Xstreamulator: A Rich Media Webcasting Application for Lectures and Events

Jeremy M. Littler

Project submitted to Ryerson University in partial fulfillment of the requirements for the

degree of

Master of Arts

in

Communication and Culture

Dr. Michael Murphy

Dr. Jerry Durlak

February 27, 2008

Toronto, Ontario

Canada

Xstreamulator: A Rich Media Webcasting Application for Lectures and Events

Jeremy Littler

Abstract

Xstreamulator is a .NET based webcasting application that utilizes the Microsoft Windows Media Server to broadcast classroom lectures and events. Uniquely, the application supports the synchronized delivery of captured bitmap content (slides), which are displayed in an ASP/HTML based cross-browser viewing environment. At present, Xstreamulator supports bitmap slide capturing from PowerPoint presentations, computer desktops, images, web pages and external VGA sources. Additional capture capabilities are currently in development. Although Xstreamulator has been used extensively for live webcasting, it can also be employed to record webcasts for distribution through on-demand delivery or removable media. In contrast to commercial solutions, Xstreamulator's live webcasting functionality is not constrained to traditional academic settings (i.e., classrooms). Indeed, many instructors at Ryerson University have successfully employed Xstreamulator to webcast lectures from their office or home. In addition, Xstreamulator has been employed effectively in the delivery of events, lectures, symposiums and conferences.

Xstreamulator has from the outset been designed to operate reliably in diverse

hardware environments. For example, the application can be installed on personal computers, classroom presentation systems, or portable encoding "stations". Thus, by leveraging the existing computer infrastructure at Ryerson University, it has been possible to circumvent the acquisition of costly commercial webcasting systems. Xstreamulator's comprehensive content delivery approach and hardware neutrality has addressed the entire range of webcast requirements within the University environment in very cost effective and scalable manner.

Xstreamulator's development process has been driven by the philosophy of participatory design (PD). Students, faculty and staff at Ryerson University have generously donated their time to test Xstreamulator prototypes, and have contributed significantly to the evolution of the application's user interface and functionality. Therefore, the Xstreamulator project demonstrates the significant advantages of implementing participatory design goals in the development of rich media webcasting solutions. Indeed, while the technological achievements of the project are noteworthy, they could have only been achieved in an environment that fostered collaboration at all levels.

The development of an in-house webcasting solution requires a commitment of development personnel and technical resources. However, the cost of providing these in-house resources will be offset by reduced webcasting costs over the long-term. Additionally, applications like Xstreamulator can be rapidly employed to generate

webcasting revenue from university events (e.g., conferences). In summary, as the use of Xstreamulator at Ryerson University has eliminated a dependence on commercial solutions, it has been possible to re-assign these cost savings to the design of some of the most powerful event webcasting systems in North America.

Acknowledgments

This project would not have been possible without the support of the faculty, staff and students at Ryerson University, who have generously donated their time and expertise. First, I would like to thank my advisors, Dr. Michael Murphy, Dr. Jerry Durlak, and Dr. Paul Hearty, for their ongoing support and advice. I would also like to extend my appreciation to Brian Lesser for permitting me to undertake the Xstreamulator project.

I would be remiss in not acknowledging the significant contribution of the members of Xstreamulator's beta test team. Specifically, I would like to extend my thanks to John Hajdu, Steve Pelletier, Stephanie Walsh, Janet Lum, Wendy Freeman, Silvia Carfora, Gerry Amram, Sean Sedlezky, Stephanie Goetz, Andrew Furman, John Sustersic, Carol Stewart, Kim Snow, Many Ayromlou, Bill Brishna, Jim Loney, Deborah Fels, Dana Lee and Brad Fortner, for contributing so much to Xstreamulator's development. Your willingness to push the technological boundaries of webcasting has undoubtedly made Xstreamulator a far better application. I would also like to thank the staff at Media Services, and Computing and Communication Services, for their tireless efforts in helping to make Xstreamulator event webcasts so successful.

Finally, I wish to extend my deepest appreciation to my wife Katherine, and to my daughters Emma, Charlotte and Madeline, whose support and encouragement has enabled me to undertake a project of this magnitude.

Table of Contents

Figures

Chapter 1

Introduction

1.1 Overview

The implementation of a commercial webcasting solution in a typical university, with its large number of classrooms and geographically distributed infrastructure, can result in a costly dependence on single-solution webcast technology providers. Furthermore, commercial webcasting solutions may not be flexible enough to address the diverse requirements of an institution and its departments. Xstreamulator addresses these issues by providing a more adaptable and cost effective approach to webcast delivery. Specifically, Xstreamulator does not require proprietary computers, specialized capture hardware, or complex server applications. Any Windows XP/VISTA compatible computer is capable of running the application, and Xstreamulator webcasts can be viewed on a variety of popular web browsers. Furthermore, Xstreamulator leverages computer hardware present in many university classrooms and enables instructors to use their own computers to deliver media rich webcasts at a time and location of their choosing. Thus, Xstreamulator's hardware neutrality and flexible content delivery approach challenges the assumption that commercial webcasting solutions are the only viable webcast delivery option.

1.2 Project Objectives

The primary objective of the Xstreamulator project was to design a flexible solution for the delivery of classroom-based university lectures, which would support the real-time webcasting of video, audio and synchronized presentation content. While early prototypes of Xstreamulator were developed specifically for this purpose, it soon became apparent that the application could be extended to encompass the following goals:

- That the application support webcasting from classroom and non-classroom locations.
- That the application support event webcasting (e.g., conferences, symposiums).
- That the application be designed so that non-technical users could produce professional results.
- That the application be platform neutral and support a wide range of audio, video and external (VGA) capture devices.
- That the application provides content delivery functionality not available in commercial webcasting solutions.
- That the application development process actively involve staff and faculty at Ryerson University.

At the outset it was decided that the Xstreamulator's project would implement a participatory design process (see Chapter 2). Specifically, instructors and staff at Ryerson University were encouraged to contribute to Xstreamulator's development at every stage of the project lifecycle. Xstreamulator's PD approach was not based on a rigid methodology. Instead, it consisted of a more informal information gathering process,

relying on personal interviews, phone conversations, email correspondence and observations of the application in everyday use. Recommendations gathered during the ongoing PD process were implemented in subsequent Xstreamulator releases, through a process of Rapid Application Development (see Chapter 2). Significantly, the PD approach was capable of being tailored to address the inherent difficulties faced in developing a software solution in an environment where resources were limited and faculty availability was constrained. Furthermore, Xstreamulator's development process revealed that application architects who wish to include PD in their methodological approach should be prepared for ongoing changes in the membership and commitment of their PD team. Indeed, it appears that PD teams are inherently unstable over time, and that the successful development of a rich media application in a university environment requires an adaptable (i.e., tenacious!) development process.

1.3 Project Benefits

As Xstreamulator was developed in-house it was possible to rapidly implement content delivery functionality not provided by commercial webcasting solutions. Specifically, this functionality consisted of powerful content integration components and support for flexible webcast delivery scenarios. In addition, during the development process it became evident that the Xstreamulator platform could be extended to support non-classroom and event webcasting scenarios.[1] The adaption of the Xstreamulator framework to support a variety of delivery scenarios was a significant project milestone, as it enabled the

application to be put into "production" while other potential uses were being explored. Although Xstreamulator was primarily developed for live webcasting, it was readily adopted as a post-delivery solution. Furthermore, Xstreamulator's post-delivery capability effectively addressed situations where live webcasting was unfeasible. Ultimately, the Xstreamulator framework was expanded to support five distinct webcast delivery scenarios:

- The webcasting of on-campus events using a mobile Xstreamulator system.[2]
- The webcasting of off-campus events using a portable Xstreamulator system (i.e., a laptop encoding station).
- The webcasting of classroom lectures using in-podium Xstreamulator encoder systems.
- The webcasting of lectures and presentations from classrooms, offices and off-site locations.
- The recording of lectures for post-delivery and distribution via removable media.

Xstreamulator was also developed to support webcasting from fixed and wireless Internet access points, and was successfully employed in this manner from offices and residential locations. In contrast, commercial systems are too costly to be used outside of monitored environments (e.g., classrooms), and continually require dedicated production personnel. Furthermore, commercial webcasting solutions necessitate hardware/software acquisitions that grow proportionally with the number of systems in use. Xstreamulator's flexible platform-neutral and user-driven approach uniquely addresses these constraints. Moreover, Xstreamulator can be distributed to an ever growing user community with

minimal cost. The project demonstrates that the webcasting of lectures and events within academic environments need not be dependent on trained technicians or proprietary hardware/software solutions. In addition, the project establishes that non-proprietary implementations can be tailored to specific pedagogic objectives. Ultimately, it is hoped that the Xstreamulator project will encourage the academic community to extend classroom-based lectures to Internet audiences. Clearly, Xstreamulator offers new opportunities for e-learning and course delivery.

Though the development of a customized webcasting solution is not easy, the investment of time and resources becomes increasingly viable when large scale commercial systems are only alternative. Furthermore, the development of an in-house webcasting solution enables the rapid implementation of application enhancements, as needs arise. However, the Xstreamulator project clearly demonstrates that the determination of application features and functionality should be derived through a process of staff, faculty and student consultation.

Chapter 2

The Participatory Design (PD) Process

This chapter discusses the methodological approach employed in Xstreamulator's development. The implementation and findings of the participatory design (PD) research process are revealed.

2.1 Theoretical Directions and the Benefits of Participatory Design

The fundamental theoretical direction of this project encompassed the perspective that rich media e-learning environments are spaces for co-operative learning. Learning is not as one might perceive an inherently individualistic process. In reality, it is the result of a complex process of social interaction. In essence, to understand of how meaning is created, practitioners must approach e-learning environments from a sociocultural standpoint. Therefore, the process of formulating pedagogic criteria for evaluating the requirements of e-learning environments inevitably requires us to consider theories and methodological approaches derived from the sociocultural tradition. However, there are a many theoretical and methodological approaches that can be considered as valid evaluative techniques, though some of these more traditional techniques may entail less socially constructed and arguably less pedagogically valid forms of inquiry. For example, Teemant (2005) asserts that it is generally accepted that a more constructivist or group-based approach to learning has significant cognitive benefits:

> Learning occurs through internalization and automatization of social activities. Individuals construct personal understandings and abilities by way of cooperative interactions and negotiation of shared meanings in social contexts. Language and other social tools mediate learning, and structured experiences can produce expected patterns of development. (Teemant et al., 2005, p. 1176)

Sociocultural sensitivity requires practitioners to study e-learning environments as "knowledge-construction systems" (Barab, Kling, Gray, 2004, p. 259), where pedagogic analysis emphasizes the value of consensus building. This "convergence" approach, which has been a central component of the sociocultural tradition for some time, is perhaps best exemplified by Marsha Scott Poole's structurational theory of the group decision making process (Littlejohn, Foss, 2005). Ultimately, research in this area indicates a need to apply meta-theoretical frameworks (e.g., Berger and Luckman's social constructionism) and potentially, secondary methodological approaches (e.g., ethnographic and ethnomethodological research). Therefore, e-learning tools (e.g., rich media webcasting applications) and the online environments they foster must be evaluated using methodological approaches that address the community in all its varied forms (Littlejohn, Foss, 2005).

As Dickey (2003) reveals, collaborative models run contrary to the more traditional conceptualizations of the communication process, including those based on Shannon and Weaver's transmission model (1949), where the flow of information is essentially a transfer of information from an active sender (e.g., an instructor) to a passive receiver (e.g., a student):

> Concurrent with the development of new tools for distance learning has been an epistemological shift in paradigms of learning from an objectivist perspective to a constructivist perspective. Central to a constructivist theoretical perspective is the belief that knowledge is constructed, not transmitted, and that learners play an active role in the learning process. (Dickey, 2003, p. 105)

Of particular significance is Teemant's assertion that e-learning environments should encourage social interaction. For example, within the pedagogic literature is generally accepted that in e-learning communities, participants "share a social responsibility to learn from and learn for the community." (Barab, Kling, Gray, 2004, p. 26). There is an almost overwhelming consensus in the literature that sociocultural research approaches are central to interpreting these "qualitatively distinct structural environments" (Jaffee, 2003, p. 233) which are "capable of reshaping the role behaviours and social relations between instructors and students" (Jaffee, p. 233). However, it is highly unlikely that a single sociocultural theory or methodological approach could address e-learning objectives within all university environments, or that a single approach would be universally accepted. Instead, the process of implementing "experimental" e-learning technologies should encourage practitioners to reformulate a variety of theoretical and methodological approaches.

Sociocultural research on e-learning environments has also revealed that "non-verbal channels carry more information and are believed more than the verbal band." (Wheelan, 2005, p.169) Thus, rich media webcasting applications should enable the transmission of

non-verbal activity in all its varied forms. In this regard Xstreamulator is uniquely positioned, as it supports the recording of gestures, body language, and content from an almost unlimited array of electronic sources. However, given the complex nature of the underlying technology, it is extremely difficult to convey verbal and non-verbal content in a universally accessible manner. Therefore, it is encouraging that Think-Aloud-Protocol (TAP) approaches are now being extended to evaluate accessibility in software interfaces (Roberts, Fels, 2005). Furthermore, while Xstreamulator's accessibility goals are as yet only partially realized, the inclusion of accessibility experts in the application's development process has contributed significantly to the elevation of these goals in future Xstreamulator releases. Unfortunately, TAP and many other research approaches were determined to be too resource intensive to be implemented in the Xstreamulator project. Indeed, one of the projects initial challenges was the identification of a more manageable approach to addressing the rich media webcasting requirements of the University. Ultimately, participatory design (PD) was selected as the most appropriate and sustainable research methodology for the Xstreamulator project.

Participatory design (PD) is a research approach that actively seeks the involvement of end-users in the application development process. Historically, the approach has been employed in fields relating to the development of human environments (e.g., urban development, architecture). Today, it is perhaps best known as a method of addressing user needs in the area of information technology (IT). One of the primary factors in the growing use of PD research is the observation that many IT based projects fail because

they ignore the needs of the end-user or the broader requirements of the organization (Damordan, 1996). Therefore, it is now an accepted principle that formalized user participation in information technology projects is a prerequisite for success. (Butler, Fitzgerald, 1997)

The implementation of PD research methodologies can be broadly described as serving two purposes. First, the active involvement of users in the design of software applications results in products that better address the need of the entire user community (e.g., instructors and students). Second, involvement of users in the decision making process raises the likelihood that an application or process will be readily adopted by the target group. Underlying these principles is the concept of the value of Workplace Democracy, as espoused by Muller and Kuhn, particularly as it relates to a user-centered approach to technology implementation (Damordan, 1996). However, there are many challenges to implementing PD approaches, not the least of which is the necessity to protect and maintain PD teams despite institutional and extraneous obstacles. However, as Beirne (1998) reveals, rationalistic design approaches are also fraught with many challenges:

> In an oft-cited paper, Curtis et al. (1988) demonstrate that rationalistic design methods are inadequate in themselves since the context and target environment of 'live' software projects frequently display a complexity and subtlety that is beyond conventional computing textbooks and models. Related research by Guindon and Curtis (1988) has taken this further, indicating that the software process is less than rational in practice, with opportunism tending to characterize the behaviour of engineers and designers. (Beirne, Ramsay, Panteli, 1998, p. 302).

The implementation of PD research within universities should raise the likelihood that applications will become the product of many academic "voices", and thus be far more flexible than commercial approaches that assume rigid content delivery modalities. Ideally, the PD process will lead to a more thorough understanding of user requirements and the recognition that inflexible delivery approaches are of limited value. Therefore, though the implementation of a PD process introduces many challenges, it enables developers to focus on advancing software projects in a more inclusive and sustainable fashion. Moreover, the objective of "focusing on designing work, rather than just technology" (O'Day, Bobrow, Shirley, 1998, p. 315) necessitates the adoption of a constructive process of tailoring software interfaces and functionality to "real world" communication processes and user communities. Thus, PD was selected as the most appropriate research approach, as it enabled the development of a rich media e-learning application that addressed the requirements for Ryerson's academic community and the institution as a whole.

Though the use of PD in IT projects is commonplace, the use of the approach in the development of applications for e-learning is less established. However, there is a growing body of work in this field, including the application of PD approaches in the development of virtual reality (Anderson, 2001) and video applications (O'Conner, Fitzpatrick, Buchannan-Dick, McKeown, 2006). While further research is required in this area, the success of the Xstreamulator project suggests that PD approaches are ideally suited to the development of a variety of e-learning technologies in academic environments. It is hoped

that the success of Xstreamulator's PD process will encourage others to apply and further refine the approach.

2.2 Implementation and Findings of the Participatory Design Process

The Xstreamulator beta test team at Ryerson University consisted of faculty members, staff and students. Members of the beta test team were provided (via CD or direct download) with prototypes for personal use, or were given access to the application installed on computers in podium technology (PT) classrooms at Ryerson University. In general, the beta test team members could be described as early adopters. The team possessed varying technical expertise and familiarity with rich media delivery technologies. In total, the beta test team consisted of faculty and staff from the following departments at Ryerson University: Business Management, Retail Management, Radio and Television Arts, Applied Geography, Continuing Education, Nursing, Interior Design, Social Work, Disability Studies, Politics and Governance, The Digital Media Projects Office, Media Services and Computing and Communication Services.

It was decided that any interested faculty/staff member at Ryerson University could participate in the project, as it was hoped that by being inclusive the application would better reflect a wide range of delivery practices. However, potential beta testers were warned that the project was experimental. They were also informed that the application's functionality and user interface (UI) design would undergo frequent updates. The methods

used to encourage potential beta testers to participate in the project included faculty conferences, prototype demonstrations, articles, live webcasts and webcast postings. However, the most useful recruitment method was determined to be word-of-mouth recommendations. Specifically, it was revealed that potential beta testers were frequently directed to the project on the advice of existing members of the beta test team. In addition, potential beta testers were often informed about the Xstreamulator project through their interaction with staff at Ryerson's Digital Media Projects Office. Indeed, faculty requests for webcasting assistance were a significant factor in Xstreamulator's development, as they helped to highlight the limitations of commercial webcasting solutions. The absence of a commercial webcasting solution at Ryerson University provided an opportunity for experimentation in this area. It is also likely similar projects would be more difficult to implement in institutions where proprietary solutions have become entrenched.

Xstreamulator's PD research was carried out through an informal process of face-to-face consultation. In general, potential application functionality was discussed with the beta test team during the application's installation and update stages. In addition, communication technologies (e.g., email, telephone, web pages) were employed to augment the feedback collection process and to inform team members of project milestones. Ideally, a more expansive PD process would have included group "brainstorming" sessions. However, this was unfeasible, as the beta test team could not commit to an ongoing schedule of meetings. Consequently, it was decided that a Rapid Application Design (RAD) process would be implemented to encapsulate user feedback

directly in the Xstreamulator application framework. In effect, Xstreamulator is a digital representation of contributions from participants on the beta test team.

As new application functionality could not be implemented sequentially, team members were not informed about the prioritization of recommendations. However, attempts were made to include the core elements of recommended enhancements at the earliest opportunity. New functionality would then be further improved and extended as time permitted. In addition, suggestions from the beta test team were freely discussed amongst the entire team. Often, this approach encouraged team members to contribute to elements of Xstreamulator that they had not initially considered worthwhile. Furthermore, as the RAD process enabled functionality to be added at near real-time, the beta test team were made aware that their input in the project was being rapidly implemented. For example, one instructor commented that "I make a feature request and Jeremy programs it in half an hour!" While this statement was overly optimistic, the end result was an extremely fast implementation of team recommendations. As anticipated, the recommendations of the beta test team focused primarily on improvements to Xstreamulator's usability and rich media content capturing functionality. However, application requests covered a wide range of functionality, including: improved capture device support, the ability to "markup" presentation content, podcasting support, the implementation of editable webcasting presets, cross platform support, application stability and so forth.

Major enhancements and modifications to the Xstreamulator's core functionality were

regularly vetted with members of the beta test team. For example, the decision to "centralize" archived webcasts led to the implementation of a user request to enable instructors to place content in personal FTP folders. Interestingly, one of the primary findings of Xstreamulator's PD process was discovery that instructors want to maintain control over the distribution and location of their webcast content. While technical reasons dictated that a more rigid archiving model was required, the addition of support for user-specified hosting addressed concerns that the content would be permanently locked to a single server or delivery platform. In comparison, commercial webcasting systems make transferring content to alternate servers extremely difficult or impossible. Xstreamulator addresses this issue by generating the entire viewing environment on the user's local hard drive. Furthermore, the viewing environment can be easily published to any Windows Media server or Web server.

If possible, beta testers were observed using Xstreamulator prototypes in live and non-live webcast sessions. The observation of the application's use by instructors in their working environments revealed problems with the application's UI, technical constraints, and other functionality issues. These sessions were in essence an informal implementation of Talk-Aloud-Protocol (TAP) procedures. Webcasts were also monitored remotely. The approach consisted of observing the webcast and student/instructor comments in Xstreamulator's chat interface. This process helped to reveal issues relating to student access and impressions of Xstreamulator's webcast delivery model. As chat sessions were archived in later Xstreamulator prototypes, it became possible to review these sessions

during subsequent prototype development stages. While institutional demands precluded direct student involvement in the project, it was possible to gather student feedback by actively participating in online sessions. Furthermore, the response from the student community was overwhelmingly positive. In general, students appeared very willing to contribute their ideas to the project. Indeed, many of Xstreamulator's unique features were the direct result of feedback from these sessions. For example, the student community requested the addition of podcasting functionality and the attachment of PowerPoint slideshow summaries to archived webcasts. The most significant challenges faced by the student audience were browser/player incompatibilities. Overall, these issues were much more common on the Macintosh OS/X platform. To address these compatibility issues, an HTML help file was added to the webcast viewer interface. This guide greatly assisted students in optimally configuring their browser/player environments.

Unexpectedly, a review of chat sessions revealed that a significant amount of chat "traffic" occurred between students, though student-to-instructor chats remained the most common. Generally, student-to-student chats pertained to the clarification of themes being presented by the instructor. However, a significant portion of the chat traffic was revealed to be social (e.g., often supportive) in nature. Significantly, many students indicated that the chat sessions were as important to their learning outcomes as the webcast content itself. The student's adaptability to Xstreamulator's chat environment and the generally positive chat messages suggest that students were comfortable communicating in Xstreamulator's e-learning environment.

Interestingly, the production of webcast content was driven as much by student demand as instructor interest. For example, an instructor on the beta test team commented that her students had become "addicted" to Xstreamulator produced content! In addition, many students stated that Xstreamulator webcasts addressed distance barriers. Ultimately, the evaluative component of the PD research process combined feedback from instructors and students with the evaluation of live and on-demand webcasting sessions. In addition, members of the beta test team were encouraged to relay both their experiences and feedback from students. It is expected that future work on the Xstreamulator project will augment these approaches with group "brainstorming" sessions and a greater degree of student participation.

To improve usability, features were only added to the Xstreamulator's framework if they streamlined the webcasting process or enhanced the quality of media produced. Indeed, Xstreamulator's primary development goal was simplifying the webcasting process for users. Ideally, users wanted to be protected from the complexities of webcasting process. This concern was raised in the literature and by many members of the beta test team. Nevertheless, as Lee (2006) asserts, it is unlikely that users could ever be fully protected from the cognitive demands of a webcast environment, as the nature of webcasting itself introduces challenges that are not exclusively technical:

> The conditions imposed on an online session are not just technological in nature. The instructor must apply a high degree of implicit communication during the presentation. This means that the professor should use eye

> contact by looking at the webcam lens as if it were an actual student. Facial expressions and vocal inflections are also important to help explain and clarify key points. In effect this is television, and an ability to play to the camera and be an on air presenter at a professional level is a definite asset. (Lee, 2006, p. 2)

Nevertheless, the success of the Xstreamulator project suggests that users are capable of overcoming these issues. Furthermore, after a short familiarization process, users can produce content of a very high standard. In addition, Xstreamulator can be implemented in classroom environments in a manner that isolates instructors from the complexities of the webcasting process. Thus, lecturers may perceive webcasting (and by association e-learning) as simply an extension of the lecture delivery process.

It was assumed that instructors would primarily use Xstreamulator for delivering live webcasts. However, it became evident that Xstreamulator was increasingly being utilized for post-delivery scenarios. When questioned about this practice, some instructors indicated that preparing content in non-live environment was more relaxing and avoided the pressures of real-time delivery. Conversely, others stated that live webcast sessions were more productive, as they involved direct communication with students (i.e., via Xstreamulator's chat component). Irrespectively, Xstreamulator's support for live/post webcast delivery from non-classroom environments (e.g., offices, boardrooms, home locations) was universally popular with the beta test team. As a result, automated post-event webcast uploading functionality was added to Xstreamulator's feature set. The facility for the automated uploading of pre-recorded webcast content overcame the technical challenges of webcast delivery in non-networked environments.

Research on the use of e-learning technologies in universities has revealed that many instructors "have had no training and little experience in the use of communications and information technology as an educational tool." (Joyes, Scott, 2000, p. 74) Therefore, the wide scale adoption of webcasting technologies within academic environments is not easily achieved. This challenge was addressed by implementing methods to reduce the complexity of the entire webcasting process. Ultimately, usability goals influenced every aspect of Xstreamulator's design process. Thus, the Xstreamulator model should be perceived as challenge to the widely held view that instructional webcasting solutions necessitate highly skilled technical support teams. However, it is worth noting that the composition and commitment of the beta test team fluctuated substantially throughout the project's development lifecycle. While this can be partly explained by the Xstreamulator's long development process it also appears that commitment to the project reflected user requirements. For example, the installation of early Xstreamulator prototypes in PT classrooms at Ryerson University was spearheaded by instructors who wished to use webcasting in an upcoming collaborative Social Work program. Once the project was completed, the instructors became far less active in the project. However, a subsequent requirement to broadcast meetings encouraged the same instructors to become more active. Interestingly, some members of the beta test team repurposed their use of Xstreamulator to fit new delivery objectives when their initial purpose for using Xstreamulator were no longer applicable. Indeed, the Xstreamulator projects demonstrated that there is potentially no limit to the usefulness of an application with

flexible webcasting capabilities. For example, an instructor in the School of Retail Management employed Xstreamulator to live broadcast telephone interviews, while other instructors employed Xstreamulator to introduce their program to incoming students. It is highly likely that a more rigid development process would have led to an application that was far less adapted to these varied delivery scenarios.

Finally, the PD research process revealed that webcasting technology can overcome institutional barriers. For example, Xstreamulator directly addressed the challenge of course delivery in situations where factors of geography and professional commitments acted as barriers to enrichment (Ostrow, DiMaria-Ghalili, 2005). Indeed, the ability to produce webcasts outside typical daytime teaching schedules was a major factor in the willingness of faculty to adopt an arguably experimental webcast delivery system. A second factor that attracted instructors to the Xstreamulator project was a lack of classroom availability at Ryerson University (Lee, 2006). Specifically, many instructors indicated that the shortage of classroom space at Ryerson made Xstreamulator the only viable course delivery alternative. It is also likely that classroom shortages are commonplace in many universities. Thus, as Xstreamulator enabled instructors to deliver content from a location of their choosing, the application could be viewed as an extremely cost effective alternative to traditional course delivery methods.

Chapter 3

Commercial and Open Source Alternatives to Xstreamulator

This chapter provides a critique of commercial and open source alternatives to Xstreamulator. The advantages and disadvantages of these systems are discussed as they specifically relate to their implementation within university environments.

3.1 Critique of Commercial Alternatives to Xstreamulator

There are number of hardware and software solutions tailored to recording classroom based presentations. However, only a handful of commercial solutions combine live webcasting with integrated desktop/presentation capturing functionality. The two commercial leaders in this field are Sonic Foundry (MediaSite) and Accordent Technologies (Capture Station). The MediaSite RL440/ML Recorder and the Accordent Capture Station are single-purpose webcasting systems that provide integrated audio, video and VGA capturing functionality. Specifically, the Sonic Foundry and Accordent products utilize third-party hardware for audio/video and desktop capturing, and are based on a standard PC architecture. The cost of either system is well over $15,000.

As the Sonic Foundry and Accordent systems are incapable of processing more than one webcasting event simultaneously, enough encoding systems must be purchased to match the number of concurrent sessions taking place. Due to these capacity constraints,

some institutions (e.g., York University in Toronto, Canada) locate their commercial capture systems in a centralized webcasting studio, where a "rack" of capture systems is pooled to support a larger number of classrooms. While centralization enables institutions to maximize their use of these systems in the short-term, the approach cannot ultimately overcome scalability constraints. For example, if the webcasting of classroom lectures becomes popular within an institution, the number of resident capture systems must grow to match the peak webcasting demand at any given time. Thus, adopters of commercial webcasting solutions can become captive to their own success. In comparison, Xstreamulator's hardware neutral design permits the deployment of a potentially unlimited number of capture stations. Furthermore, basic Xstreamulator capture stations can be assembled for under $3,000. Xstreamulator can also be installed on computers that are tailored to specific classroom environments, including small-form-factor PC systems, portable systems, and podium technology (PT) computers. For example, Xstreamulator prototypes were successfully tested on podium computers within a number of classrooms at Ryerson University; a delivery approach that becomes more viable with each generational improvement in PC performance. Indeed, Xstreamulator's low implementation cost enables integrated webcasting systems be deployed to any location on a university campus, including offices, conference rooms and studios. Xstreamulator can also be distributed to instructors who wish to record classroom or non-classroom lectures with personal computers (e.g., laptops). In contrast, the Sonic Foundry and Accordent solutions are specifically designed for webcasting content in environments (e.g., classrooms) where the presenter is recorded by a permanent capture system. As a result,

instructors are required to use dedicated webcasting classrooms or licensed presentation capture applications (e.g., Accordent PresenterPro). Based on experience at Ryerson University, it appears that the licensing of presentation applications may be overlooked during an investigation of webcasting solutions.

The Sonic Foundry and Accordent solutions require server licensing arrangements that add greatly to the cost of implementing these platforms. In addition, their delivery architectures require more staff than Xstreamulator. Specifically, Xstreamulator's webcast delivery process does not require that events are scheduled in advance, as the delivery framework is user-driven instead of event-based. For example, to announce a pending webcast, Xstreamulator users simply distribute their personalized webcast homepage to students (e.g., via email). Once created, an instructor's broadcast URL does not change in subsequent webcasts. Furthermore, users activate their webcast homepage directly from Xstreamulator's UI. Thus, the Xstreamulator's approach greatly reduces administrative staffing requirements. In contrast, the Sonic Foundry/Accordent broadcast models require that webcast sessions are scheduled in advance. An evaluation of a Sonic Foundry system at Ryerson University revealed that the webcast scheduling process was time-consuming and potentially error prone.

3.2 ePresence: An Open Source Alternative to Xstreamulator

Developed by Ron Baecker, Peter Wolf and by Kelly Rankin of the Knowledge Media

Design Institute (KMDI) at the University of Toronto, the ePresence system is a comprehensive open source alternative to commercial webcasting solutions. Like Xstreamulator, ePresence utilizes "standard off-the-shelf computer hardware and audio/visual equipment" (Rankin, Baecker and Wolf, 2004, p. 2890). A key feature of the ePresence system is its ability to simultaneously record presentation content in a variety of video codecs (i.e., Windows Media, QuickTime, RealMedia and Flash). However, the ePresence system only live webcasts Windows Media and Real Media streams. The ePresence system utilizes a Viewcast Osprey 210 capture card and Viewcast Osprey SimulStream software to implement multiple-format capturing. Specifically, the Osprey SimulStream (i.e., virtual) driver enables a single Osprey 210 capture card to emulate a multiple format capture device. Consequently, while users of an ePresence system are free to install the ePresence Producer application on any Windows XP compatible computer, they must purchase a costly Osprey capture card and SimulStream license if they wish to produce multi-format streaming video. In addition, VGA slide capturing is supported exclusively through an Epiphan VGA to USB frame grabber. Unlike Xstreamulator, the ePresence system does not support alternative VGA capture sources (e.g., PCI based frame grabbers) or other capture devices (e.g., cameras). A "light" version of the ePresence system does support non-Osprey capture devices, but it is not capable of external VGA capturing.

The ePresence system consists of the ePresence Server, Producer, Presenter and PDA Remote Control software applications. In situations where VGA capturing is not feasible,

PowerPoint slideshows can be dynamically captured with ePresence Presenter. However, the Windows only ePresence Presenter application must be installed on a presenter's system to enable this functionality. The ePresence Producer application can also capture PowerPoint presentations, but it must convert these files to slides prior to the start of a webcast. Testing revealed that this time consuming pre-conversion process would be problematic in a live webcasting situation. In comparison, Xstreamulator does not require software to be installed on a presenter's computer, as Xstreamulator provides integrated real-time PowerPoint capturing functionality. Furthermore, Xstreamulator is capable of dynamically loading and capturing PowerPoint presentations at any time during a live webcast session.

ePresence's support for multiple video formats is commendable. However, its Windows Media format webcasts can only be viewed on Internet Explorer 6+ in Microsoft Windows. In contrast, Xstreamulator creates a single Windows Media file that is viewable on Macintosh and Windows systems.[3] Moreover, as ePresence is restricted to live webcasting platform specific Windows Media/Real Media streams,[4] Real Networks Helix Server is required to maintain cross-platform compatibility. However, the ePresence and Helix server components can be installed in Windows or Linux. In comparison, Xstreamulator requires Windows Server 2003 and Windows Media Services. As Windows Media Services is included with Windows Server 2003, server deployment may be less costly than with the ePresence solution. The ePresence system implements database functionality through open source XML protocols, including Service Oriented

Architecture Protocol (SOAP) and MySQL. At present, Xstreamulator does not implement a database system for webcast archiving and searching. This functionality is planned for future release. Specifically, it is expected that Xarchiver will be employed to populate searchable SQL databases. In addition, front-end search functionality will likely be based on an Active Server Page (ASP) implementation.

ePresence provides comprehensive support for webcast searching and tagging. An embedded Flash applet enables viewers to "search within a particular archive or across a repository of archives based on key words" (Rankin et al., 2004, p. 2889). The system's dependence on Adobe's Flash Player could be viewed as a shortcoming, though it is likely that the vast majority of viewers have this popular web plug-in installed on their computer. Uniquely, the ePresence Producer application automatically populates the Flash applet UI with PowerPoint slide titles. Viewers of ePresence presentations can also attach markers to the webcast playback timeline and forward presentation links to other viewers. During a webcast, the ePresence system generates a table of contents that can be further edited in a post-production interface. It is expected that the Xstreamulator solution will eventually implement webcast indexing and searching capabilities that are comparable to those provided by the ePresence system.

The ePresence application suite is clearly not designed for casual use. Potential users are faced with an application architecture that is as complex as it is comprehensive. Overall, the extensive capabilities of the ePresence system are somewhat overshadowed by

the complexity of configuring and running live webcast sessions. For example, the ePresence operation guide states that an ePresence operator is required to oversee the capturing process. The guide also suggests that an audio/video operator and a moderator may be required for some webcast scenarios.[5] In contrast, Xstreamulator's non-classroom webcasting functionality is streamlined to such an extent that the delivery process can be managed solely by the presenter. Furthermore, all of Xstreamulator's classroom-based webcasting functionality (e.g., PTZ camera control and audio level management) will eventually be implemented in a unified application UI.

3.3 Summary of Commercial and Open Source Alternatives

Commercial webcasting solutions require users to conform to less flexible content delivery modalities. For example, the Accordent/MediaSite solutions are incapable of delivering full-motion video/desktop webcasts. If commercial solutions fail to address specific end-user requirements, then less optimal content delivery processes can become entrenched. Moreover, the adoption of commercial solutions can lead to the concentration of single-purpose hardware installations. In contrast, in-house webcasting solutions can be easily distributed and modified to support a potentially unlimited number of delivery environments. It is also more likely that emerging e-learning applications (e.g., video conferencing, podcasting) will be added to internally developed systems. In comparison, commercial hardware solutions are typically treated as sacrosanct environments.

The ePresence system is significantly more flexible than the solutions provided by Sonic Foundry and Accordent Technologies. For example, ePresence's development team recently added Flash-based streaming and video conferencing capabilities[6] to the feature set. Furthermore, the system's sophisticated searching/indexing architecture and support for multiple media formats/delivery modes are noteworthy. However, the consequence of ePresence's sophistication is a complex hardware/software environment and significant media storage/delivery overhead. In contrast, Xstreamulator's development process has focused on reducing the complexity of webcasting process to a point where non-technical users are able to create "professional" results. In addition, by using a single streaming format, Xstreamulator significantly reduces media storage and delivery requirements. From the outset, Xstreamulator has been designed to be operated by non-technical users (e.g., instructors). This goal has been achieved by integrating the entire production process into a user-friendly application UI.

Chapter 4

Xstreamulator and Xarchiver Webcasting Framework

This chapter discusses Xstreamulator's client/server framework. In addition, Xstreamulator's webcast delivery process and client-server architecture are described.

4.1 Overview of the Xstreamulator/Xarchiver Webcasting Framework

Xstreamulator's webcasting framework consists of two interconnected applications, the Xstreamulator "encoder" application and the server resident Xarchiver application. While Xstreamulator can deliver webcasts without being connected to a server running Xarchiver, it is not possible to synchronously archive live Windows Media streams without installing Xarchiver on a Windows Enterprise Edition server. Xarchiver directs the Windows Media Server service to archive Windows Media streams to the user's webcast directory. However, as Xstreamulator records complete webcasts to the local computer, it is possible to configure Xstreamulator to live webcast with Xarchiver's archiving functionality disabled. In this scenario, Xstreamulator post-uploads the streaming media file via the File Transfer Protocol (FTP). Though this approach precludes automatic archiving, it does enable users to install Xstreamulator-based live webcasting functionality on less costly versions of Microsoft's Windows Server; if required, Xstreamulator can be configured to provide publishing credentials to the Windows Media Server service. Thus, Xstreamulator can interoperate with any Windows Media server.

In addition to synchronous webcast archiving, a permanent server-based recording of media streams generated during a live event, Xarchiver is responsible for the activation of live webcasting "mount points", user authentication (currently in development) and a variety of pre-production and post-production functions. Significantly, situating this functionality on the server enables Xarchiver to bypass firewalls and interact directly with the Windows Media service. In comparison, commercial encoding systems typically communicate with the Windows Media service via Microsoft's Distributed Component Object Model (DCOM). This approach requires the opening an entire range of firewalled ports (1-65,535, User Datagram Protocol); a process that makes the Windows server significantly less secure. As Xstreamulator communicates with Xarchiver through a single TCP/IP port, it avoids the security implications of the DCOM approach. Moreover, a major advantage of the TCP/IP based approach is that it enables Xstreamulator user's to live webcast their presentations from locations outside of the university. The TCP/IP implementation does not require firewall reconfigurations that worry IT security staff. It is worth noting that Xarchiver's TCP/IP port can be changed to any value within the TCP range (1-65,535). Furthermore, Xarchiver can utilize a dedicated network card for load-balancing purposes or for Virtual Private Network (VPN) operations. As the Windows Media Server and FTP services are components of Microsoft's Windows Server, no additional third-party components are required to implement Xstreamulator.

Intriguingly, the Xstreamulator to Xarchiver TCP/IP approach could be implemented

as a reverse process. This would enable Xstreamulator to be remotely controlled from a web-based scheduling system. In this scenario, a scheduling system would dispatch webcasting sessions to Xstreamulator clients, and manage the webcasting process with minimal user intervention. However, to be fully automatic, the process would require the development of automatic slide capturing support. Furthermore, the approach could not be fully self-operating without automatic camera control and audio level management functionality.

4.2 The Xstreamulator/Xarchiver Webcasting Framework

Xarchiver utilizes application threading to process requests from Xstreamulator clients. The addition of threading functionality to Xarchiver was a critical development milestone. It would not have been possible to have multiple Xstreamulator "encoders" connected to a single Windows Media Server if these connections could have only been processed in a sequential manner. This challenge was addressed in Xarchiver by instantiating independent application threads for each Xstreamulator publishing directive (i.e., parallelization). Xarchiver's threading model is undergoing significant development. However, the results thus far indicate that Xarchiver's multi-threaded approach is extremely efficient and stable.

To manage live webcasts, Xarchiver communicates through a .NET Interop assembly with the Windows Media service and the Windows Media Server Object

Model/Plug-in 9.0 Type Library. This library is imported into the Xarchiver's .NET development environment through a COM reference to the dynamic link library WMSServerLib.dll. The Windows Media Server Object enables Xarchiver to interoperate with local or remote Windows Media servers. For example, Xarchiver dynamically creates and removes webcast publishing points, and instructs the Windows Media service to archive live media streams to specific user directories. The on-demand creation of webcast mount points eliminates the time consuming process of configuring webcasting accounts. Xarchiver publishes JavaScript and ASP/HTML files that monitor the streaming status of a user's live webcast account (Figure 1). For redundancy purposes, this functionality is also duplicated in the Xstreamulator client.

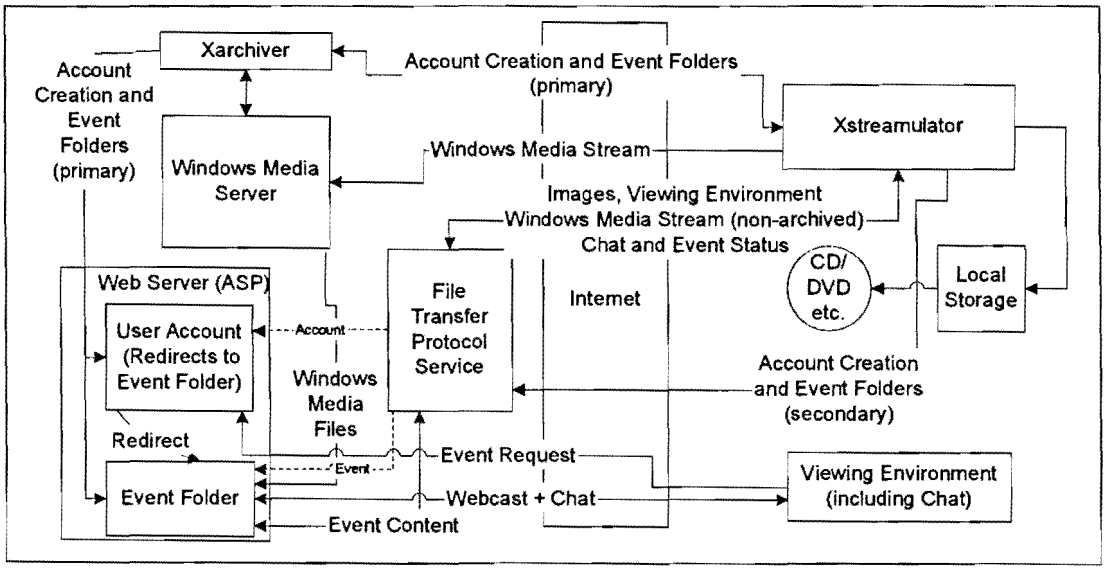

Figure 1: The Xstreamulator/Xarchiver Webcasting Framework

Furthermore, dynamically generated webcast publishing points can accept Xstreamulator webcasts that are "pushed", "pulled"[7] or (potentially) multicast. To implement server load

balancing, Xarchiver will eventually support the distribution of mount points across a "farm" of Windows Media servers.

A JavaScript-based XML parsing process enables webcast viewers to directly connect to the user's webcast directory at the moment a webcast goes "live". Presently, the XML loading process identifies the title and date/time of the event. A planned enhancement will enable instructors to send messages to the redirect page if events are running behind schedule. If a user has not commenced a webcast, viewers are informed that the webcast is pending. For example, a user wishing to view a live webcast from Instructor A will be permitted to connect to the subdirectory that is currently archiving instructor A's webcast (i.e., the location where the archived webcast will permanently reside). If instructor A has not commenced a live webcast, then the viewer will be "locked" to instructor A's redirect page (which contains the XML/JavaScript based redirection components), until the point that the webcast begins. A future Xstreamulator release will be automatically return viewers to the user's redirect page at the completion of a live webcast.

During a live webcast Xstreamulator populates the user's webcast folder with a viewing environment consisting of HTML, Active Server Pages (ASP), Asynchronous JavaScript (AJAX), media pointer files and XML. Xarchiver can also generate the viewing environment, though at present these files are uploaded to the streaming server by Xstreamulator. The viewing environment is accessed during live webcasts and archived for post-delivery. An administrative interface that enables users to modify webcast settings

and a viewer authentication wrapper is currently in development. Even if Xarchiver eventually assumes responsibility for this functionality, it is likely that Xstreamulator will maintain this role in a failsafe capacity, as this increases the likelihood that Xstreamulator webcasts will work in the event of a server or Xarchiver failure. Although the delegation of live webcast delivery responsibilities between the Xarchiver and Xstreamulator has yet to be finalized, it is expected that Xarchiver will assume a greater role in Xstreamulator's webcast delivery process. For example, the move to a server based delivery approach makes the deployment of new ASP/HTML viewing templates much easier.

At the end of a webcast, Xarchiver and Xstreamulator simultaneously commence a post-production process that adds XML based markers to the media stream. Marker insertion enables viewers to "jump" to segments of a particular webcast. At present, Xarchiver utilizes the live archiving functionality provided by the Windows Media Server service on Windows Server Enterprise Edition. However, Xstreamulator can upload marker-inserted streaming files via FTP if automatic archiving is not supported. It appears that Xarchiver's implementation of synchronous Windows Media archiving is unique. The ability to view an entire webcast immediately upon its completion is a significant feature of the Xstreamulator framework. In contrast, commercial webcasting solutions rely on a post-production process to upload webcast content. Consequently, webcasts are not immediately available for on-demand delivery.

Xstreamulator's webcast delivery framework is based on a streamlined user-driven

model. As all live webcasts are situated in the user's home directory (e.g., www.ryecast.ryerson.ca/xs/jlittler/) the need to distribute successive live webcast URLs is avoided. This approach does not require that technical staff manage webcasting schedules. In addition, the process of notifying viewers (e.g., students) of live webcasts is greatly simplified, as viewers need not continually update event URLs. Moreover, as an instructor's live webcast location is automatically generated by Xstreamulator during (or prior) to an initial webcast, the process of granting webcasting permissions is significantly streamlined; first-time Xstreamulator user need only activate their account (i.e., create their initial live webcast directory). The activation of user accounts can be completed from any system running Xstreamulator, and may eventually be enabled from a secure webpage. In an emergency, user accounts can be activated by manually copying webcast status/redirection components to a new user directory.

During a live webcasting session, Xstreamulator broadcasts user information and streaming settings to Xarchiver. After Xarchiver authenticates the Xstreamulator user, all further communication employs a secure FTP password/username. Xarchiver uses Xstreamulator broadcasted settings to configure webcasts and to populate live webcasting folders with content specific ASP/HTML viewing components. Again, this functionality is duplicated in the Xstreamulator client. In addition to creating live webcast folders, Xarchiver instructs the Windows Media Server service to create webcast mount points. In summary, the live webcasting process is as follows:

1. Xstreamulator transmits a username/password and webcast settings

to Xarchiver.

2. Xarchiver authenticates the user. From this point on, Xstreamulator employs a secure password/username for subsequent communication (e.g., the uploading of slides).

3. Xarchiver creates a webcast delivery folder for the user. This delivery folder is populated with ASP/HTML/XML components by Xarchiver or Xstreamulator.

4. Xarchiver creates a live webcast mount point and instructs the Windows Media Server service to archive live webcasts to the user's webcast directory.

5. Xarchiver instructs Xstreamulator to commence broadcasting the media stream.

6. Viewers connect to the user's webcast redirection URL. Then, they are redirected to the live webcast directory.

7. Upon completion of the webcast, Xstreamulator transfers an xml based "marker" file to the Windows Media server. Xarchiver uses this file to prepare the streaming media file for post delivery and slide playback. Xstreamulator duplicates this functionality on the local computer.

8. Xarchiver removes the publishing mount point from the Windows Media server.

9. Xarchiver sends an email to the user's account to inform them of the location of the webcast content. Users distribute this email to viewers for on-demand delivery.

If required, the above delivery process can include a secure user authentication process (e.g., a Light Directory Access Protocol based authentication process managed by Xarchiver). Also, a future release of Xstreamulator will enable instructors to include an

authentication process for viewers of live and on-demand webcasts. It is foreseeable that an authentication process could be implemented to enable viewing permissions to specific users. However, a more generalized authentication process would likely be sufficient in the majority of cases. The implementation of complex viewer authentication systems in commercial webcasting solutions (e.g., MediaSite) can result in time-consuming requests for changes to the authentication database.

Chapter 5

The Viewing Environment

This chapter describes Xstreamulator's viewer interface. In addition, Xstreamulator's "slide" insertion processes and "slide" delivery functionality are discussed. The chapter concludes with a discussion of Xstreamulator's support for cross-platform browser delivery.

5.1 Viewer User Interface (UI) and Functionality

Webcast viewers receive Xstreamulator broadcasts by connecting to a page hosted on an HTTP/Windows Media streaming server. When an Xstreamulator user captures slide content (e.g., images or PowerPoint slides) the resultant bitmap image and an HTML "wrapper" are uploaded to the webcast server via File Transfer Protocol (FTP). To synchronize slide delivery, a URL loading command is inserted in the script track of the Windows Media stream. Xstreamulator uses the intrinsic buffering time of the Windows Media Server[8] live streaming delivery process to upload the captured slide content to the webcast server before it is required for display in the viewer's browser. The difference between the time a slide URL is inserted in the Windows Media stream and the time as slide is required for display in the viewer's interface (i.e., the buffering delay) is long enough (i.e., 10-20 seconds) to post the captured file on the webcast server. However, Xstreamulator's slide capturing/insertion process is so efficient that it could potentially

operate in situations where the Windows Media Server buffering delay falls below 5 seconds. Viewers of live Xstreamulator webcasts must have access to an Internet connection. To compensate for low-bandwidth connections, Xstreamulator can be configured to generate a multibitrate stream that includes a low-bandwidth profile (e.g., 56K). Optionally, instructors can distribute self-contained Xstreamulator webcasts on removable media (e.g., CD-ROM or Flash Memory) to viewers lacking Internet access.

During a live webcast, an Xstreamulator inserted URL is detected by the embedded Windows Media Player control in the viewer's browser, and the appropriate slide is loaded via Hypertext Transfer Protocol (HTTP) in an inline frame adjacent to the embedded Windows Media Player window (Figure 2).

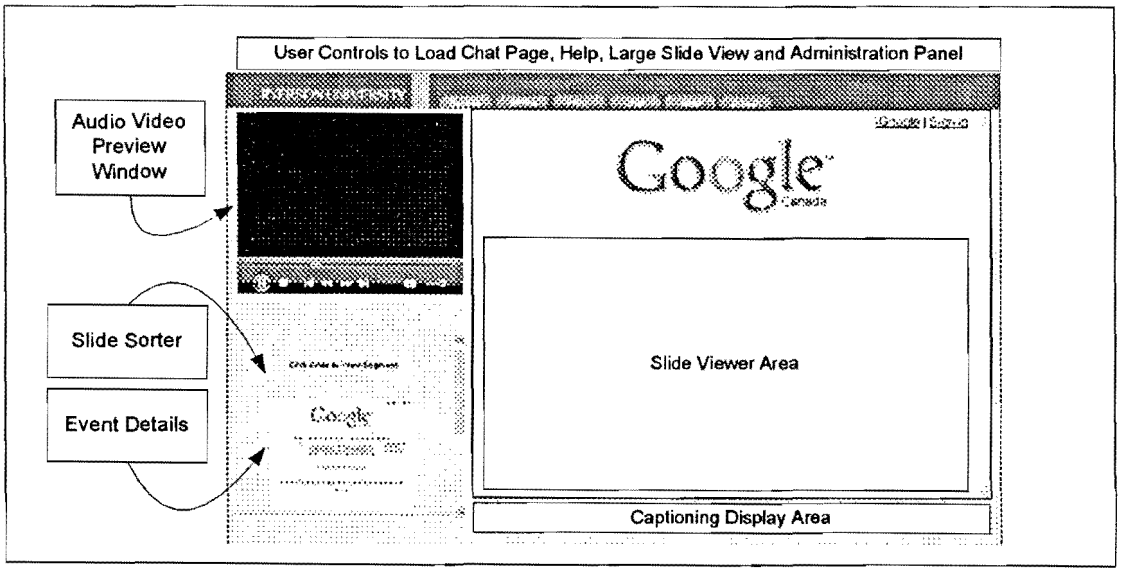

Figure 2: Webcast Viewer User Interface (UI)

Xstreamulator's slide insertion process integrates the video portion of the webcast with captured slides, so that the two media elements arrive at the viewer's computer in a synchronized manner. At the completion of a webcast, the Windows Media stream and its associated slide markers are merged by Xarchiver and Xstreamulator. On-demand viewers can click on a slide to jump to the section of the webcast where the slide appears.

Xstreamulator minimizes the software requirements for webcast viewers and creates a viewing environment that is compatible with both Windows and Macintosh OS/X operating systems. In addition, it was determined through discussions with the beta test group that viewers did not want to be forced to view webcasts with Microsoft Internet Explorer. Therefore, a cross-platform HTML interface was developed that embedded the Windows Media Player plug-in object in a manner compatible with the majority of popular web browsers.[9] At present, users of the OS/X operating system are directed to install Telestream's Flip4Mac plug-in, which replicates Windows Media Player's functionality on OS/X. Flip4Mac is actively supported by Microsoft. Unfortunately, at present, the Flip4Mac plug-in implementation does not support Xstreamulator's slide marker "seeking" feature. It is expected that this issue will be resolved in the next Xstreamulator release. Xstreamulator webcasts can also be viewed on the Linux platform, by installing MPlayer or VLC. Unfortunately, these players do not currently support the automatic display of captured slides. To address this issue an AJAX-based slide loading process that supports Linux is currently in the early stages of development.

The process of implementing a cross-platform playback solution was made more difficult by Media Player's lack of browser scripting support; other than for Internet Explorer browsers on the Windows platform. Specifically, it was not possible to jump to slide positions if the Windows Media file was being viewed on non-Internet Explorer browsers. To resolve this issue, an unconventional approach of dynamically generating the Windows Media plug-in at the desired marker time was implemented in JavaScript. It is expected that this functionality will continue to be improved and extended to other web browsers. Additional enhancements will likely include an increased emphasis on the server-side generation of the viewing environment and the addition of more advanced Active Server Pages and AJAX code.

Microsoft's Silverlight plug-in, a rich media plug-in that is similar to Adobe's Flash Player, is a potential alternative to Windows Media Player. However, the beta version of this plug-in does not currently support live streaming. Preliminary testing has revealed that the Silverlight plug-in greatly improves on-demand webcast performance on OS/X and Windows. Furthermore, Microsoft has enlisted Novell to port Silverlight to the Linux platform. Therefore, whether the ASP/HTML approach is maintained or not, it appears that long-term cross-platform Xstreamulator compatibility is assured.

Chapter 6

System Requirements, Device Support and Audio Level Management

This chapter describes Xstreamulator's software and hardware requirements. Both basic and advanced hardware capabilities are discussed. The application's external content capturing capabilities are examined in detail. The chapter concludes with a discussion of audio level management approaches.

6.1 System Requirements

Xstreamulator currently requires a system running Windows 2000, XP or VISTA. Recent tests have also confirmed that Xstreamulator runs extremely well on Intel-based Macintosh computers using Apple Bootcamp or Parallels Desktop.[10] While there are no specific CPU requirements, Xstreamulator's audio/video encoding processing realistically demands a later generation processor, though users can create audio-only webcasts on more modest (e.g., Pentium III) systems. Users of multicore/multiprocessor based systems can create high-bandwidth or multibitrate Windows Media files.

6.2 Video and Audio Capture Hardware Requirements

In addition to a sufficiently powerful PC, Xstreamulator requires a video and audio capture device. Typically, a USB (1.0 or 2.0) or Firewire (IEEE 1394) web camera and a microphone headset (USB or analog) are acquired by the user. Although these devices are

common in Xstreamulator webcasting systems, the beta testing process has revealed that universally adopted capture devices simply do not exist! As Xstreamulator supports any Windows Driver Model (WDM) or DirectX capture device, many users have not been forced to purchase additional hardware. Furthermore, Xstreamulator supports systems with multiple capture devices. On these systems, Xstreamulator permits the independent assignment of video and external capture devices. In total, Xstreamulator was successfully tested with: web cameras, integrated array microphones, TV tuner cards, digital video cameras, and a variety of dedicated capture cards.[11]

Uniquely, Xstreamulator provides automatic capture device detection. Specifically, Xstreamulator enumerates available capture devices and automatically activates the device most suited to webcasting. A significant amount of programmatic code in Xstreamulator is focused on this advanced functionality. Moreover, Xstreamulator automatically sets audio capture sources to their "microphone" input. Thus, users are not required to perform this often overlooked step. Usually, Xstreamulator's automation device detection subsystem selects the appropriate capture hardware. However, these settings can be overwritten (and saved) if required. In comparison, commercial systems do not provide automatic device detection or automatic audio capture configuration.

6.3 External Capture Device Support

In addition to video capturing, Xstreamulator enables desktop capturing via USB/PCI VGA "frame grabber" devices.[12] The use of VGA acquisition hardware conveniently

circumvents the need to install desktop capturing software on presentation systems. Xstreamulator's VGA capturing functionality was employed extensively in the recording of conferences, lectures, and events. Indeed, the Digital Media Projects Office developed laptop based solutions specifically for the purpose of webcasting presentations.[13] Furthermore, as Xstreamulator is hardware independent, it was possible to deploy VGA capturing functionality to all of Ryerson's mobile encoding systems. Therefore, Xstreamulator's support for inexpensive VGA to USB capture devices greatly lowered the cost of integrated webcasting systems. In comparison, commercial webcasting systems exclusively use costly PCI-based capture hardware, which is reflected in the high price of these systems.

Notably, Xstreamulator supports the recording of VGA content as slides or Windows Media streams. For example, from personal or presentation computers, Xstreamulator can be configured to webcast desktop content as full-motion video. Xstreamulator users can also combine video streams with slides captured from secondary video capture sources (e.g., video cameras). Interestingly, though these devices are not specifically designed for VGA capturing, they can be effectively used for this task. Moreover, Xstreamulator's multi-device and multi-format capture subsystem provides a significant degree of capture flexibility (see Appendix II). For example, it is ideally suited to situations where instructors need to demonstrate 3D objects. Indeed, an instructor at Ryerson's School of Interior Design stated that Xstreamulator's support for secondary capture sources was ideally suited to presenting physical objects and materials.

6.4 Audio Level Management

Xstreamulator's beta test process has revealed that it can be difficult to maintain consistent audio recording levels during a webcast. For optimal results, digital recording levels should consistently fall within the range of -12db to -3db. Audio input levels exceeding 0db are to be avoided, as they cause unpleasant "digital clipping" in the media file. In addition, extremely low levels decrease the signal-to-noise ratio of recordings. To limit these issues, the following audio level management approaches have been implemented in Xstreamulator:

- The addition of audio level metering in Xstreamulator's UI.
- The addition of a 0db (i.e., "clipping") indicator in Xstreamulator's UI.
- The addition of an audio mixer control in Xstreamulator's UI.
- The automatic recall of audio mixer settings.
- The discussion of appropriate recording techniques in Xstreamulator's Help system.
- The use of hardware-based broadcast limiters and compressors.

The aforementioned approaches have helped to improve the consistency and quality of the recording process. Nevertheless, it is anticipated that these approaches will be augmented by sophisticated Automatic Gain Control (AGC) functionality in future Xstreamulator releases. Specifically, Xstreamulator will sample and adjust (i.e., trim) recording levels to achieve optimal results. These adjustments are made on the recording device's input channel. At present, commercial encoding solutions do not provided AGC functionality, as

they assume that audio levels are adjusted by webcast technicians. A DirectX software-based limiter/compressor was also successfully tested in an earlier Xstreamulator prototype. However, while software-based compression/limiting helps in maintaining consistent audio levels, it cannot resolve digital clipping on the input channel of a recording device (e.g., the soundcard microphone input).

Chapter 7

Single-User and Multiple-User Operation

This chapter describes how Xstreamulator can be deployed for single-user or multiple-user operation. Xstreamulator's single-user mode is intended for instructors operating Xstreamulator from a personal computer. Alternatively, Xstreamulator's multiple-user mode is specifically designed for webcasting from classrooms.

7.1 Single-User and Multiple-User Operation Modes

Xstreamulator supports single-user and multiple-user operating modes. The application's single-user mode provides the maximum amount of capture flexibility and operational control to users. In addition, single-user mode enables Xstreamulator users to permanently store streaming profiles and user-specific settings. This is the default operational mode for non-classroom environments. As Xstreamulator activates the appropriate operational mode during its startup process, and dynamically enables/disables mode specific interface elements, it has not been necessary to develop two versions of Xstreamulator or multiple application user interfaces. A "key" file located in the application directory determines Xstreamulator's default operating mode.

Typically, webcast viewers watch a lecture while simultaneously viewing integrated slides. In addition, viewers are encouraged to post questions or comments in

Xstreamulator's Chat interface. This approach enables real-time conversations to occur between students and instructors. To provide a context for the lecture, chat messages are automatically archived with the webcast. The web-based viewer interface is identical in Xstreamulator's single-user and multiple-user configurations.

In either mode, Xstreamulator provides webcasting presets targeted to specialized delivery scenarios. For example, some instructors may prefer to webcast screen content (e.g., application demonstrations) without accompanying video; a delivery mode that Xstreamulator fully supports. If viewers have access to a high-bandwidth connection, Xstreamulator can be configured to composite a video window over captured desktop content. At present, it appears that Xstreamulator is the only application capable of delivering live webcasts using this approach. Xstreamulator is also unique in that it supports the broadcasting of video/audio and screen content at high frame rates. A future version of Xstreamulator will enable administrators to configure (i.e., enable/disable) Xstreamulator's webcasting presets for specific multiple-user environments.

Xstreamulator provides a variety of synchronized capture components that support the acquisition of images and presentation media (Figure 3). The most sophisticated of these components is Xstreamulator's PowerPoint capture module. This module enables instructors to easily embed PowerPoint slides in their webcast. Uniquely, Xstreamulator does not require that PowerPoint files be converted prior to the commencement of the webcast. Indeed, in either mode, Xstreamulator does not require that any content be

converted prior to a live webcast. Xstreamulator also provides a Web Browser module that enables instructors to insert website screen captures and URLs' (i.e., web tours). Finally, Xstreamulator provides basic support for real-time closed captioning. Xstreamulator's captioning module will eventually be extended to include support for captioning hardware (e.g., dictatype systems) and voice-to-text recognition systems. The Images, Screen Capture, Web Browser and PowerPoint capture modules are also available in multiple-user mode. However, users would typically use the External (i.e., VGA) capture module for capturing content in classroom environments. Indeed, this module can be set to start automatically in multiple-user mode.

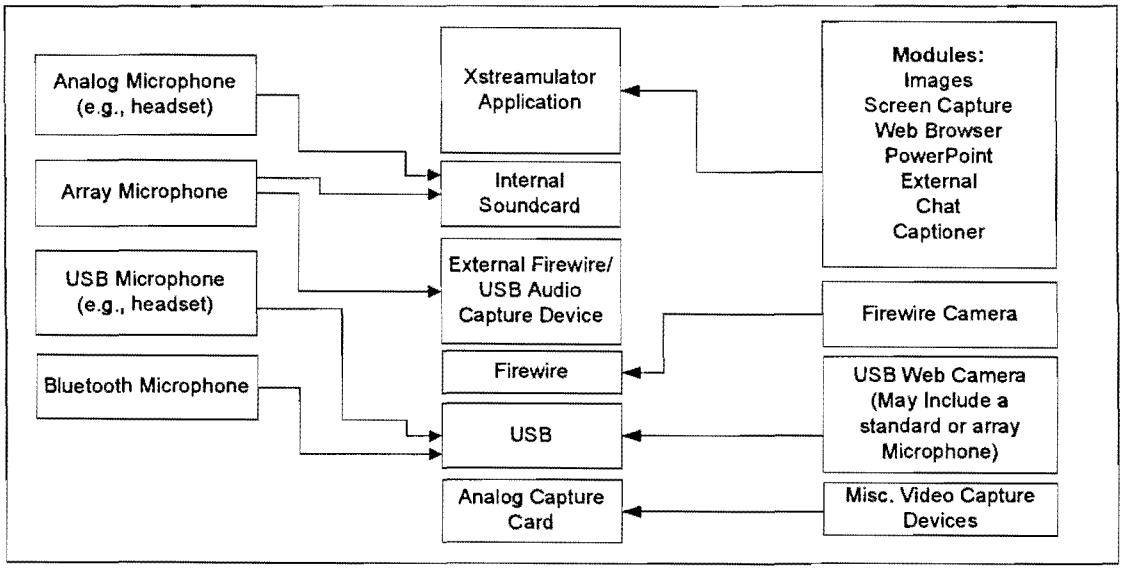

Figure 3: Xstreamulator's Single-User Mode

In single-user mode, instructors typically utilize USB or Firewire cameras to generate audio/video and synchronized slide presentations. Generally, users wear a microphone

equipped USB/analog headset. However, the use of microphone array equipped web cameras is becoming more commonplace. Importantly, the selection of appropriate capturing hardware is left to the instructor's discretion. The variety of devices used by the beta test team demonstrated that Xstreamulator's flexible hardware support was well worth implementing.

In the proposed multiple-user scenario (Figure 4), Xstreamulator would be installed on computers inside of classrooms (e.g., in podiums). Indeed, Xstreamulator's in-podium approach was successfully tested in a number of classrooms at Ryerson. Furthermore, feedback from these trials has led to many significant changes in Xstreamulator's multiple-user functionality. It is even possible to install Xstreamulator on existing presentation computers. However, the declining cost of computing hardware is making this option less compelling.

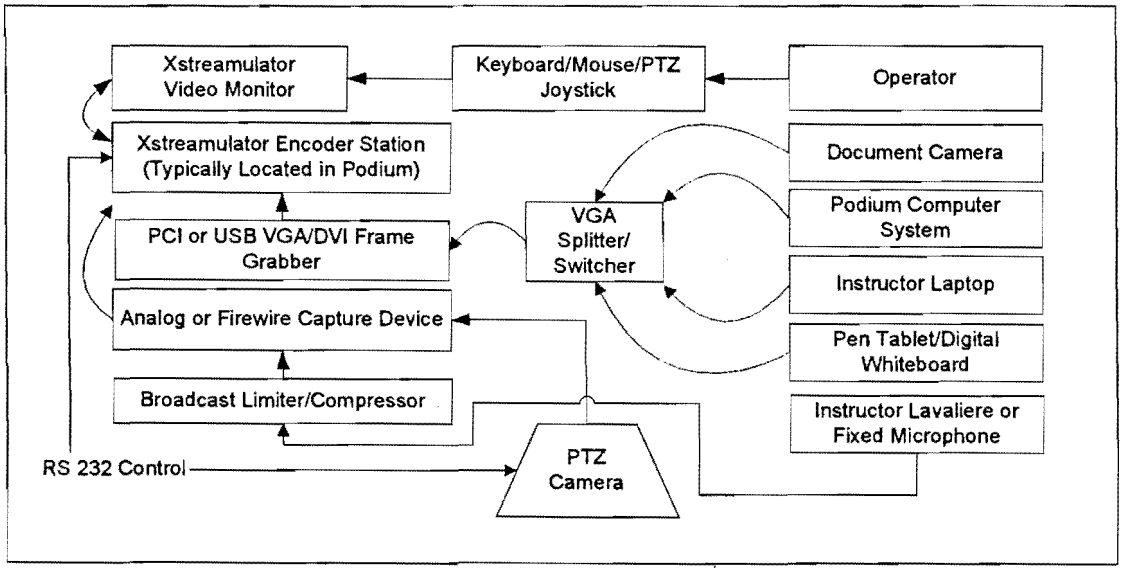

Figure 4: Xstreamulator's Multiple-User (i.e., Classroom) Mode

Alternatively, Xstreamulator could be installed on computers situated in a broadcasting cluster; an approach common in environments with commercial encoding systems. Nevertheless, this implementation requires the installation of costly cables runs (e.g., baluns). Typically, instructors would appoint a student to operate/monitor the encoding process from an LCD monitor in the classroom. The designated operator would control the positioning of a Pan Tilt Zoom (PTZ) camera, monitor and adjusts audio recording levels, oversee the encoding process, and (optionally) manages the chat system directly through Xstreamulator's UI. One option that is currently being considered is the use of a touch-screen interface for managing Xstreamulator webcasts.

Xstreamulator's multiple-user mode is specifically configured for unsecured webcasting environments. For example, users are not allowed to modify hardware settings or to save usernames/passwords on Xstreamulator systems installed in unsecured (i.e., classroom) environments. However, it is anticipated that a future release of Xstreamulator will enable users to store their settings to an external storage device (e.g., a USB key) so that settings could be quickly recalled. Furthermore, Xstreamulator implements webcasting functionality according to its operational mode. For example, Xstreamulator's multiple-user mode requires that instructors provide an automatic stop-time for webcasts. However, a stop-time extension feature has been added to Xstreamulator's UI, for situations where classes exceed their allotted schedule.

Chapter 8

Xstreamulator's User Interface (UI) and Application Modules

This chapter presents Xstreamulator's integrated user interface (UI) model. Xstreamulator's content capturing capabilities are discussed in detail. The major components of Xstreamulator's UI are described, and planned functionality is revealed.

8.1 Xstreamulator's User Interface (UI)

Xstreamulator encapsulates all of its content capturing functionality into an easily navigateable user interface (UI). Critical application components, including the video and audio preview windows, reside in a portion of the UI that cannot be hidden (i.e., the Preview area). Xstreamulator's remaining interface components are located within a collapsible multi-tabbed UI adjacent to the Preview Area. If a user's screen space is limited, the multi-tabbed portion of the UI can be hidden, though Xstreamulator's is typically run in its fully maximized state. Xstreamulator's multi-tabbed UI is efficient, as it enables a large number of buttons and other visual components to be displayed within a small area. Xstreamulator entire UI fits within a 1024X768 resolution screen without compromising the display of an extensive array of buttons, preview windows, dropdown lists and so forth (Figure 5). Development of the application's UI has been driven by feedback from the beta test team and by the direct observation of webcasting sessions. For example, the beta test team requested that Xstreamulator's webcasting functionality be

streamlined to the extent that it did not unduly interfere with the lecturing process. Therefore, Xstreamulator has been designed to operate with a minimal amount of user intervention. The use of a multi-tabbed approach has enabled the application's capturing functionality to be divided into easily managed UI subcomponents, an approach that allows users to focus only on the components they need for their specific webcasting scenario.

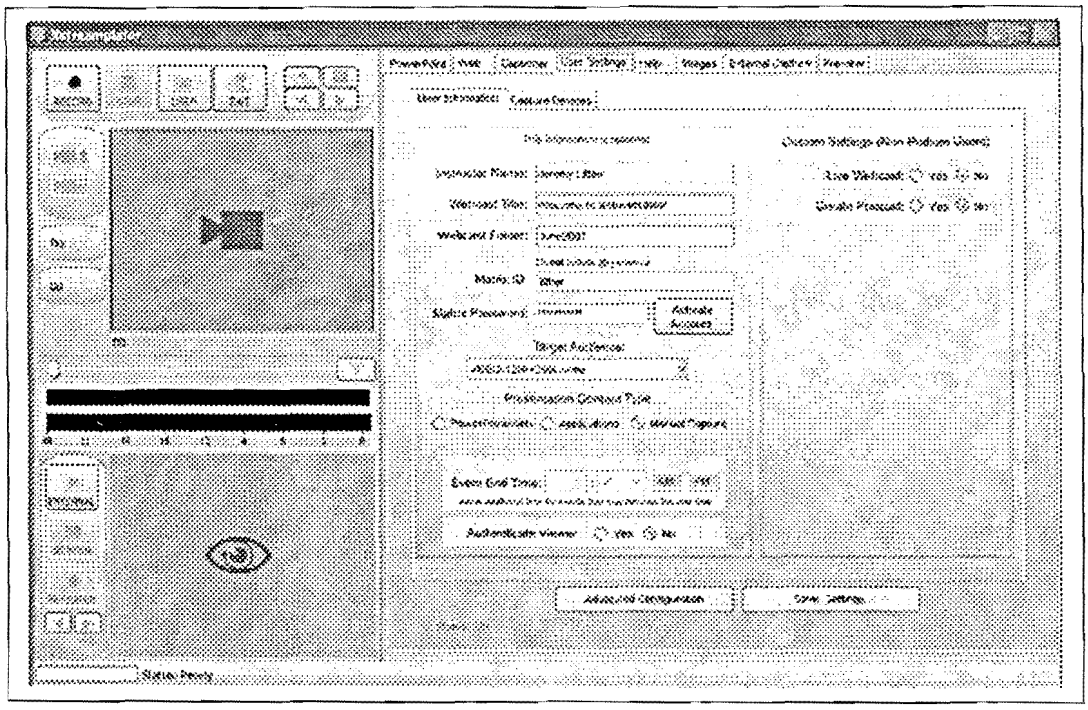

Figure 5: Xstreamulator: Integrated User Interface (UI)

All of Xstreamulator's UI components provide rollover help, and the application contains an embedded HMTL help system. In addition, a reduction in the frequency of user interventions has been achieved by the implementation of automatic error handling

routines and advanced event recovery functionality. The evolution of Xstreamulator's single form UI is discussed in Appendix I.

8.2 Xstreamulator Components

8.2.1 User Settings

Xstreamulator's User Settings component encapsulates user information and hardware (i.e., capture device) settings. Application users are encouraged to save their settings so that they can be automatically recalled in subsequent webcasts. A button on the User Settings screen enables users to easily switch Xstreamulator to a live or non-live webcasting mode. The primary advantage of the non-live webcasting mode is that it supports the creation of webcast content in situations where an Internet connection is not available. Furthermore, Xstreamulator provides an automated FTP uploading component to streamline the uploading process for locally recorded (i.e., non-live) webcasts.

A critical step in the webcasting process is the user's selection of an appropriate webcasting capture mode and bitrate. Members of the beta test team typically used a bandwidth-efficient combined (i.e., video and slide) capture mode. However, while combined capture modes are suited to many instructional scenarios, they are not suited to capturing high-motion screen content or application demonstrations. Therefore, Xstreamulator provides capture modes targeted specifically to these scenarios. Uniquely, Xstreamulator employs editable Windows Media Encoder presets for storing webcast

encoding settings. Xstreamulator's support for editable encoding presets enables advanced users to tailor encoding settings to their specific requirements.

8.2.2 PowerPoint Module

The most frequent feature request from the beta test team was that Xstreamulator be updated to include support for embedded PowerPoint slideshow capturing. While it was possible to import converted PowerPoint presentations in earlier versions of Xstreamulator (i.e., as a series of bitmap images), the process was unwieldy and resulted in sub-optimal slide conversions. The addition of PowerPoint slideshow capture, conversion and display functionality eliminated the requirement for external slideshow conversion (see Appendix II). Furthermore, Xstreamulator's PowerPoint component has since been updated to support complete slideshow playback functionality and slide annotation capabilities. A significant challenge to implementing PowerPoint slideshow capturing functionality is the potential for encountering performance draining slide transitions. Therefore, Xstreamulator's PowerPoint module automatically disables slide transitions during the document loading process.

8.2.3 Images Module

Xstreamulator's Images module enables users to insert bitmap images in their webcast. At present, Xstreamulator supports the capturing/conversion of Graphics Interchange Format (GIF), Windows Bitmap (BMP), Portable Network Graphics (PNG), Windows

Metafile (WMF) and Joint Picture Experts Group (JPEG) images. As web browsers typically support a limited number of bitmap file formats, Xstreamulator automatically converts imported files to the universally supported JPEG file format. In addition, Xstreamulator resizes images to consistent vertical and horizontal dimensions. The Images module has been undergoing constant modification as a result of ongoing feedback from the beta test team. Specifically, beta test team members have requested that the following module enhancements be implemented:

- That the folder of converted PowerPoint slides be accessible by clicking a dedicated button on the Images module.
- That the Images module have dedicated access buttons for the user's "Desktop" and for "My Documents" folders.
- That a folder watch feature be implemented to automatically refresh the file list if new images are added.[14]
- That automatic image capturing functionality be added to the Images module UI.

In addition, predefined status images will be added to the Images module is the next release. For example, Xstreamulator users will be able to set the start time of an on-demand webcast by loading a predefined "starting now" image at the commencement of the webcast; this functionality will address situations where webcasts have delayed start times. At present, the starting slide (i.e., start time) of a webcast must be specified in an XML document. Eventually, predefined status images will be developed for a variety of scenarios (e.g., session breaks, technical difficulties and so forth).

8.2.4 Screen Capture Module

Xstreamulator's Screen Capture module enables users to integrate desktop captures (i.e., screenshots) in their webcast. Interestingly, Xstreamulator's screen capture functionality was conceived as a means of enabling users to easily grab desktop/application windows "on-the-fly". However, the beta test team often used the Xstreamulator's Screen Capture module in preference to more appropriate capture modules! Thus, the popularity of screen capturing indicates that a more sophisticated approach should be investigated. One of the challenges in implementing the Screen Capture module was the necessity to hide the Xstreamulator UI during the capturing process. To accomplish this, Xstreamulator briefly sets its UI transparency level to zero (i.e., invisible).

8.2.5 Captioner Module

Xstreamulator's Captioner module enables users to provide closed-captioned webcasts. The module employs a simplified text editor interface and supports cut-copy-paste operations. At present, the module inserts captions directly in the Windows Media file (i.e., in the media file's captioning track), though support for HTML "wrapped" captions is being investigated. Significantly, the "wrapping" of captions in an HTML document would extend captioning support to non-Microsoft Internet Explorer browsers. Furthermore, Xstreamulator will eventually support captioning devices (e.g., stenotype chorded keyboards) and speech-to-text recognition systems.[15] It is expected that this

technology will make live captioning much more efficient and ubiquitous. Presently, Xstreamulator generates an XML-based Synchronized Multimedia Integration Language (SMIL) file at the completion of the webcast. SMIL files can be easily synchronized with a Windows Media stream, particularly in situations where the stream is viewed outside of Xstreamulator's ASP/HTML viewing interface. Additionally, the generation of SMIL captioning files enables captioning errors to be easily corrected.

Accessibility enhancements are currently being added[16] to Xstreamulator's application UI. Thus, it hoped that users with accessibility issues will be able to operate Xstreamulator. Ryerson's Centre for Learning Technologies has been involved in the Xstreamulator project from its early prototype stage, and it is hoped that this department will continue to participate in the application's future development. At present, the Centre for Learning Technologies is utilizing Xstreamulator prototypes to test a descriptive captioning system. In general, commercial encoding systems include limited support for closed captioning or assume that the content producer will add captions during a post-production process. Adding captions during post-production is an incredibly time consuming and expensive process. Furthermore, it does not address the needs of a live audience. In contrast, Xstreamulator's approach to captioning is focused on enabling the real-time insertion of captioning text.

8.2.6 Web Browser Module

Xstreamulator's Web Browser module enables users to insert screenshots of websites in their webcast. To provide this functionality, Xstreamulator embeds a Microsoft Internet Explorer control in the tabbed portion of Xstreamulator's UI. In addition to capturing images of web pages, Xstreamulator attaches a website hyperlink to each captured slide. Webcast viewers click captured website slides to open the attached URL in a pop-up browser window. An experimental version of Xstreamulator was developed to directly "push" website hyperlinks to a frame in the viewer interface. Unfortunately, live testing of the hyperlink "push" feature revealed that a small number of websites are specifically coded in JavaScript to override any frameset that contains them. This scenario causes Xstreamulator's viewer interface to be overwritten. As it is impossible to predict whether a website will exhibit this behaviour, it is has been determined that the current approach is at present the safest alternative. Nevertheless, the "push" approach is worth pursuing, as it was extremely well received by the beta test team. Indeed, one beta tester became so engrossed with the ability to "push" hyperlinks that they forgot that they were also recording audio and video!

At present, Xstreamulator's Web Browser module captures a predefined region of a webpage (i.e., the area visible in the Web Browser preview window), though users can employ scrolling to capture the off-screen portion of a webpage. One of the challenges faced in the design of the Web Browser module was the requirement to support basic browsing functionality. Ultimately, the Web Browser module was equipped with basic

controls, including: page back, page forward, refresh and homepage buttons. A future release of Xstreamulator will enable users to access their Microsoft Internet Explorer bookmark list.

8.2.7 External Capture Module

Xstreamulator supports the capturing of bitmap images from external sources. While typical capture sources include USB/PCI based VGA "frame grabbers", any capture device that is supported by Microsoft DirectX or the Windows Driver Model (WDM) can be used. When users upgrade their existing capture technology, older capture devices can be repurposed for external capturing duties. At present, the external capturing process requires users to manually initiate external captures. As this process is under user control, the capturing process is extremely efficient. However, the process of initiating manual captures is laborious. Consequently, Xstreamulator will offer automatic capturing functionality in a future release. Support for automatic capturing will be particularly beneficial in classroom environments, as it will relieve webcast operators from this repetitive task. In addition, automated capturing from document cameras and Tablet PC's will enable instructors to employ presentation technologies that are inherently more "interactive" than PowerPoint presentations (Anderson, et al., 2000, p. 3).

The External Capture module provides functionality for connecting and disconnecting external devices, and for refreshing the capture engine when the resolution of the capture device changes. For example, when using a USB/PCI frame grabber devices presenters

may change their display resolution. If required, a "refresh" function in the External Capture module can be invoked to adjust the capture "engine" to the new resolution. In addition, users can adjust capture device settings (e.g., brightness, contrast and so forth) by opening a capture device properties panel.

8.2.8 Webcast Preview Module

Although Xstreamulator automatically notifies users in the event of a webcast failure, beta testers have requested that the application display a live preview of webcasts in-progress (i.e., a confidence monitor). Prior to the implementation of the Webcast Preview module, many instructors loaded the ASP/HTML viewer interface to assure themselves that their webcast was being received by the viewing audience. This approach was not ideal, as the viewer interface consumed a large portion of the user's screen space. Ultimately, a Windows Media Player control was added to the multi-tabbed control in Xstreamulator's UI. The Webcast Preview module has since been configured to display both live webcasts and previews of locally recorded content. Recently, a slide preview window was added to the Webcast Preview module UI. Thus, the Webcast Preview module completely replicates the ASP/HTML viewer interface's functionality.

8.2.9 Last Slide Preview Window

Observation of Xstreamulator's use by the beta test group revealed that users often forget if a slide had been inserted into the webcast. To resolve this issue, a Slide Preview

window was added to Xstreamulator's UI. Any slide captured via the External, Screen Capture, Images, Web Browser or PowerPoint capture modules is automatically displayed in the Last Slide Preview area. Additionally, to make optimal use of the application's UI, the Last Slide Preview window is employed to display event status messages.

8.2.10 Chat Module

The success of Xstreamulator's Web Browser module established that an embedded Internet Explorer browser approach could be implemented in other areas. For example, earlier versions of Xstreamulator's UI did not include a "front-end" interface for the application's Chat system. Instead, users were required to load the Chat system in a separate web browser window. This approach was unwieldy. Ultimately, it was determined that embedding the Chat module directly in Xstreamulator's UI would be the most efficient solution. Moreover, integrating Xstreamulator's Chat system made the interface compatible with the application's slide capturing subsystem. For example, users can now capture relevant portions of a chat session as bitmap slides. Furthermore, this implementation will eventually permit the synchronization of chat sessions with streaming media files.

8.3 Proposed Modules

As Figure 6 demonstrates, Xstreamulator's implementation of capture modules and acquisition functionality is extremely comprehensive. However, it is anticipated that two

additional modules will be implemented before Xstreamulator's feature set is complete.

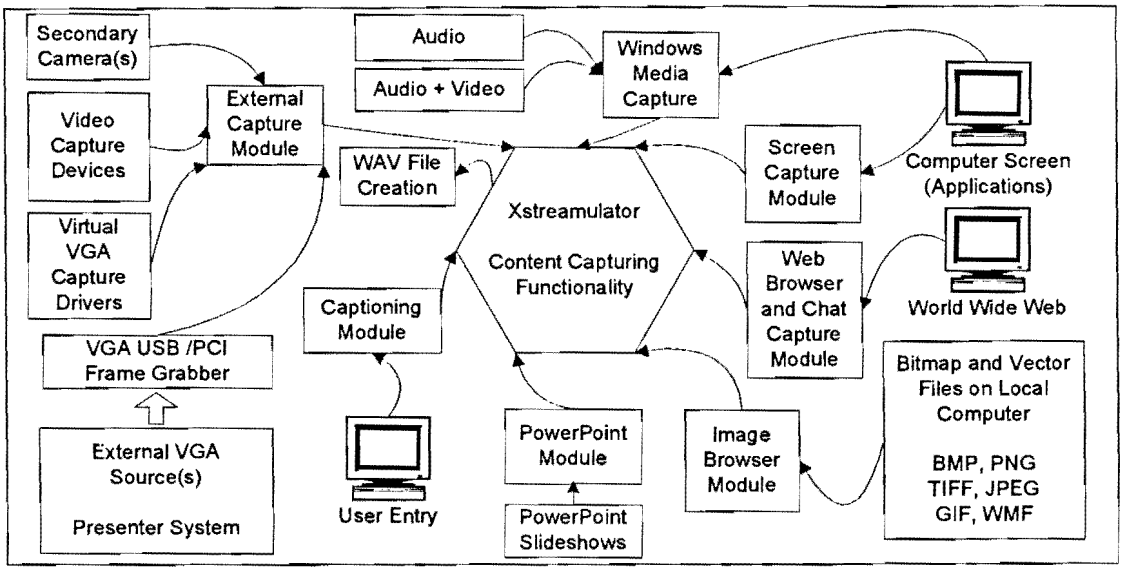

Figure 6: Xstreamulator: Integrated Capture Modules and Functionality

A whiteboard module is planned for future inclusion in Xstreamulator's UI. This module will enable instructors to annotate captured slides. In addition, the module will replicate the functionality of a physical whiteboard. The benefits of handwriting support is well established, as "many manual systems (including overhead projectors and document cameras) support high quality handwriting over slides, enabling the instructor to augment prepared materials with supplemental text or diagramming" (Anderson, et al., 2000, p. 2). The decision to include a whiteboard module in Xstreamulator is a direct result of feedback from the beta test team. Team members have indicated that a whiteboard module would better replicate the methods they use in typical classroom environments. However, it has not been decided if the whiteboard module will be implemented as a distinct UI

component or as an integrated component of existing UI modules. In either case, the visual design of the whiteboard module and its associated functionality will be driven by feedback from the beta test team.

An integrated PTZ camera control module will also be added to Xstreamulator's UI. This module will be unique, as existing commercial solutions assume that camera control is an independent function of the webcasting process. For example, in commercial solutions the process of adjusting camera-subject positioning (i.e., a camera's pan, tilt and zoom settings) is typically relegated to costly hardware-based (i.e., joystick) control surfaces or integrated into a third-party systems (e.g., Crestron systems). Thus, the cost of providing camera control solutions is a barrier to their implementation within academic settings. The alternative solution, currently being investigated, eliminates the requirement for physical control surfaces or podium integration. Instead, this functionality is replicated directly within Xstreamulator's UI. In this scenario, camera settings are adjusted by the Xstreamulator user via a dedicated UI module. The proposed Pan-Tilt-Zoom (PTZ) module would require a physical connection to the target camera. However, as this entails the installation of an inexpensive serial control cable (e.g., RS232), it is a less costly solution than current alternatives. Moreover, a software based camera control solution would be far more flexible than hardware-based implementations.

The software-based approach could potentially be used in developing custom control interfaces for multiple webcasting environments. For example, Xstreamulator's camera

control system could be displayed as an overlay of the classroom layout. Webcast operators would click on the portion of the classroom schematic that matched the physical location of the instructor. An inexpensive "entertainment" joystick could be added to the environment if operators find precise camera positioning too tiring. Indeed, an entirely kiosk based approach to webcasting and PTZ camera control is worthy of further investigation.

Chapter 9

Conclusions and Future Work

9.1 Future Work

Development of the application framework will focus on improving Xstreamulator's functionality and usability. Ultimately, it is hoped that Xstreamulator can be developed to a point where the entire webcasting process is fully automated. To accomplish this goal, further work is required in the areas of usability, audio management and PTZ camera operation. It is anticipated that the development of automatic audio leveling and simplified PTZ camera control will greatly improve the results of webcast sessions. One of the more exciting enhancements, currently in the prototype development stage, is the addition of automatic external content capturing functionality. Ideally, Xstreamulator will support the automatic insertion of VGA content into webcast streams. For efficiency reasons, the full implementation of automatic external capturing will require the development of a heuristic capture process. Ideally, the proposed system will capture the minimal number of slides possible, while simultaneously obtaining enough content to accurately replicate presentation content.

As Xstreamulator can be installed on dedicated computer systems within classroom settings, there is a potential for remote operation. For example, the functionality employed in Xstreamulator to Xarchiver communication can be used to remotely control

Xstreamulator "capture stations". Indeed, webcasting sessions could be initiated from a web-based scheduling system or from "light-weight" applications on classroom presentation systems. It is even conceivable that the entire encoding process could be initiated and monitored directly by the lecturer, with the encoding and content capturing processes relegated to a computer system located potentially anywhere on campus. While the full implementation of this method of delivery will require sophisticated camera tracking technologies and audio management hardware, it will completely eliminate the need for dedicated webcast operators. Fortunately, the remote operation model is relatively easy to implement, as the framework's webcasting functionality is contained entirely within Xstreamulator.

It is expected that Xstreamulator will continue to be developed through a participatory design process. However, it is also likely that the development methodology will be expanded to encompass new approaches. In particular, Think-Aloud/Talk-Aloud protocols, focus group and survey methodologies will likely be employed as the Xstreamulator user group expands beyond manageable levels. Nevertheless, the PD process has yielded benefits that could not be easily duplicated by the adoption of other research approaches. Indeed, the encouragement of the beta test team has been critical to the success of the project and the process will be maintained, albeit in a modified form, well into the future.

9.2 Conclusion

The Xstreamulator project demonstrates that an affordable in-house webcasting solution can be successfully implemented within academic environments. However, the development process must address the needs of the academic community and the institution as a whole. The use of participatory design (PD) in the application's development process has significantly contributed to the project's successful outcome. Specifically, instructors, students, and staff at Ryerson University have identified numerous areas of potential improvement. Furthermore, it is unlikely that these would have been implemented in a less flexible development process. It is expected that this approach would yield similar results in other academic institutions. However, practitioners wishing to employ PD research should be prepared to encounter a variety of institutional obstacles, and be committed to encouraging participation from all members of the community. Fortunately, these challenges can be significantly offset by the enthusiasm of beta test participants.

From a technical standpoint it should not be assumed that the development of rich media applications is beyond the capabilities of casual programmers or high-level programming environments. Indeed, the use of high-level languages and RAD development encourages experimentation. Typically, application development is far more arduous with low-level programming tools. Therefore, development teams should focus on addressing end-user needs, not on acquiring low-level programming skills. To maintain interest, development teams should release functional prototypes (using RAD approaches)

to the beta test community as rapidly as possible. Application features can always be refined in subsequent updates. Furthermore, to minimize development and implementation costs, rich media webcasting solutions should adopt open source tools and flexible hardware requirements. However, developers should be prepared to use commercial components if necessary. In general, programming efforts should be focused on the achievement of user-centric goals, not on the implementation of complex delivery solutions. The project's success is the result of a user-centered approach to application development. It is also the result of a determined effort to increase the penetration of webcasting technologies in traditional lecture delivery environments. Hopefully, academic institutions will apply this approach to new and exciting rich media applications.

References:

Anderson, R., Anderson, R. (Ruth), VanDeGrift, T., Wolfman, S., & Yasuhara, K. (2004). Experiences with a Tablet PC Based Lecture Presentation System in Computer Science Courses. *Proceedings of SIGCSE'04: the 35th SIGCSE technical symposium on Computer Science Education*, 56–60.

Anderson, J., Nahella, A., Douther, C., & Mervyn, A. (2001) Presence and Usability in Shared Space Virtual Conferencing: A Participatory Design Study. CyberPsychology, 4(2), 287-305.

Barab, S., Kling, R. & Gray, J. (2004). Designing Virtual Communities in the Service of Learning. Cambridge: Cambridge University Press.

Butler, T., Fitzgerald, B. (1997). A Case Study of User Participation in the Information Systems Development Process. *International Conference on Information Systems,* December 14 – 17, 411 – 426.

O'Connor, C., Fitzpatrick, G., Buchannan-Dick, M. & McKeown, J. (2006). Exploratory prototypes for video: interpreting PD for a complexly disabled participant. NordiCHI 2006, 14-18. 232-241.

Damordan, L. (1996). User involvement in the systems design process - a practical guide for users. *Behaviour & Information Technology*, 15.6, 363-377.

Dickey, M. (2003). Teaching in 3D: Pedagogical Affordances and Constraints of 3D Virtual Worlds for Synchronous Distance Learning. *Distance Education*, 24.1, 105-121.

Jaffee, D. (2003). Virtual Transformation: Web-Based Technology and Pedagogical Change. *Teaching Sociology,* 31.2, 227-236.

Joyes, G., & Scott, R. (2000). A reflection on a collaborative process of courseware development. *Information Services & Use*, 73-82.

Lee, D. (2006). Streamulator: A New Approach to Online Office Hours. *Information Visualization*, 7, 605- 608.

Littlejohn, S., Foss, K. (2005). Theories of Human Communication. Toronto: Thomson.

O'Day. V., Bobrow, D. & Shirley, M. (1998). Network Community Design: A Social-Technical Design Circle. *Computer Supported Cooperative Work 7*, 315–337.

Ostrow, L., DiMaria-Ghalili, R., (2005). Distance Education for Graduate Nursing: One State School's Experience. *Journal of Nursing Education,* 44(1), 5-10.

Rankin, K., Baecker, R. & Wolf, P. (2004). ePresence: An Open Source Interactive Webcasting and Archiving System for eLearning. *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education,* 2888-2893.

Roberts, V., Fels, D. (2004). Methods for inclusion: Employing think aloud protocols in software usability studies with individuals who are deaf. *International Journal of Human-Computer Studies,* 64, 489-501.

Teemant, A., Smith. M., Pinnegar, S., & Egan, M. (2005). Modeling Sociocultural Pedagogy in Distance Education. *Teachers College Record* 107.8, 1675-1698.

Wheelan, Susan. The Handbook of Group Research and Practice. Thousand Oaks: Sage Publications, 2005.

Appendix I: Software Framework and User Interface (UI) Development

This appendix provides a detailed description of Xstreamulator's development process from early prototypes to the current release. Specifically, the appendix discusses how the adoption of the Visual Basic.NET (VB.NET) development environment and contributions from the beta test team led to the Xstreamulator's evolution from a multi-form UI to an integrated single-form UI. The appendix reveals that low level application development environments are not required for the successful development of webcast delivery applications.

The Evolution of Xstreamulator's Software Development Framework

Xstreamulator was initially developed in the Microsoft Visual Basic 6.0 (VB) programming environment. However, it eventually became evident that a more extensible programming environment was required to achieve the project's goals. For example, early Xstreamulator prototypes were dependent on the "predefined" Windows Media Encoder UI for displaying video and audio mixer previews. This implementation required that the Windows Media Encoder UI be attached to the bottom of Xstreamulator's application window (Figure 7). In addition to being aesthetically unappealing, the disjointed UI caused application performance issues.[17] Moreover, the original Visual Basic 6.0 design made Xstreamulator appear "bolted together", which did not inspire confidence in the beta test team! Therefore, it was decided that the "predefined" Windows Media Encoder UI design

would be supplanted by a fully integrated UI that containing embedded video and audio previewing components (Figure 5). This integration process necessitated the adoption of a new programming environment.

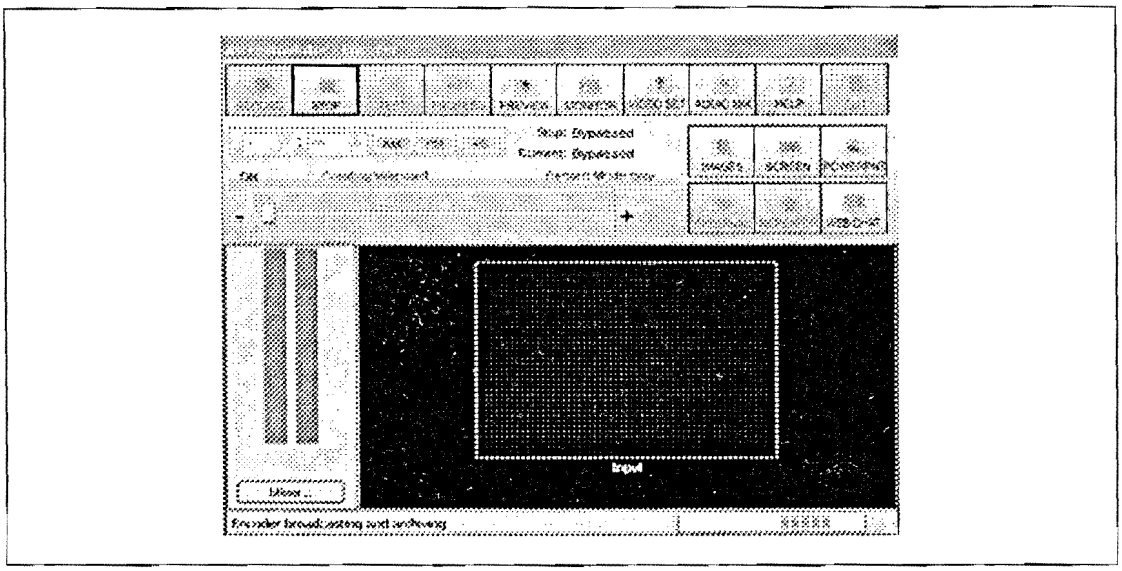

Figure 7: The Non-Integrated User Interface (UI) Version of Xstreamulator

Interoperability requirements for Windows Media Encoder and the Windows Media Encoder Software Development Kit (SDK) precluded consideration of programming environments lacking this support. As a consequence, the candidate programming languages were shortlisted to Visual C#, Visual C++ and Visual Basic.NET (VB.NET). It was confirmed that Visual C# and Visual C++ provided full support for the Windows Media Encoder libraries. However, it was also determined that Xstreamulator's conversion to a completely new codebase would have substantially delayed progress on the project. For example, as Visual Basic 6.0 projects cannot be converted to Visual C# or

Visual C++, a complete rewriting of prototype code, modules and application forms would have been required. Moreover, this time-consuming undertaking would have inevitably introduced numerous coding/logic errors. Therefore, Visual Basic.NET (VB.NET) was selected as the most suitable application development platform. At present Xstreamulator is being developed in the Visual Basic 2008 (VB.NET 2008) programming environment.

The Development of the Single Form Application User Interface (UI)

The non-integrated prototype's content capturing functionality was distributed to independent application windows (i.e., forms).[18] Therefore, Xstreamulator's UI became increasingly unwieldy as new capturing components were added; the constant repositioning and opening/closing of application hampered usability and caused CPU spikes. Furthermore, users tended to open all of the application's forms to avoid being distracted during the webcasting process. Consequently, it was determined that a new UI design was required. A reexamination of Xstreamulator's UI model was undertaken.

A variety of potential UI designs were discussed with the beta test team, including those based on a multiple document interface (MDI) model.[19] However, while an MDI based UI would have greatly reduced screen space requirements, it would not have resolved the issue of constant window repositioning and form opening/closing. Ultimately, the most appropriate design was determined to be a single form UI with a multi-function tab control component. Thus, a tab control component was developed to contain all of

Xstreamulator's capture modules (Figure 5). As the tab control component encapsulated all of the application's content capturing interfaces (i.e., modules), it became possible to preload Xstreamulator's capture interface during the application's loading process. Though component preloading increased application load times slightly, it significantly improved Xstreamulator's usability and post-load responsiveness. To further improve Xstreamulator's usability, UI expand/contract functions were added.[20] This functionality, which was well received by the beta test team, enabled users to hide the tab control interface in situations where the capturing components were not required (e.g., video only webcasts). At present, Xstreamulator's UI development efforts are focused on reducing user prompts and warning messages, so that users are isolated from the "mechanics" of the webcasting process as much as possible. For example, in the current Xstreamulator release, users can initiate a live webcasting session with a single button click.[21] In summary, though Xstreamulator's conversion to a fully integrated UI required a time consuming conversion of "predefined" encoder subroutines and a significant graphical redesign, the benefits of this conversion were proven to be well worth the substantial effort.

Implementation of the Single Form User Interface (UI) Model

One of the consequences of reducing the Xstreamulator's UI to a single form is that it became more difficult to prototype new functionality, as even the smallest optimizations/enhancements required debugging an increasingly complex main application

form. This issue was partially addressed by prototyping new functionality in separate application workspaces (i.e., standalone projects). Significantly, it was discovered that this approach generated applications that could be re-purposed for other projects. However, while the external prototyping approach helped to streamline the application development process, it could not address the predicament of managing the application's growing and fragmented codebase.[22] As the project was already successfully employing text-based code modules for image capturing and global variable declarations,[23] a decision was made to move the previously form-based code segments (e.g., functions and subroutines) to freestanding code modules. Ultimately, the concentration of the code base in modules proved worthwhile, as the project became easier to manage and more extensible.[24] The long-term benefit of this approach is that it will be easier to repurpose Xstreamulator code in future projects. Eventually, it is expected that Xstreamulator's entire code base will be contained within portable text-based modules.

Optimization of Media Encoding and Image Capturing

In theory, the use of low level programming environments, including Visual C# and Visual C++, could have yielded improved application performance. However, the primary performance constraint in Xstreamulator is the processing demands of the Windows Media Encoder subsystem. As the Windows Media Encoder "engine" is highly optimized and multiprocessor aware, it was determined that the benefits of implementing low level languages (i.e., C# or C++) would be marginal at best. Furthermore, Xstreamulator's development process necessitated the implementation of a Rapid Application

Development (RAD) model. While C#/C++ programming environments are considered RAD toolkits, they are in some cases less suited RAD tasks than Visual Basic, particularly when they are employed by casual programmers. Ultimately, the use of the RAD capable Visual Basic.NET environment in combination with low-level components, dynamic link libraries (DLL's), and hand-optimized functions, achieved the project's performance goals without introducing an arduous development process.

Testing confirmed that Xstreamulator's performance was influenced more by the capabilities of a user's Central Processing Unit (CPU[25]) than the granularity of the application's programming language. For example, greater performance gains were achieved by running Xstreamulator on systems with multi-core, hyper-threaded or multiprocessor CPU architectures than could have been achieved through the use of low-level algorithms. However, as there was no guarantee that users would have access to a multi-core system, Xstreamulator was required to run reliably on single-CPU configurations. The situation was exacerbated by the requirement that single-core CPU systems remain responsive during webcasting sessions. This challenge was met by meticulously auditing Xstreamulator's codebase to optimize functions likely to introduce CPU spikes.[26] Ultimately, the process of codebase optimization (e.g., load balancing) represented one of the most time-consuming and complex facets of Xstreamulator's development.

Potentially, through additional application code, Xstreamulator's could dynamically

adjust encoding and image processing demands to maximize performance on an assortment of computer platforms. On slower systems, Xstreamulator would trade-off capture quality for application stability. While the aforementioned approach has not been implemented, Xstreamulator does provide a significant degree of performance flexibility through its native support for editable encoder profiles. However, while Xstreamulator's encoding process is isolated from CPU overloading, it is not possible to guarantee that the webcasting process will survive system or network failures.[27] Thus, additional "recovery" functionality has been implemented to deal with these rare but severe scenarios.[28] Interestingly, the beta testing process revealed that the efficiency of audio and video capture hardware and the degree of PC optimization influenced CPU loads significantly. For example, Firewire video capture devices place lower demands on the Windows Media Encoder engine than USB devices,[29] and "optimized" systems performed better than those with burdened with background tasks.[30]

Migrating Xstreamulator's development environment from VB 6.0 to VB.NET provided an opportunity to implement application threading functionality. With application threading it is possible to prioritize and distribute (e.g., offload) encoding/image capturing processes. For example, CPU intensive image capturing activities can be distributed to a second processor core to isolate the encoding process from CPU spikes. Therefore, VB.NET's threading support will become increasingly beneficial as multi-core and hyper-threaded processors become commonplace. At present, threading support is being implemented across Xstreamulator's entire codebase. Finally, it was decided that the VB

6.0 to VB.NET codebase conversion would include a comprehensive code optimization process. This process entailed replacing numerous VB 6.0 code segments with more efficient routines based on the .NET Framework. The conversion process also included the addition of error trapping functionality to the converted codebase. These procedures greatly improved application stability and performance.

One of the barriers to implementing rich media applications in Visual Basic is the pervasive opinion that the language is not efficient enough for processing images and video. The Xstreamulator project clearly establishes that this assumption is incorrect. Unfortunately, written and on-line documentation on media programming typically assume that developers are using low-level programming environments (e.g., C++/C#). Therefore, it is hoped that the Xstreamulator project will encourage Visual Basic programmers to development and document applications with a rich media focus. Hopefully, the myth that a low-level language programming approach is needed to achieve worthwhile results has been dispelled.

Application Distribution, Updates and User Persistence

Xstreamulator's application updates were distributed from a centralized (i.e., Internet) location. In addition to providing the most recent Xstreamulator builds, the download page provided beta testers with links to the prerequisite library (i.e., the NET 2.0 Framework runtime) and the Windows Media Encoder installation package. Though the process of updating existing Xstreamulator installations should have been relatively easy,

it ultimately became a major programming challenge. While VB.NET provided a class[31] to persist Xstreamulator's settings, no consistently reliable approach was found to guarantee that these settings would be saved during the application updating process. The source of this problem appears to be VB.NET's application installer framework. Specifically, updates were not being recognized as such, and instead were being installed as completely new application instances. A customized method for storing user settings was recently implemented to address this issue. This approach stores Xstreamulator's user and distribution settings in independent XML documents. It is expected that future iterations of Xstreamulator will utilize encryption (i.e., based on the System.Security.Cryptography .NET class) to protect sensitive user settings. Importantly, implementing XML for storing user/distribution settings enables Xstreamulator's functionality to be de-localized. As a result, Xstreamulator can be easily configured to work with webcasting systems in other universities.

Appendix II: Core Components and Application Architecture

This appendix provides a detailed description of Xstreamulator's core application libraries. The selection and implementation of these libraries is discussed in relation to the overarching goal of maximizing flexibility, performance and stability. Ultimately, a mix of commercial and open source libraries were required to implement the application's objectives.

Implementation of the Windows Media Encoder Libraries

The primary encoding component in Xstreamulator is WMEncoderLib, a dynamic link library (DLL). WMEncoderLib is registered as a Component Object Model (COM) object during the installation of Microsoft's Windows Media Encoder. Xstreamulator also employs "secondary" Windows Media Encoder libraries, including WMPREVIEWLib for video previewing and WMEncBasicEditLib for post-webcast marker insertion (Figure 8). Therefore, Xstreamulator cannot be used on systems without Windows Media Encoder. It is worth noting that Windows Media Encoder implements functionality not available with WMEncoderLib and its associated dynamic link libraries.[32] Therefore, one must assume that Windows Media Encoder employs additional code to extend the application's functionality beyond what is available to .NET programming environments. These routines are rumored to be DirectX based functions implemented in C++.

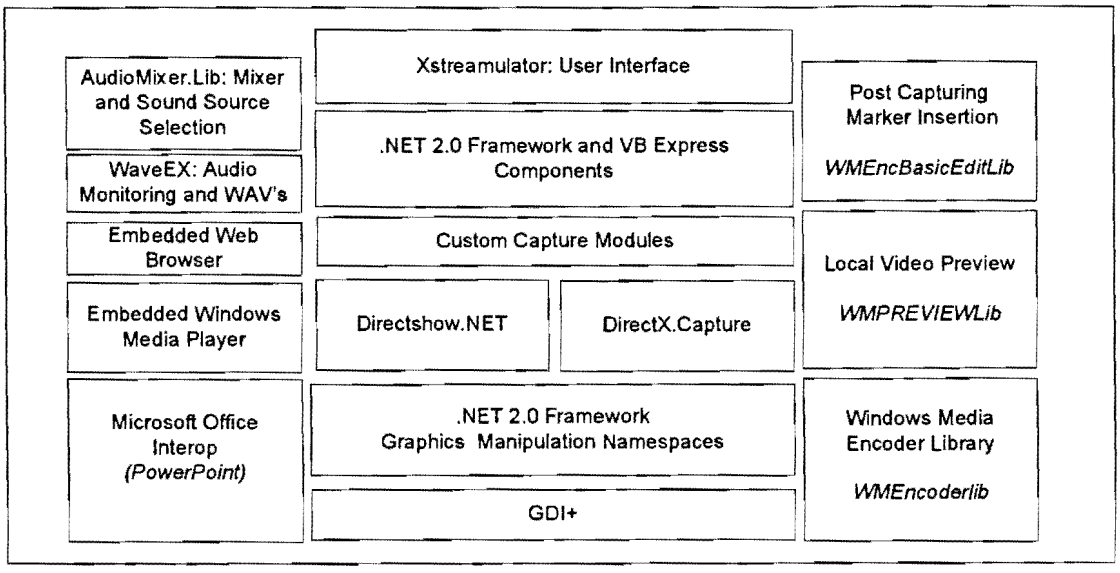

Figure 8: Xstreamulator: Application Architecture

Though the programming for COM libraries is generally consistent between Visual Basic, C# and C++, there are a few significant variations that reflect the inherent capabilities of the aforementioned languages. The C++ programming language is unquestionably the most flexible option for Windows Media Encoder based development, as developers are capable of implementing all of the functions supported by WMEncoderLib, and its associated libraries. However, the complexity of programming in C++ is a significant barrier to many developers. Therefore, it is beneficial that Microsoft's Windows Media Encoder 9.0 COM libraries (Figure 8) support the development of fully functional encoding applications in VB.NET. Indeed, the Windows Media framework's support for high level programming languages was determined to be unique.[33]

Although the Windows Media Encoder Software Development Kit (SDK) is not

strictly required for programmatic interaction with the WMEncoderLib library, it usefully describes the core components of the Windows Media Encoder architecture, and identifies the encoding objects (i.e., the "Object Model") that are available for programmatic control. In addition to exposing the Object Model, which is a hierarchical framework of objects branching from WMEncoderlib, the SDK provides code samples for interacting with WMEncoderLib and its dependent components (denoted as "Objects" in Visual Basic[34]). Therefore, the Windows Media Encoder SDK is a nearly essential "road map" for using WMEncoderlib and for Windows Media development in general. Nevertheless, one of the project's major challenges was the process of modifying SDK examples to fit the specific requirements of the Xstreamulator approach. For example, many of the SDK code samples assume that the developer is using the predefined Windows Media Encoder UI, which is referenced/displayed in VB.NET through a COM component called MSPropshell. While this control was used in early Xstreamulator prototypes, the adoption of the integrated UI approach rendered this control and its associated SDK documentation unusable.

Implementation of the Image Processing and Capture Libraries

In addition to capturing Windows Media "streams", Xstreamulator was required to efficiently capture and process bitmap images sourced from the application's integrated capture modules (e.g., the PowerPoint, Images, External and Web Browser modules). Earlier version of Xstreamulator used a third party image processing library to perform

this functionality. However, the use of this library introduced significant performance issues. Ultimately, the migration to VB.NET enabled a more efficient image processing approach, based on functionality provided by the .NET 2.0 Framework. Specifically, the .NET 2.0 Framework provided a number of optimized functions for image exporting (to the JPEG file format), display and thumbnail generation. However, testing revealed that the .NET 2.0 libraries were not efficient enough to be viable for capturing raw image data. Fortunately, the VB.NET environment provided support for an alternative image capturing and processing approach. Currently, Xstreamulator's image capturing functionality is based on the highly optimized libraries provided by Microsoft's Graphics Device Interface (GDI+) subsystem.[35] The GDI+ interface is a compatibility layer that enables the transfer of pixel-based content from a user's graphics card (i.e. their desktop) to a programmatically addressable memory array.[36] For example, in the case of screen capturing, a GDI+ "Bit-Blitting" process transfers screen data into a memory array. The memory array is then exported as a JPEG image file. As GDI+ image capturing functions are optimized to rapidly move raw pixel data to and from memory, the performance of the library is unsurpassed, even in situations where these functions are implemented in a high-level programming language. Indeed, the GDI+ based programmatic code used in VB.NET is very similar to the same subroutines implemented in C# or C++. However, while the GDI+ libraries are extremely efficient, they do require that the source content be fully visible on a user's screen. If the target content is outside of the user's visible desktop area (i.e., off-screen) or obscured by another application window, the resulting capture data will be incomplete. To compensate for this scenario, Xstreamulator's UI was

configured to float above all other application windows. This prioritized display approach was implemented in VB.NET through the modification of Xstreamulator's z-order property setting. In addition, a feature was added to the UI that enabled users to easily anchor the Xstreamulator UI to the visible portion of their desktop.

Implementation of the External Capture Libraries

Xstreamulator's external capturing subsystem was initially based on a third-party OLE control (OCX) component. While the OCX component supported the required external capturing functionality, it was inefficient and caused installation problems on end-user systems. Ultimately, the component was replaced by two open source DirectX based capture libraries, DirectShow.NET and DirectX.Capture.[37] The discovery of the DirectShow.NET and DirectX.Capture libraries was a critical milestone in the development of Xstreamulator's external capturing functionality. Specifically, it did not initially appear that VB.NET could support DirectX based capturing through an approach other than the inclusion of third-party OCX components. The DirectShow.NET and DirectX.Capture capture libraries were ultimately employed for external VGA capturing, combined desktop/video capture operations and the display of capture device property pages. In addition, a DirectX based device enumeration process was added to the existing Window Driver Model (WDM) enumeration codebase. The addition of DirectX enumeration significantly improved the detection of esoteric video capture devices.

The original OCX capture component was unable to dynamically detect changes in the resolution of connected VGA/Video capture devices. As this functionality is essential for correctly capturing images from external (e.g., VGA) sources, a solution had to be found. Fortunately, the DirectX.Capture library supported manual and automatic identification of external resolution changes. However, the automatic adjustment process (i.e., auto-detection) is currently too CPU intensive to be implemented. Consequently, Xstreamulator's resolution change functionality is, at present, executed at the start of an encoding session or at the discretion of the user. Nevertheless, the success in programming the external capture subsystem to reliably respond to changes in VGA capture resolutions was a significant project achievement, and it is expected that a more efficient automatic resolution detection system will be implemented in a future release. It is worth noting that the use of the DirectShow.NET and DirectX.Capture libraries required a significant programming effort. However, these libraries offer unlimited potential for future Xstreamulator functionality.

Although the DirectShow.NET and DirectX.Capture libraries were successfully employed in Xstreamulator, an alternative library is under consideration. The DirectShowLib-2005.dll library,[38] available under the GNU Library or Lesser General Public License (LGPL), appears to offer even better performance. Furthermore, this library provides greater scope for low level device control and capture optimization, as it is less abstracted from its DirectX underpinnings than DirectShow.NET and DirectX.Capture. However, the viability of implementing this library is in question, as its

use within VB.NET is very poorly documented.

Implementation of PowerPoint Slideshow Capturing Functionality

It is worth noting that earlier Xstreamulator prototypes lacked PowerPoint slideshow capturing functionality. Discussions with the beta test team revealed that inclusion of this functionality was critical to Xstreamulator's success. However, the process of programmatically implementing PowerPoint functionality as an embedded UI component was extremely challenging. The typical approach to implementing PowerPoint interoperability requires users to install a Visual Basic For Applications (VBA) plug-in directly in PowerPoint. As this approach is dependent on "weaker" VBA code, it is inherently less flexible than a fully hosted solution. Ultimately, a unique approach was implemented; PowerPoint was relegated to plug-in status within Xstreamulator's UI. However, the implementation of this approach was difficult, as the core PowerPoint functionality (opening presentations, displaying slideshows, advancing slides and so forth) had to be developed entirely in VB.NET. Fortunately, Microsoft provides a VB.NET Interop framework for communicating with its Office suite. Specifically, the process of embedding PowerPoint required the inclusion of the Microsoft Office.NET core libraries and the referencing of the Microsoft PowerPoint Object Library in the VB.NET project workspace. Once these libraries were referenced in VB.NET, it became possible to display, control and capture PowerPoint presentations using customized VB.NET code subroutines.

While it would have been possible to use PowerPoint's internal image exporting engine too populate Xstreamulator's slide capture subroutines, it was discovered that this approach produced poor quality results, as the exported slides were not anti-aliased. Instead, a higher quality approach was implemented by "Bit-Blitting" the embedded PowerPoint Slideshow window to Xstreamulator's GDI+ image processing subroutines. Ironically, this approach resulted in better quality images than are possible through PowerPoint's built-in exporting engine! In addition, the quality of Xstreamulator's PowerPoint slide conversion processes exceeds what is possible through alternative methods, including the capturing of presentations with a VGA frame grabber device. However, earlier versions of Xstreamulator required that users manually activate the slide capturing process. Some members of the beta test team found this process too distracting. Thus, automatic slide capturing functionality was added. When this feature is activated, Xstreamulator automatically captures PowerPoint slides as users advance through a presentation. This functionality will be eventually added to every capture component.

As the loading of large PowerPoint presentations causes CPU spikes on older PC systems, Xstreamulator supports the conversion (exporting) of PowerPoint presentations to a bitmap image sequences prior to a webcast. This process takes a few seconds with the majority of PowerPoint files. Xstreamulator's slideshow exporting functionality has proven particularly useful during live events. Specifically, some presenters are unwilling to furnish PowerPoint documents. However, presenters are much more willing to have their

presentations converted to non-editable images "on-the-spot". Indeed, a watermarking process will be implemented to further allay presenter concerns. The success of the embed application approach has opened up the possibility of programmatically controlling/capturing other Microsoft Office applications (e.g., Microsoft Word). However, thus far the beta test team has not requested that support for the remaining Office application be included in a future Xstreamulator release.

Audio Level Monitoring and Sound Card Management Libraries

The only remaining hurdle to the implementation of Xstreamulator's integrated UI design was the requirement to monitor and display audio input (i.e., recording) levels. Unfortunately, the monitoring of audio input levels proved far more difficult to achieve than was originally expected, as it was discovered that the Windows Media Encoder framework provided no programmatic method to access incoming audio levels. Ultimately, to implement audio level monitoring functionality, a third party .NET component (i.e.,) was required.[39] WaveEx.NET provided the necessary programmatic methods to determine recording levels and has proven to be extremely efficient and stable. Furthermore, the component supports the (simultaneous) capturing of audio streams to the Windows Audio Format (WAV) or MPEG-1 Audio Layer 3 (MP3) formats.[40] WaveEx.NET's support for WAV file recording is extremely useful, as recorded files can be compressed to a more portable format (e.g., MP3) with minimal quality loss. Thus, Xstreamulator can now simultaneously produce Windows Media content and an uncompressed audio source file for distribution to portable media devices (e.g., for the

purposes of podcasting). Interestingly, the inclusion of WAV recording support has proven so popular with the beta test team that it would now be difficult to remove this feature from Xstreamulator!

The success of the WaveEx.NET implementation and interest from the beta test team has led to an investigation of the potential of adding more sophisticated podcasting support in the Xstreamulator framework. One option that is currently being examined is the implementation of a server based application, that converts Xstreamulator captured Windows Media files to the H.264/MPEG-4 format (e.g., IPOD video). Preliminary tests indicate that this approach is feasible, as Windows Media files can be cross-converted to the MP4 format without significant quality degradation. The likely implementation of this approach is a dispatching process, whereby Xarchiver automatically initiates a podcast conversion process on a server dedicated to video transcoding.

Notes:

1 Xstreamulator has been used in numerous live events, including The Kodak Lecture Series: 2007 (http://www.ryecast.ryerson.ca/dmpstreams/2007Kodak/index.asp) and the Canadian Research Network for Care in the Community Symposium: 2007 (http://www.ryecast.ryerson.ca/ dmpstreams/crnccfeb2007/index.asp).

2 The first live Xstreamulator webcast of took place at Ryerson University on October 7, 2006. This event was a presentation by photographer Finbarr O'Reilly (http://www.ryecast.ryerson.ca/dmpstreams/2006Finbarr/index.asp).

3 Linux support is currently in development.

4 However, the addition of live Flash streaming capabilities (planned for a 2008 release) may improve cross platform compatibility.

5 Webcast operator requirements are discussed in the Media Production Guide on the ePresence website (http://code.epresence.tv/wiki/MediaProductionGuide).

6 Through support for the open source Red5 Flash Server.

7 Windows Media webcasts can be delivered using Push, Pull or Multicast delivery protocols. At present, Xstreamulator supports Push and Pull based delivery. Support for Multicast delivery is planned for a future release.

8 There is a delay from the time Windows Media content is captured to the time the content is delivered to the viewer's browser. This "buffering" time typically varies between 10 and 20 seconds.

9 Xstreamulator supports the following web browsers: Microsoft Internet Explorer, Netscape Navigator, Apple Safari and Mozilla Firefox. Other browsers may be compatible, but are not officially supported.

10 Parallels Desktop is an emulated (virtualized) environment for running Windows XP/VISTA on the Macintosh (Intel-based) OS/X platform.

11 Winnov and Osprey manufacture audio/video capture cards that are specifically designed for webcasting applications. These cards are fully compatible with Xstreamulator.

12 Epiphan Corporation (Ottawa, Canada) develops inexpensive external VGA to USB 2.0 "frame grabber" devices.

13 This system was a surprisingly effective revenue generator!

14 At the request of one of the Beta Testers, a folder-watch feature was implemented that monitored a selected folder for the presence of newly acquired image files. This feature made Xstreamulator compatible with USB cameras that capture images at set intervals (timed capture).

15 In contrast to traditional captioning, live descriptive captioning ("live described") describes physical behaviors, expressions, clothing, movement and so forth. An example of a live descriptive captioning can be viewed at: http://www.ryecast.ryerson.ca/dmpstreams/massexodus2006/index.asp.

16 VB.NET's support for accessibility includes UI object "tagging". A completely accessible version of Xstreamulator would be more difficult to implement, as the use of text readers by visually impaired users could interfere with the audio encoding process; however, the use of sound isolating headphones and multiple sound cards could help to address this issue.

17 One of the major constraints of the predefined Windows Media Encoder User Interface (UI) was that it did not support z-order functionality. Without an intrinsic z-order property there was no way to guarantee that the predefined Windows Media Encoder UI would not be obscured by an application with a higher z-order. A method to force the Encoder UI to the front of the z-order "stack" was eventually developed, but this procedure was too CPU intensive to run as a constant process. Eventually, a "brute force" solution was implemented to bring the UI to the front of the z-order when a user's mouse clicked or moved over the Xstreamulator application UI. However, this solution still required user interaction to bring the predefined UI to the top of the z-order "stack". Furthermore, CPU "spikes" occurred when the UI was moved by the user. Attempts to hide the UI during application moves did not resolve this problem, as the process of hiding and redisplaying the UI placed a high burden on the user's CPU.

18 Initially, each Xstreamulator application module was developed in a separate form. For example, the PowerPoint form was solely responsible for PowerPoint capture and display functionality.

19 A Multiple Document Interface (MDI) application is composed of a single "parent" form that contains other "child" forms.

20 Expand/contract functionality was implemented to enable users to reduce the size of Xstreamulator's UI when required.

21 Instructors are required to enter a password/username when webcasting from a classroom. However, in a future Xstreamulator release, authentication settings will be automatically recalled from a USB memory key or RFID tag. The use of biometric authentication (e.g., the use of fingerprint readers) is also being explored.

22 The main encoder form in Xstreamulator has, on occasion, become corrupted. As a result, application backups are performed on a very frequent basis!

23 Code modules contain text-only application subroutines. They do not include UI components, which are instead stored in binary resource files. In essence, code modules permit the separation of visual elements from an application's codebase.

24 As modules are "public" code repositories they can be referenced from any part of the application project workspace. Thus, modules contribute to a more efficient and manageable codebase.

25 There are no specific CPU requirements for Xstreamulator. However, system performance can limit viable capture options. Realistically, Xstreamulator requires a Pentium III or faster computer.

26 A CPU "spike" occurs when the computer's Central Processing Unit is being fully utilized, usually during a brief period of time. Periodic CPU spikes are rarely problematic. However, "spikes" that occur over a longer time frame can result in sluggish performance or application crashes.

27 A network or system failure.

28 During a "failure event" Xstreamulator attempts to recover/upload the existing encoded content. In addition, Xstreamulator automatically self-resets so that an encoding session can be rapidly restarted.

29 USB web cameras tend to use more CPU cycles than comparable Firewire cameras. Unfortunately, due to USB's market dominance, Firewire cameras are becoming increasingly scarce.

30 Background tasks include Windows services and applications loaded during the Windows startup process. For example, anti-virus applications are notorious for degrading the performance of a Windows PC.

31 The user setting syntax is: `My.Settings.Variable Name = Value.`

32 Windows Media Encoder 9.0 has a built-in audio input level indicator. Unfortunately, there is no method to monitor volume levels using the COM interfaces provided by Windows Media Encoder 9.0. Specifically, the Windows Media Encoder SDK identifies a record level property, but states that it is "not supported in the current release". Yes, this is very annoying!

33 QuickTime does not support the live-insertion of URL "flips" or multi-bitrate encoding. In addition, the QuickTime SDK requires C++ or JAVA.

34 C++, C# and Visual Basic .NET control WMEncoderLib objects using procedures specific to each language.

35 GDI and GDI+ are the primary libraries used to display 2D content in Windows. In Windows Vista, GDI+ is supplanted by a new low-level graphics/screen management system, the Windows Driver Display Model (WDDM). However, GDI+ functionality is fully supported in a VISTA as a "compatibility layer".

36 Interestingly, this communication can occur bi-directionally, although Xstreamulator does not at present make use this feature.

37 The DirectX.Capture library was developed by Brian Low, and is available on the Code Project Website (http://www.codeproject.com/KB/directx/ directxcapture.aspx). The DirectShow.NET library was developed by "NetMaster" and is also available from the Code Project website (http://www.codeproject.com/cs/media/directshownet.asp).

38 DirectShowLib-2005.dll is available from SourceForge (http://sourceforge.net/projects/directshownet/).

39 The vendor's website is: http://www.tapiex.com/WaveEx.Net.htm.

40 MP3 support is currently in development. An earlier version of Xstreamulator supported this feature. However, the control used to generate MP3 files was replaced with WaveEx.NET. WaveEx.NET's MP3 implementation is not currently activated in Xstreamulator.