie. A note on the information-theoretic basis for Fitts' law. *Journal of Motor Behavior*, 21:323–330, 1989.

[27] Marco Dolcetto Mate, Darlene Mari Velasquez, and Luis F. G. Sarmenta. An enhanced lecture viewer for eLearning. In *3rd National Conference on eLearning*, Manila, August 2004. Philippine eLearning Society.

[28] Robert K. Merton. *On the Shoulders of Giants: A Shandean Postscript.* University of Chicago Press, Post-Italianate edition, 1993.

[29] Microsoft. Microsoft Producer. `http://www.microsoft.com/windows/windowsmedia/technologies/producer.aspx`.

[30] Microsoft. The .NET show. `http://msdn.microsoft.com/theshow/`.

[31] Jacob Nielsen. *Usability Engineering.* Academic Press, San Diego, CA, USA, 1993.

[32] Open Source Initiative. Open Source Definition. `http://www.opensource.org/docs/definition.php`. version 1.9.

[33] Jim Pick. Kaffe.org — ports. `http://www.kaffe.org/ports.shtml`.

[34] The MPlayer Project. MPlayer: The movie player. http://www.mplayerhq.hu.

[35] RealNetworks. Helix Community — RealAudio and RealVideo technology for Helix. https://helixcommunity.org/realcodecs/.

[36] RealNetworks. Helix Community Project Info — Xiph. https://helixcommunity.org/projects/xiph/.

[37] RealNetworks. Helix Player. http://player.helixcommunity.org/.

[38] RealNetworks. RealPlayer. http://www.real.com/.

[39] J. Rintala. Computer technology in higher education: An experiment, not a solution. *Quest*, 50(4):366–378, 1998.

[40] S. Roucos and A. M. Wilgus. High quality time-scale modification for speech. In *Proceedings ICASSP 86*, International Conference on Acoustics, Speech, and Signal Processing, pages 493–496, Tokyo, March 1985.

[41] Marco Schmidt. Java development kits and Java runtime environments (JDK / JRE). http://www.geocities.com/marcoschmidt.geo/java-jdk-jre.html, May 2004.

[42] Singapore-MIT Alliance. http://web.mit.edu/sma/.

[43] Richard M. Stallman. The danger of software patents. In Joshua Gay, editor, *Free Software, Free Society: Selected Essays of Richard M. Stallman*, chapter 16. GNU Press, Boston, 2002.

[44] Bjarne Stroustrup. What is object-oriented programming? *IEEE Software*, 5(3):10–20, May 1988.

[45] Sun Microsystems, Inc. JMF 2.1.1 — supported formats. http://java.sun.com/products/java-media/jmf/2.1.1/formats.html.

[46] University of Minnesota. UNITE Instructional Television. http://www.unite.umn.edu/streaming-video/index.shtml.

[47] Xiph.Org Foundation. Ogg Theora. http://www.theora.org/.