Multimedia Capture of Events to Support Passive Virtual Attendance

by

Mohamed Atef Hendawi

Submitted to the

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

in partial fulfillment of the requirements

for the degrees of

BACHELOR OF SCIENCE

and

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1995

© Mohamed Atef Hendawi, 1995

The author hereby grants to MIT and Bellcore permission to reproduce and to distribute copies of this thesis document in whole or in part.

Signature of Author

Department of Electrical Engineering and Computer Science, May 12, 1995

Certified by

Andrew Lippman, Associate Director Media Lab, MIT Academic Advisor

Certified by

Michael Lesk - Company Supervisor, Bellcore

Accepted by

F.R. Morgenthaler, Chair, Department Committee on Graduate Students
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUL 17 1995
Multimedia Capture of Events to
Support Passive Virtual Attendance

Mohamed Atef Hendawi

Submitted to the Department of Electrical Engineering and Computer Science on
May 12, 1995 in partial fulfillment of the requirements for the degrees of
Bachelor of Science and Master of Science.

Thesis Supervisor: Andrew Lippman, Associate Director MIT Media Lab

Abstract

Often times it is desirable to record events such as lectures, training sessions, and
meetings so that they may be reviewed later. Current techniques for creating
such recordings, such as video cameras and tape recorders, are inadequate
because they fail to capture enough of the essence of an event. Attempting to use
multiple media to capture more of an event results in a cumbersome collection of
videotapes, audio cassettes, and papers that is difficult or impossible to play back
in a synchronized manner. In addition, such analog recordings are difficult to
browse. The main reason for these difficulties is that the media exist largely in
isolation from one another.

This research describes a digital solution for capturing events while preserving as
much of the event as possible. Multiple media are captured digitally as separate,
but associated, time-correlated streams. Because the streams are kept separate
and represented digitally, there are opportunities for utilizing sophisticated
searching and browsing techniques, and flexible distribution paradigms. The
work being done is part of the STREAMS effort at Bellcore. STREAMS is a
multimedia capture/playback system for recording and reviewing events.

A capture environment was created to allow users to describe the content of
events and then record them. These events consist of a number of streams that
are used to record specific parts of an event. For example, an event may have a
video stream of a speaker talking, video of an audience, audio streams, electronic
whiteboard and viewgraph streams. The capture system is extensible and allows
for future stream types to be incorporated into the system without necessitating
changes to the main capture application. A generalized notion of capture devices
was designed and utilized in the implementation. The capture system also
supports creation of different representations of stream data.

This thesis will first precisely describe the role of the capture environment within
a system like STREAMS. Design issues involved in realizing a capture
environment will be discussed. Specifically, issues of how to ensure future
extensibility of stream types and capture devices, will be addressed. Finally, an
implementation of a capture system, and the difficulties encountered, will be
considered.
Acknowledgments

First and foremost I'd like to thank Gil Cruz for letting me in on his vision of STREAMS and allowing me to take on a part of the system that afforded me almost unbounded potential for growth. Gil always made sure that he was available and interested, far beyond what could reasonably be expected of any thesis advisor. His continually upbeat and forward-looking attitude was key in making me strive to achieve the best possible. Thanks for helping me get here Gil. Keep on pushing that envelope!

Thanks to Mark Castrovinci for his technical help and for introducing me to newer and better ways of doing things. He was always willing to take time out of his day to show me a trick or two, or to just exchange ideas.

Ralph Hill provided technical support and advice that was invaluable to me. Thanks for helping me get up to speed on things.

Thanks to Jonathan Rosenberg for his sensible and practical advice - without which I might have lost focus. I especially appreciated the freedom he offered me to pursue topics of interest.

Leo Chang and Ted Ko helped keep me sane during the entire process and were also fantastic brainstorming partners.

Thanks to Bellcore for the use of its facilities and for bringing me in contact with a great group of people.

Thanks to Andy Lippman and Michael Lesk for reviewing, and approving, this thesis.

As my years at MIT come to a conclusion, I would like to dedicate this thesis to my parents for always being behind me in whatever I've done and instilling in me the drive and curiosity that have brought me to this point. I am approaching graduation with a sense of great satisfaction and for that, I have my parents to thank.
Table of Contents

Abstract ........................................................................................................................................... 3
Acknowledgments .............................................................................................................................. 5
Table of Contents .............................................................................................................................. 7
Table of Figures .................................................................................................................................. 13
1. Motivation and Related Work ......................................................................................................... 15
   1.1 Introduction ................................................................................................................................ 16
       1.1.1 The Problem: Current user scenarios and approaches ......................................................... 16
       1.1.1.1 Analog recording techniques ............................................................................................ 17
       1.1.1.2 The Role of Video ............................................................................................................ 18
       1.1.2 Use of “Multiple-Media” In Event Capture ........................................................................... 18
       1.1.2.1 Problems with multiple-media solutions ......................................................................... 19
       1.1.3 Digital recording techniques ............................................................................................... 19
       1.1.3.1 Advantages/Disadvantages of digital capture ................................................................... 19
       1.1.3.2 The State of Multimedia .................................................................................................. 20
   1.2 The STREAMS approach - a new solution for event capture .................................................... 21
       1.2.1 Browsing and Searching ...................................................................................................... 22
   1.3 The focus of this research ........................................................................................................... 22
       1.3.1 Roadmap to this thesis ......................................................................................................... 22
   1.4 Related Work .............................................................................................................................. 23
2. STREAMS - Then and Now ............................................................................................................ 25
   2.1 Capturing in the prototype environment ..................................................................................... 25
   2.2 Playback in the prototype environment ...................................................................................... 25
       2.2.1 Search tools ........................................................................................................................... 27
       2.2.2 Dependence on stream types and media types ...................................................................... 27
       2.2.3 Playback model .................................................................................................................... 28
   2.3 STREAMS - requirements for the next generation ..................................................................... 29
       2.3.1 Device and hardware independence ................................................................................... 29
2.3.2 Support for different stream types ............................................. 29
2.3.3 Flexible distribution paradigms - Networked client/server interaction and CDROM .......... 29
2.3.4 Scalability  ................................................................................. 30
2.3.5 Flexible codec support ................................................................. 30

2.4 Next generation STREAMS architecture ......................................... 30

2.4.1 Capture System ........................................................................... 30
  2.4.1.1 Platform selection ................................................................. 31
  2.4.1.2 Capture environments and locales ......................................... 32

2.4.2 Event Server ................................................................................ 32
2.4.3 Presentation Clients ..................................................................... 33

2.5 The context of this work within the next generation of STREAMS .................................. 33

3. Capture system design and implementation ........................................ 35

3.1 Capture System Entities .................................................................. 35
  3.1.1 Streams ...................................................................................... 35
    3.1.1.1 Identification of streams ...................................................... 36
    3.1.1.2 Types of streams ................................................................ 36
      3.1.1.2.1 Standard types .............................................................. 37
    3.1.1.3 Stream format .................................................................... 37
      3.1.1.3.1 Stream handlers ............................................................ 38
    3.1.1.4 Stream Data - Samples ...................................................... 38
      3.1.1.4.1 Sample organization ...................................................... 38
      3.1.1.4.2 Stream rate .................................................................. 39
    3.1.1.5 Stream Versions ................................................................. 39
      3.1.1.5.1 Support for different hardware and software codecs ......... 42
      3.1.1.5.2 Support for preview and focus ....................................... 42
      3.1.1.5.3 Streams & versions ....................................................... 43

  3.1.2 Events ....................................................................................... 43
    3.1.2.1 Event as a collection of streams ......................................... 43
      3.1.2.1.1 Canonical forms of events ............................................ 43
    3.1.2.2 Event information .............................................................. 44
    3.1.2.3 Participant involvement ...................................................... 45
    3.1.2.4 Implementation issues for events ....................................... 45
      3.1.2.4.1 Persistent event storage - atomic vs. distributed .......... 45
      3.1.2.4.2 Extensibility ................................................................. 46

2.3.2 Support for different stream types ............................................. 29
2.3.3 Flexible distribution paradigms - Networked client/server interaction and CDROM .......... 29
2.3.4 Scalability  ................................................................................. 30
2.3.5 Flexible codec support ................................................................. 30

2.4 Next generation STREAMS architecture ......................................... 30

2.4.1 Capture System ........................................................................... 30
  2.4.1.1 Platform selection ................................................................. 31
  2.4.1.2 Capture environments and locales ......................................... 32

2.4.2 Event Server ................................................................................ 32
2.4.3 Presentation Clients ..................................................................... 33

2.5 The context of this work within the next generation of STREAMS .................................. 33

3. Capture system design and implementation ........................................ 35

3.1 Capture System Entities .................................................................. 35
  3.1.1 Streams ...................................................................................... 35
    3.1.1.1 Identification of streams ...................................................... 36
    3.1.1.2 Types of streams ................................................................ 36
      3.1.1.2.1 Standard types .............................................................. 37
    3.1.1.3 Stream format .................................................................... 37
      3.1.1.3.1 Stream handlers ............................................................ 38
    3.1.1.4 Stream Data - Samples ...................................................... 38
      3.1.1.4.1 Sample organization ...................................................... 38
      3.1.1.4.2 Stream rate .................................................................. 39
    3.1.1.5 Stream Versions ................................................................. 39
      3.1.1.5.1 Support for different hardware and software codecs ......... 42
      3.1.1.5.2 Support for preview and focus ....................................... 42
      3.1.1.5.3 Streams & versions ....................................................... 43

  3.1.2 Events ....................................................................................... 43
    3.1.2.1 Event as a collection of streams ......................................... 43
      3.1.2.1.1 Canonical forms of events ............................................ 43
    3.1.2.2 Event information .............................................................. 44
    3.1.2.3 Participant involvement ...................................................... 45
    3.1.2.4 Implementation issues for events ....................................... 45
      3.1.2.4.1 Persistent event storage - atomic vs. distributed .......... 45
      3.1.2.4.2 Extensibility ................................................................. 46
3.2.3.1 Video preview - Overlay vs. unassisted preview
3.2.4 Capturing events
3.2.5 Event preparation
3.2.6 Saving events
3.2.7 Configuration files - Setting up the capture system
  3.2.7.1 CAPTURE.INI
  3.2.7.2 SYSTEM.INI

3.3 Software architecture
  3.3.1 The use of C++ - STREAMS and object-oriented organizations
  3.3.2 Capture devices
  3.3.3 Capture Device Manager
  3.3.4 Event documents and views
  3.3.5 MrCapture - the man behind the scenes
    3.3.5.1 Capturing events with video
      3.3.5.1.1 Using AVICap
      3.3.5.1.1.1 Video buffers and AVICap
      3.3.5.1.1.2 Capture file issues
    3.3.5.1.2 Multi-source Video Interleaving Algorithm
      3.3.5.1.2.1 Requirements for interleaving algorithm
      3.3.5.1.2.2 Details of The Algorithm
      3.3.5.1.2.3 Implementing the algorithm
      3.3.5.1.2.4 Caveats and Future directions
    3.3.5.1.3 Post-processing events with video
    3.3.5.2 Capturing events without video

4. Results and Conclusions
  4.1 What was implemented
  4.2 Problems encountered - Future opportunities
    4.2.1 Guaranteeing performance - a problem of trust
      4.2.1.1 Possible solutions
    4.2.2 Loss of information
      4.2.2.1 Possible solutions
  4.3 Conclusions

Appendix 1 - Specifications
References 91
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The STREAMS Prototype Playback Interface</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>Viewgraph Search Tool</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>Architecture of next generation STREAMS System</td>
<td>31</td>
</tr>
<tr>
<td>4</td>
<td>Calculating a FOURCC value</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>Reading and writing a structure</td>
<td>48</td>
</tr>
<tr>
<td>6</td>
<td>Two-tiered driver architecture</td>
<td>55</td>
</tr>
<tr>
<td>7</td>
<td>A comparison of isolation approaches</td>
<td>60</td>
</tr>
<tr>
<td>8</td>
<td>Capture Application Main Workspace</td>
<td>62</td>
</tr>
<tr>
<td>9</td>
<td>Adding a stream to an event</td>
<td>64</td>
</tr>
<tr>
<td>10</td>
<td>Video Format Dialog</td>
<td>65</td>
</tr>
<tr>
<td>11</td>
<td>Audio Format Selection Dialog</td>
<td>65</td>
</tr>
<tr>
<td>12</td>
<td>Configuration dialog for an electronic whiteboard</td>
<td>66</td>
</tr>
<tr>
<td>13</td>
<td>Initialization Dialog</td>
<td>67</td>
</tr>
<tr>
<td>14</td>
<td>Event Browser Dialog</td>
<td>69</td>
</tr>
<tr>
<td>15</td>
<td>Sample CAPTURE.INI</td>
<td>70</td>
</tr>
<tr>
<td>16</td>
<td>Sample SYSTEM.INI drivers section</td>
<td>70</td>
</tr>
<tr>
<td>17</td>
<td>The CaptureDevice C++ Class Hierarchy</td>
<td>72</td>
</tr>
</tbody>
</table>
1. Motivation and Related Work

Most of the important activities that take place in businesses today revolve around gatherings and meetings of people. It is during such meetings that company strategies are articulated, software designs are debated, and personnel are educated - to give just a few typical scenarios. Sometimes people gather at planned times. Other times, meetings are spontaneous - a result of chance inspiration. Yet despite the significance of such events, much of their essence is fleeting in nature. Often the complex sequence of propositions, negotiations, illumination, and understanding that comprise the overall learning processes in an event are lost forever - if lucky, maybe reduced to a few “salient” points. We speak of learning “processes” in the plural - for the event is not really a single process, but merely the substance from which each individual’s own interpretations coalesce to lead to a common understanding.

This work addresses how to best capture this substance, so that it may later be interpreted and experienced. Through the use of digital multimedia, it is postulated that an effective environment for rich capture and playback of events can be provided. Specifically addressed in this research are issues related to the design and context of a capture system within a specific environment - the STREAMS project at Bellcore. Some important ideas addressed by this work include:

- How to best realize a rich recording of an event - i.e. what the makeup and configuration of recorded events should be.
- The significance of using digital representations for stored multimedia data and how that affects the playback and browsing of correlated media streams.
- Techniques necessary to ensure extensibility - i.e. how to easily incorporate new and unforeseen data types and capture devices
- User interfaces for a capture system to allow easy control of capture functionality.

This chapter will discuss why it is important yet currently difficult to accomplish this kind of functionality, review some background material, and describe the contribution of this research in more detail.
1.1 Introduction

A permanent record of "interpersonal communications events" is often desirable. "Interpersonal communications events" are characterized by a group of participants engaged in information exchange, possibly with the assistance of both electronic and non-electronic devices. Examples of such events are training sessions, technical presentations, lectures, and meetings. A permanent record of events such as these can allow people who were not in attendance at the event to review the proceedings at a later time. It can also allow people who were in attendance to go back and review the record for detail or clarification.

There are currently a variety of techniques available to create such a record. However, many of these techniques are not adequate for capturing the class of events of interest here. This research discusses the relative merits and shortcomings of these techniques and describes a new approach to capturing events. This new approach depends heavily on the notion that a synthesis of many of these techniques considered together is necessary to realize a truly rich record of an event. The approach is especially well-suited to capturing interpersonal communications events (hereinafter referred to as "events" or "IC events").

1.1.1 The Problem: Current user scenarios and approaches

The simplest approach used to capture events is to have a person take written notes about the proceedings. This can be in the form of an active participant in the event compiling "minutes" after the gathering - relying heavily on the person's memory and any personal notes used to jog their memory. There are a number of problems with such an approach. If the person preparing the minutes is unaware of their role, then the notes can be expected to be of lower quality. On the other hand, if the person is aware of their role, then that person's experience of the event is likely to be corrupted by the knowledge of their secondary task. An obvious "optimization" of the approach, in deference to the latter issue, is to assign the task to a dedicated person, such as a secretary. This however, requires planning and forethought, preventing its applicability in spontaneous situations.

A problem with all of these approaches is that the resulting records are subject to the interpretations of the transcriber. While the person may provide a satisfactory account of the major points visited, adequate context and support for those points may be lacking. This potentially
results in less understanding of the notes as an individual’s enabling comprehensions are lost with time. “Enabling comprehensions” refer to the understanding that may come from the more subtle qualities of an event that may not seem to be directly relevant to the subject matter - e.g. gestures, mannerisms, group dynamics, and tone of voice.

On the other hand, taking notes of an event’s proceedings is an extremely popular and ubiquitous technique. This is due to the minimal support required and its unobtrusive nature. These latter points need to be recognized as significant in ensuring the success and adoption of other potential techniques.

1.1.1.1 Analog recording techniques

It is clear that a recording that is not subject to bias by individual human interpretation is desirable. Consider some of the possibilities afforded by conventional analog media for accomplishing this. Recording the proceedings of an event with an audio tape is an excellent way to capture the entire dialog of a meeting in an objective, unspoiled manner. However, audio (speech) is only one small part of a gathering. Other aspects, such as facial expressions and visual aids such as viewgraphs, are completely missed in such a recording. It is also very difficult to search through an audio record due to its non-visual and sequential nature. It should be noted that there are techniques available to improve the searchability of audio; however, they are not yet widespread and they almost always depend upon digital representation of the audio [Arons94].

Another popular technique today that can be used to capture events is the combination of video and audio on an analog videotape. The cost of the media is negligible and has a very small form-factor. Video equipment such as camcorders can be relatively inexpensive and are widely available. The quality of the recording varies with the level of equipment but is always more than adequate for conveying the content that was meant to be saved. Overall, it is an excellent capture medium for a variety of simple applications. However, there are several severe shortcomings for capturing IC events. For example, camcorders and video tapes can only capture one video and audio sequence at a time\(^1\). In addition, like audio, it is difficult to search and browse through the recording since it

\(^1\) Analog video tape can store a single stereo audio track. However, it is possible to utilize the stereo channels separately and store two independent monaural audio tracks if necessary.
is captured using an sequential analog representation\(^2\). There are some techniques that address these shortcomings. Multiple cameras can be used with a director selecting among the choices. Getting a director, however, is an expensive proposition and is something that requires even more planning and forethought than that required of enlisting a secretary’s assistance. SMPTE time code, a technology which facilitates indexing of video tape, can be applied to the recording, partially addressing the searching problem. However, SMPTE time code, though helpful, still doesn’t provide much in the way of browsing support.

1.1.1 The Role of Video

Despite video’s advantages for objectively capturing visual aspects of a meeting, the precise role of video in recordings of events needs to be better understood. Consider one video based medium -- television. Reuven Frank, long-time president of NBC News elucidates: “Television is best as a narrative medium, worst as an expository medium.” Video is in fact recognized as an extremely important medium; however, it is also universally recognized that video alone is unsuitable for instruction and presentations of tasks that may require “analytic or reflective” thought [Moore93]. Many events fit this characterization, such as training sessions and lectures. Video’s lack of interactivity, manifested in one dimension as poor browsability, also dooms video as a sole medium for any sort of instructional effort [McKeachie75].

1.1.2 Use of “Multiple-Media” In Event Capture

As described, events such as meetings typically have many facets that add to the overall experience. For example, in addition to the speech dialog, there are also the physical interactions of people, facial expressions, and the use of visual aids. Since events are essentially multidimensional in nature, recordings of events must be as well. What is clear from this discussion is that any single-medium solution, such as audio and video alone, is inadequate. To effectively capture events, *multiple-media* solutions must be employed. For example, to capture a lecture we might want to separately capture video of a speaker, video of an audience, audio of the speaker and possibly the audience, copies of handouts and viewgraphs, and maybe even some sort of record of whiteboard or blackboard activity.

\(^2\) It is important to note that it is the analog representation that is limiting, and not the videotape itself. Videotape can be used to store digital information, although this is not a common application.
1.1.2 Problems with multiple-media solutions

Let's try and consider actually doing what has just been described. First there is the issue of recording with all of the media. Setting up all of the equipment is an ordeal in and of itself. In addition, some things are difficult to record well, such as a whiteboard or blackboard session. Then there is the end-result of the effort - a jumble of video tapes, audio cassettes, photocopies, etc. Now consider actually going back and reviewing this - not only is it a logistical nightmare getting all of the components to work simultaneously, but it is nigh near impossible to do any sort of browsing or searching without any synchronization among the various media. There must be a better way.

1.1.3 Digital recording techniques

There are some fundamental limitations to analog media recordings that make them difficult to use for events. Analog media is cheap and common, but inflexible and generally difficult to browse. Digital capture of events is an option which is now becoming possible due to improvements in hardware, software, and compression technologies. There are currently some options for digital capture which are of interest here. Specialized systems to digitally record proceedings of events exist, but are often single-dimensional in nature. For example, Filochat provides a way of recording conversations and co-indexed written notes in a digital notebook [Whitaker94]. Softboard, an electronic whiteboard, allows recording of whiteboard sessions to a computer file for later review [Softboard94]. RAD ScreenPlay allows simultaneous capture of computer screens and audio [RAD94]. In addition to these techniques, there are also a bevy of tools available to capture digital audio and video. Each of these approaches can capture one or two aspects of an event while leaving out many potentially important sources. This severely limits their usefulness for IC events because such tools can rarely be used in a vacuum.

1.1.3.1 Advantages/Disadvantages of digital capture

There are several benefits of using digital capture over analog techniques. The most obvious are the browsing and searching capabilities that are afforded by digital representations, and the random-access features provided. Digital representation of media is also more amenable to customization and processing by computers. Tools exist to do digital video editing, filtering, and other sorts of processing as well. Another benefit is flexibility in playback. Synchronization of media is easier, although non-trivial, using digital representations. Digital representations can provide better quality than analog representations for some things such as electronic whiteboards.
and high resolution images. Finally, digital media results in better storage hygiene. Specifically, this means that it is possible to store all of the proceedings of an event digitally on a single medium such as a CD-ROM or a hard disk. There is no longer a need to maintain an assortment of video tapes, papers, and cassettes to store a single event.

There are currently some downsides to digital media. Storage requirements, and as a result, expense, is very high compared to analog media. An indirect side-effect of this is reduced quality. While there is no fundamental reduction in quality inherent in digital media, the storage requirements to produce results commensurate with analog can be imposing. In addition to the storage penalty, digital media may also incur a learning curve because the technologies are relatively new and the tools are not yet mature.

1.1.3.2 The State of Multimedia

"Multimedia" is touted as the technology that will bring computing to the masses. However, multimedia as a technology is still in its infancy. The main problem with current multimedia solutions is a lack of integration. For example, there are a bevy of tools available to capture digital video with synchronized audio on the market today. However, these are all single purpose tools that provide for little or no integration. If anything more complex than this is desired, it often involves a frustrating process of massaging file formats, synchronization, and getting different isolated tools to cooperate. In the end, more often than not, the undertaking is given up altogether, or the end result is something that is less than ideal. This lack of integration is particularly crippling for the applications of interest here - capture and playback of lectures, training sessions, etc. Such events are inherently multimedia in the most general sense of the word - multiple sequences of multiple types of media need to be handled. In addition, such events can not easily be fit into a static model. For example: training sessions can vary in the types of visual aids that are used, the size/length of the session, activity of the participants, etc. Compare that with the needs of a design review - a situation which may involve completely different visual aids and a much more symmetrical form of information exchange.

---

3 Current CDROM technology is probably inadequate for all events. However, advances in CDROM storage and bandwidth are imminent and will make CDROM an ideal distribution medium.
1.2 The STREAMS approach - a new solution for event capture

What is needed is an integrated solution for digital multimedia capture and playback of events. The overriding concern of such a system should be to handle a wide variety of different media in a uniform manner. It should not only be able to handle different types of media, but also different configurations of the media streams as well.

The context for this work is the STREAMS system developed at Bellcore. STREAMS is an integrated capture/playback solution for digital multimedia preservation of events [Cruz94]. One of the guiding principles of the STREAMS system is to replicate as much as possible the actual experience of being at the event. The ideal is to enable "passive virtual attendance" where there is no difference, from an informational content point of view, between the recorded event and the actual event. The only significant difference is not being able to actually interact with the participants in the event while the event is occurring.

The key to enabling "virtual attendance" is to empower the viewer of an event to make their own choices of what is important to concentrate on. For example, consider an audience member at a lecture. The person's focus may shift from the lecturer to a question from the audience, or to the viewgraphs. The person is allowed to absorb the available information at his/her own pace. In fact, it can be argued that the freedom a person has in deciding what to concentrate on and when to concentrate on it is crucial in successful learning and understanding [Berlin92]. STREAMS recognizes this need and incorporates the option for this freedom into the recording⁴.

In order to realize this ideal, a central concept in STREAMS is to allow as much of an event to be captured as possible. This means that only the expense of capture equipment should limit the amount and breadth of an event that can be recorded. STREAMS is intended to be accommodating of a large number of media streams of widely differing types. For example, the STREAMS system will handle video, audio, viewgraphs, electronic whiteboards, personal digital assistants, as well as any other type of media that could be useful in an event context.

---

⁴ It is interesting to note that many of today's distance-learning applications fail to recognize this need and restrict users to focusing on a TV monitor showing director-selected views.
After an event has been captured, STREAMS provides a playback environment which attempts to best present the event to the viewer. This playback environment allows the user to scan the event for areas of interest. The user can then choose to focus in on particular parts of an event for more detail. In addition, sections can easily be reviewed.

1.2.1 Browsing and Searching

By adopting a digital approach to media capture, STREAMS allows sophisticated searching techniques, custom-tailored to the media type, to be made available. These searching techniques can be used to quickly find relevant sections of an event, or to aid in comprehension of information in an event. For example, a stream of viewgraphs could be used to quickly locate the point in the event when a certain topic was discussed.

It can be argued that the success of multimedia depends a great deal on the existence of powerful searching and browsing techniques [Gabbe94]. The sheer amount of information that multimedia provides is both a boon and a source of trouble. If it is difficult to access and find information, people may avoid doing it altogether. Addressing this need for advanced searching is one of the central goals of the STREAMS system.

1.3 The focus of this research

This work is an effort at realization of the capture component of the STREAMS system. Specifically, it is an effort to discover what is necessary, from a design and implementation point of view, to provide most of the functionality desired in an integrated multimedia capture system, such as that described. There is already an existing prototype STREAMS system - however, it is merely a proof of concept, and does not consider many of the issues that need to be addressed to provide a robust, extensible, and truly useful capture environment. This work proceeds concurrently with the development of the next generation of STREAMS.

1.3.1 Roadmap to this thesis

Chapter 1 provides an introduction to the problems that are being addressed, and a general sense of what STREAMS is. Chapter 2 is a discussion of the existing STREAMS prototype system, its failings and shortcomings, and what was learned from it - good and bad. This section also introduces the general architecture for the next generation STREAMS system. Chapter 3 covers the design and architecture for the next generation capture system, focusing on the issues of device
and stream independence as well as a plug-in architecture for capture devices. The user interface to the capture system is also considered here, as it is important as an integrating environment for the capture. Chapter 4 discusses future work possibilities and concludes the thesis.

1.4 Related Work

STREAMS can be thought of as a type of groupware - enabling technology to support groups of people working together. As a result, much related work exists in the Computer-Supported Collaborative Work (CSCW) areas of research. For example, multimedia collaboration environments such as CECED [Craighill93], TeamWorkstation [Ishii90], and Nemesis [Katseff94] deal with many of the same issues that STREAMS must deal with in the real-time presentation of multimedia data over a network. Nemesis in particular is similar to STREAMS technically; however, the emphasis in Nemesis is on the enabling network protocols to support such functionality. Forum [Isaacs94], developed at SunSoft, studies related issues in event presentation, although it is in the context of a live interactive distributed presentation environment.

There is a great deal of work which has potential for use in STREAMS that deals with searching and browsing techniques. For example, work such as VideoMAP [Tonomura93] and the Hierarchical Video Magnifier [Mills92] are worthwhile considerations for use as video search tools. Other similar work includes the Video Streamer developed at the MIT Media Lab [Elliot93] and the automatic structure visualization work done by Hitachi [Ueda93]. For audio, the SpeechSkimmer developed at the Media Lab as well, has ramifications for audio searching tools [Arons94]. Work also exists on wordspotting, a technique for locating keywords and phrases within fluent speech, which can be useful for text-indexing into events with audio.
2. STREAMS - Then and Now

STREAMS is a capture/playback solution for digital multimedia preservation of events. Currently, STREAMS is a prototype system. In its current state, very basic facilities are provided for capture of events. A relatively more sophisticated system exists for playback. Support for different kinds of search tools exist as part of the playback system, such as video and viewgraph browsers, however their level of integration and automation vary from very good to little or none. In this chapter, the existing STREAMS prototype will be briefly covered to provide a basic foundation for understanding the new work. The problems encountered, and the lessons learned, will be emphasized as these are the driving forces for the next generation of STREAMS.

2.1 Capturing in the prototype environment

Capture is currently limited to one basic configuration -- zero to two audio streams and up to four video streams (given sufficient frame grabbers). There is no integrated capture environment. To capture audio and video, specialized utilities must be used. These utilities have only rudimentary facilities for synchronization among the resultant stream files. As a result, there is often a hand-processing step after the streams are captured to "homogenize" the streams. Typically the capture process involves using several UNIX workstations and a series of shell scripts invoking separate command-line programs. Capturing of an event is thus prone to many difficulties stemming from the lack of an integrated capture application.

A much more significant limitation of the capture facilities are that they are dependent on specific hardware. Capture can currently be accomplished on two Sun workstation platforms - those running the Solaris operating system with SunVideo frame grabbers and those running SunOS with VideoPix boards. There are no facilities for capturing from other devices such as electronic whiteboards or notepads.

2.2 Playback in the prototype environment
The playback environment consists of an integrated application which can present captured events. There are two main parts to the environment - the first part allows you to select among captured events. The second part is a playback interface to replay selected events. An example of the playback interface is given in Figure 1. Figure 1 shows an example of the actual playback interface. This interface presents all of the available video and audio streams in an event. There are also facilities for playing back other streams such as electronic whiteboard streams and viewgraphs provided as a sequence of bitmaps. The playback interface provides VCR-like controls for navigating through an event. There are also other controls available to provide more random access to parts of the event. As the user plays back or seeks through an event, the system ensures that the streams remain synchronized.

Figure 1 - The STREAMS Prototype Playback Interface
The playback environment maintains a notion of a preview and a focus view. All streams are given a preview view which is a spatially reduced version of the focus view. There is one visual and one auditory focus view which are used to convey greater detail and temporal resolution of a stream than can be afforded in the preview windows. The user can select which stream gets the focus view at any given moment. Preview views (such as reduced video icons) are generated from the high-resolution stream data that is recorded. Preview and focus views are necessary for several reasons. First, there are the concerns of limited bandwidth and CPU resources. Second, there are issues of practical presentation limitations due to screen size and resolution. Finally, it would be difficult for a user to focus on multiple visual streams simultaneously. Given these reasons, it is left to the user to decide what is important to focus on.

2.2.1 Search tools

The playback system provides different search tools used to allow for more sophisticated searches than those possible with VCR controls. For example, given a series of viewgraph titles, it is possible to select a viewgraph and be taken to the point in the presentation when that viewgraph was first shown. An example of such a search tool is given Figure 2. There are other search tools provided such as a video streamer [Elliot93] for video browsing and whiteboard ticker to scan through significant whiteboard events.

The generation of the data used for these tools varies greatly from tool to tool. For example, the generation of the data for the viewgraph and video search tools is largely automated and can be generated from the streams themselves. Data for audio search tools, such as information about who is speaking at any given moment in an audio stream, needs to be constructed manually due to a lack of adequate technology. If technology becomes available to do automatic speaker identification in continuous audio, it could be used to automate creation of such search tool data. Likewise, future technology to do accurate word-recognition could be used to avoid manually transcribing audio streams for use in more sophisticated text-retrieval tools.

2.2.2 Dependence on stream types and media types

The playback environment can only interact with the few well-defined stream types discussed - video, audio, viewgraphs, and electronic whiteboard. No provisions are made to incorporate any other kinds of streams; if other stream support was desired, additional code would need to be
incorporated into the main playback application. In fact, this was the way support for electronic whiteboard streams was added long after the original stream types were first supported.

Not only is the playback system dependent upon specific stream types, but it is also dependent upon specific stream formats as well. For example, video streams must either be in Motion JPEG, uncompressed RGB or a special 8-bit palletized format. Electronic whiteboard support is only limited to files generated by the Microfield Graphics Softboard. These limitations make it difficult to incorporate new capture hardware or additional software compressor/decompressors (codecs) into the STREAMS environment.

2.2.3 Playback model

Currently, the STREAMS playback system consists of a standalone playback module that plays back a captured event from a local or networked disk. No provision for playback over a network is available, except through techniques such as networked file systems. Such techniques work, but are not optimized for multimedia and in general provide unacceptable performance. While the capability to play back from local files is useful with distribution paradigms such as CDROM, in general it is unacceptable in providing a large library of available events for users to choose from.
2.3 STREAMS - requirements for the next generation

The overall goal of STREAMS is to serve as a platform for widespread dissemination of multimedia information and content in support of knowledge transfer within a company. In order to support this goal, the next generation of STREAMS is being designed to support the following:

2.3.1 Device and hardware independence

Consider the typical office computing environment of today. Normally, one will find networks of heterogeneous computers. The variety of different desktop platforms is large. Not only are there a wide variety of computing platforms available running different operating systems and windowing systems, there are also a huge number of different capture devices and add-on boards available as well. A requirement of STREAMS is that there should be no fundamental constraints that prevent using any of these platforms or devices. Even if a device is not supported directly in the STREAMS capture environment, it should be easily integrated into the environment without necessitating modification to the STREAMS applications themselves. This also includes supporting new stream types that may be introduced by the capture device in the playback environment. This extensibility requires STREAMS to have a plug-in architecture that makes use of dynamic linking to provide runtime support for devices. Such an architecture has been developed and is described in Chapter 3.

2.3.2 Support for different stream types

Related to the device-independence requirement is stream-independence. What this essentially means is that there can be no fixed notion of the types of streams that can be handled in the STREAMS environment. For example, it should be completely reasonable to be able to handle unforeseen stream types in the future without necessitating massive rewrites of the STREAMS applications themselves. This requirement also includes automated generation of search tools based on stream types. To realize this ideal the intelligence about stream-dependent handling needs to be moved out of the STREAMS applications and into the streams themselves. This is a classic opportunity for utilizing object-oriented techniques for encapsulation. Mechanisms for doing this, via C++ classes, are discussed in Chapter 3.

2.3.3 Flexible distribution paradigms - Networked client/server interaction and CDROM

STREAMS must function within a networked client/server environment. The reasons for this are clear. If STREAMS is to serve as an information dissemination platform, in large offices for
example, it must provide a centralized repository for events because of the massive storage requirements of maintaining redundant/replicated event libraries. Such centralization allows for quick notification and availability of new events as well as the flexibility to make updates and optimizations to events in a single location.

However, standalone playback as in the current prototype system still needs to be supported to allow for distribution media such as CDROM to be utilized. This is especially important considering the soon-to-be ubiquitous nature of the CDROM. Allowing for CDROM support gives users of the STREAMS system an intermediary point on the migration path to fully networked interaction.

2.3.4 Scalability
A limitation of the current prototype is lack of scalability. For example, as more and more CPU cycles and disk/network bandwidth become available, it makes sense for STREAMS to utilize the additional power to provide better quality event recreation. These issues are also discussed further in the section on stream versions.

2.3.5 Flexible codec support
This can be summarized as: “use software, but don’t ignore hardware”. STREAMS needs to be able to use software codecs to ensure that events can be played back on as wide a variety of platforms and configurations as possible. However, if playback-assist via hardware codecs is available, STREAMS should take advantage of the hardware. In the new STREAMS architecture, the same techniques used to support scalability are also used to provide this functionality.

2.4 Next generation STREAMS architecture
In order to realize all of the requirements for STREAMS, a new architecture for STREAMS has been developed. A high-level representation of this architecture is given in Figure 3. The system is now clearly separated into three distinct parts: a capture system, an event server, and presentation clients.

2.4.1 Capture System
The capture system is intended to record all events in real-time, generating appropriate information for searching tools, and providing different versions of streams as needed. It is the part of the
system that directly interacts with devices such as video frame grabbers, electronic whiteboards etc. The capture system will be responsible for creating different versions of streams. These versions will include stream data in different representations and different quality levels. The post-processing functions in the capture application will be used to accomplish this and other tasks, such as generation of search tool data and indices.

The capture system will also provide an integrated environment for specifying and capturing events. This environment will allow all capture activities to be coordinated from a single console. A plug-in architecture will be supported through which additional device types and stream types can be dynamically added. This will also enable device independence as well.

![Architecture of next generation STREAMS system](image)

**Figure 3 - Architecture of next generation STREAMS system**

2.4.1.1 Platform selection

The platform chosen for the capture system is the IBM PC compatible. There were several reasons for this choice. First and foremost, PCs are compatible with more capture devices on the market today than any other platform. In addition, PCs are extremely extensible. A standard PC usually contains 7 expansion slots which provide enough room for growth. This is a key requirement for
capture systems since a typical setup may interact with many capture devices. Another reason for using the PC platform is that it is the most widespread, and as a result, is by far the cheapest to obtain peripherals for. The capture system will typically need to have a large amount of disk storage available combined with fast I/O performance. Fast CPU performance would also be required if software codecs are to be employed. PCs suit these requirements at reasonable prices.

The choice to support a single platform for the capture system was deemed acceptable since it is not something that one would typically need to buy many of. In fact, it is totally reasonable to expect that capture needs can be adequately provided by a single PC that is appropriately configured with peripherals.

The capture system is intended to be a simplified, turn-key solution to capture. This means that it should be easy to set up, have few if any loose components, and be easily portable. A typical PC can be outfitted with capture hardware and large hard disks in a small case. This PC can then be quickly set up for capture via the capture application software. All that would be required is to plug in the external capture devices such as cameras and microphones into the capture PC, specify an event, and capture.

2.4.1.2 Capture environments and locales

Two different scenarios are possible for using the capture system in a room where an event is taking place. The system can be brought in specifically for the event, and set up along with all of the desired cameras and microphones. Another approach is to carefully outfit and dedicate a room to STREAMS capture. This room can contain a STREAMS capture system that is already setup for the specific arrangement of capture devices in the room.

2.4.2 Event Server

The requirements for the server are much more stringent than those for the capture system. The server must essentially serve as a repository for all of the captured events and play the events back to many clients concurrently. In an office environment, the server might be a powerful UNIX workstation with massive amounts of storage and throughput capacity. In a smaller environment
the server requirements can be relaxed a bit but are still fairly strict. This will typically be the most expensive part of a STREAMS installation\(^5\).

2.4.3 Presentation Clients

STREAMS clients need to run on a wide range of platforms. Clients will run on individual user workstations. Clients will be able to browse through the library of events stored on an event server and select one for playback. Typically, client workstation capabilities will vary widely in terms of computing power, hardware support, and network bandwidth available. Clients will negotiate with servers over the appropriate playback parameters of an event, and choose the best possible representation given the constraints.

2.5 The context of this work within the next generation of STREAMS

The focus of this work is to design a capture system that can support the capture requirements specified in section 2.4.1, as well as the overall STREAMS system requirements laid out in section 2.3. Chapter 3 is a discussion of the capture system design, and of one implementation of that design.

\(^5\) It is possible that improved technology in video servers and disk technology resulting from efforts in areas such as interactive television, will reduce server-related costs significantly. However, such potential benefits are several years away.
3. Capture system design and implementation

This chapter discusses the design and implementation of a capture system to support the requirements set forth for the next generation of STREAMS. The first part of this chapter discusses the design in general, from the point of view of the major entities within the system. Then there is a survey and discussion of the user interface for the capture system. Finally, there is a description of some of the implementation decisions made and difficulties encountered in creating the capture application - including the algorithms used.

3.1 Capture System Entities

The capture system deals with several distinct types of entities to accomplish its task. On the highest level, the capture system is dealing with the specification and definition of events. These events are comprised of different streams which combined together form the event record. Given an event and the streams that comprise it, the capture system interacts with different capture devices to actually realize the capture. Each of these entities will be examined in detail in this section. At first the discussion will focus on a higher level description and will then proceed to consider actual implementations.

3.1.1 Streams

Streams are the basic building blocks that make up events. A stream is a single-media record of one aspect of an event. It is important to emphasize that streams in the system are single-media. For example: there would not be a single stream to represent all of the actions of a speaker at a lecture. There would be separate streams for each aspect of the speaker’s presentation - e.g. an audio stream for the voice, a video stream for a visual view of the speaker, and perhaps a text stream with the actual words spoken. There are several reasons for keeping streams separated. First, it makes the handling of streams within the capture system easier if it can be assumed that each “stream” has a single source. Second, it offers more flexibility. It’s possible that a completely generic stream format that makes allowances for multiple sources would introduce limitations that might prevent future stream types or semantics to be easily incorporated. Finally, there are no clear advantages, other than housekeeping issues, to the multi-source approach.
3.1.1.1 Identification of streams

Streams must be identified within the capture system to differentiate them from other streams. Each stream is assigned a name that is used to uniquely identify a stream within an event. This name is a variable length string such as “Speaker’s voice”. This name is also intended to be used to identify the stream both within the capture user interface and client playback application.

3.1.1.2 Types of streams

In addition to the stream name, a stream must also be identified by the type of data that the stream represents. The stream type is necessary for several reasons. The most obvious reason is that the capture system must know how to handle the streams and how to interact with the capture devices that support the streams. Another reason is that the stream types are used to match up client capabilities with event composition. For example, an event may be comprised of several streams of different types. Before a client can play back a stream in a recorded event it must be sure that it can handle the stream type.

A Four-character code (FOURCC) is used to represent stream types. A FOURCC is, just as its description implies, four ASCII characters taken as a numeric value. An example of a FOURCC

<table>
<thead>
<tr>
<th>ASCII code</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>118 * 256³ +</td>
</tr>
<tr>
<td>i</td>
<td>105 * 256² +</td>
</tr>
<tr>
<td>d</td>
<td>100 * 256¹ +</td>
</tr>
<tr>
<td>s</td>
<td>115 * 256⁰ = 1986618483</td>
</tr>
</tbody>
</table>

(FOURCC code for ‘vids’)

Figure 4 - Calculating a FOURCC value

is ‘vids’. The numeric value of the ‘vids’ FOURCC is 1986618483. An explanation of how this numeric value is obtained is given in Figure 4. FOURCCs are used because they are more descriptive than simple integer codes, while retaining the measure of safety that integers provide over string representations. FOURCCs are also a fairly well established standard for identifying multimedia data types - they are used throughout the Microsoft Windows environment as well as in
QuickTime and Video For Windows (VFW). VFW will be discussed in greater detail in section 3.1.3.4.1.

3.1.1.2.1 Standard types
The capture system can handle many different types of streams. Most of these stream types will be dealt with dynamically at runtime. For example, a ‘wbds’ stream representing a captured electronic whiteboard session, is handled dynamically. However, the capture system handles audio and video streams directly. The FOURCC codes for these types are ‘vids’ and ‘auds’ respectively. These FOURCC codes were chosen because they are compatible with existing FOURCC codes provided by Video For Windows. The “direct” handling of audio and video streams will be further explained in the sections on system architecture.

3.1.1.3 Stream format
In addition to a type, streams also have a format which describes the particulars of the data representation. This format is dependent on the stream type and describes parameters of the stream that are specific to the captured media. For example, some parameters of video streams are image size, compression type, and compression quality. The stream format is what contains this information. Audio streams can be described by their sampling rate, sample size, compression type, and number of channels. This information is also captured within the stream format.

The stream format is a generic, all-encompassing mechanism for specifying details about data representation. With the exception of the audio and video stream formats, the capture system does not know anything about the contents of the format. For example, a format for a whiteboard stream is completely opaque to the capture system.

A set of possible stream formats is obtained from the capture devices that are used to capture the stream. Once we have decided on a stream type to capture, such as ‘vids’, the relevant capture devices are queried as to what capture formats are available. A format is selected, and stored with the stream. Once a format is selected, a stream is then associated with a single capture device which shall be used to capture that stream. Associating a stream with a single capture device, as opposed to using multiple capture devices to capture a stream, is a design decision that was made for simplicity.
3.1.1.3.1 Stream handlers

One issue that arises when adopting the stream format approach described is how to determine whether or not a particular format can be handled by a client. This is difficult because in general, the capture system does not know what is contained within the stream format, so it is impossible to ascertain anything about it. In order to deal with this issue, stream handlers are maintained for each stream. The stream format is used to obtain a handler from the capture device.

Like the stream type, stream handlers are also used to match up client capabilities with event requirements. For example, if a video stream in an event is only available in hardware assisted Motion-JPEG format, and a client does not possess the appropriate decompression hardware, the client will be unable to view the stream. Clients can determine what stream handlers are supported on their system via special configuration files present on the system. This is how the clients can negotiate with the server and agree on stream formats that can be handled by both.

Like stream types, stream handlers are referred to with FOURCCs. FOURCC's were chosen as the naming scheme for stream handlers because they are used by Video For Windows to identify codecs. For example, VFW uses FOURCCs to identify codecs such as ‘MJP’ (Motion JPEG) and ‘IV32’ (Indeo Version 3.2).

3.1.1.4 Stream Data - Samples

Streams contain the actual multimedia data that is captured. Because streams are dealing with digital capture, the data is organized into samples. A sample is one atomic “unit” of stream data captured at a particular instant in time. For video data, a convenient and logical sample is a captured video frame. Alternate samples could be a field of video, or even a single row of digitized video data. For audio data, a sample is usually one or two bytes of data representing the quantized audio level at a particular instant in time. For an electronic whiteboard, a sample might be a time-stamped representation of a complete stroke or a single point in a stroke.

With the exception of audio and video data, the capture system deals with stream samples abstractly and generally does not know exactly what a sample represents. The capture system obtains samples from capture devices and stores them in sequence in the stream.

3.1.1.4.1 Sample organization
One of the most important requirements on a stream is that it must be indexable by time. This is necessary to maintain synchronization over multiple streams. Related to this, we can separate stream data into two basic classes: periodic and aperiodic.

**PERIODIC** - Periodic stream data refers to data that has samples at fixed intervals of time. Examples of periodic streams are audio and video. In each of these streams, there is a specific number of samples per second, and a guaranteed steady bandwidth. Since the rate at which data is captured is precisely known, the absolute time into the stream can always be calculated from the number of samples and the rate.

**APERIODIC** - Aperiodic streams refer to data that is not continuous and is at unpredictable intervals. Individual samples must be time-stamped in order to be indexable by time. Examples of aperiodic stream data is stroke information from an electronic whiteboard or viewgraphs presented during a talk.

### 3.1.1.4.2 Stream rate

The stream rate specifies the rate at which samples are obtained/captured for a stream. The rate is given in samples per second. For periodic streams such as audio and video, typical rates might be 44,100 and 30 respectively. These rates correspond to 44.1 kHz audio and 30 frames per second video. A stream rate of 0 is used to indicate that the stream is comprised of aperiodic data.

### 3.1.1.5 Stream Versions

Ideally, a STREAMS client could select a recorded event and then have the best quality representation of that event delivered. However, in practice there are restrictions that make such delivery impractical. For example, network bandwidth is a limited resource, especially considering the extremely high demands of multimedia data such as audio and video. In addition to bandwidth restrictions, client and server computational power varies widely as well as hardware capabilities.

Let’s consider a typical scenario where a client requests a recorded event from the STREAMS server. This event contains a video stream of a speaker and an audio stream. The video stream was captured in as high quality as possible - say 640x480x24bit frames at 30 frames per second. This is clearly too much data for typical networks, and typical client hardware to handle.
Chapter 3

<table>
<thead>
<tr>
<th>Media Type</th>
<th>Transmission Bandwidth Requirements (bits/second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Styled Text</td>
<td>2400-9600</td>
</tr>
<tr>
<td>Speech Grade Audio</td>
<td>32k - 64k</td>
</tr>
<tr>
<td>Still Images</td>
<td>9600 - 128k</td>
</tr>
<tr>
<td>High Fidelity Audio</td>
<td>64k - 1.5M</td>
</tr>
<tr>
<td>Low Quality Compressed Video</td>
<td>100k - 1.5M</td>
</tr>
<tr>
<td>Medium Quality Compressed Video</td>
<td>1.5M - 6M</td>
</tr>
<tr>
<td>High Quality Compressed Video</td>
<td>6M - 24M</td>
</tr>
<tr>
<td>Uncompressed 640x480 30fps video</td>
<td>221M</td>
</tr>
</tbody>
</table>

| Table 1 - Bandwidth Requirements of Different Media |

Table 1 gives a comparison of some of the bandwidth requirements for typical media types. For comparison, some of the bandwidth characteristics for typical networks are given in Table 2. The data in these tables comes from a study done on building network based interactive multimedia systems [Blackketter92] done in 1992. Advances in technology have reduced requirements a bit, however these numbers are still relatively accurate.

Clearly, stream bandwidth requirements can potentially swamp a server and the network through which it delivers the streams to clients. As a result, the server needs to somehow scale the streams of an event to fit the resource constraints of the network and computing hardware. One possible approach is to have the server generate a lower-quality version of the stream when it’s requested and deliver that version. For video, this could mean scaling 640x480 pixel frames down to 160x120 pixels and generating a 10 frames per second version of the stream. Then this lower quality version could be sent. This approach however is impractical because it would introduce a huge delay while the new version is being created. A fixed, one-time delay of only a few seconds would be acceptable, but current technology is inadequate to realize such a quick conversion time.

An obvious optimization to this approach is to generate the lower quality versions on-the-fly as they are being delivered. For example, the server could read a 640x480x24bit video frame, convert it to an 8-bit 160x120 frame, and send out every third frame. This is a more reasonable
approach but is still extremely demanding. Converting a 640x480 frame to 160x120 can be done quickly in a naive fashion. However, techniques to do the reduction while maintaining high quality are much more taxing. In addition, algorithms to do 24bit to 8 bit conversion are non-trivial as well. Even if all of these conversion steps could be done instantaneously, there is another more significant issue. A STREAMS server can be expected to simultaneously service a number of clients. If the server has to scale each stream in real time as the streams are being delivered, for each client, it is easy to see that even the most advanced hardware will become quickly swamped. In addition to this, some conversions (such as 16-bit to 8-bit audio conversion) cannot be done in real-time with satisfactory results.

There are some other significant problems with both of these approaches. Each approach relies on the server doing the conversion. This entails that the server must know and understand the data representations for each stream - an undesirable proposition since it drastically increases the complexity of the server.

Since it is currently technologically impossible to adopt either of the aforementioned approaches, and because it increases the complexity of the server, a compromise solution was chosen. Several predefined quality versions of each stream will be generated and made available on the server. If

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Data Transmission Bandwidth (Bits/second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>POTS (Analog Phone Line)</td>
<td>2.4k-28.8k</td>
</tr>
<tr>
<td>Basic Rate ISDN</td>
<td>64k - 128k</td>
</tr>
<tr>
<td>Primary Rate ISDN</td>
<td>1.5M</td>
</tr>
<tr>
<td>ADSL/HDSL</td>
<td>1.5M</td>
</tr>
<tr>
<td>Token Ring</td>
<td>4-16M</td>
</tr>
<tr>
<td>Ethernet</td>
<td>10M</td>
</tr>
<tr>
<td>Fast-Ethernet/ ATM</td>
<td>100M</td>
</tr>
<tr>
<td>FDDI, FDDI-2</td>
<td>100M</td>
</tr>
<tr>
<td>Broadband ISDN</td>
<td>150M</td>
</tr>
</tbody>
</table>

Table 2 - Bandwidth Characteristics of Typical Networks
none of these predefined quality versions are satisfactory for delivery to a specific client, then the client will not be able to view that stream. The criteria for determining whether a stream is viewable are bandwidth requirements and processing requirements.

Each of these predefined versions will be generated on the capture system. This is an advantage because the capture system is what generated all of the streams in the first place. Through the use of stream handlers, the capture system already has the knowledge necessary to interact with streams at the sample level - thus no additional complexity is required.

3.1.1.5.1 Support for different hardware and software codecs

The capture system will not only be used to create lower quality versions of existing streams, but also to create alternate versions of the streams based on different codecs. For example, video streams can be converted to a variety of different compression formats to suit different client playback capabilities. This is an extremely important capability considering that client capabilities vary significantly from platform to platform.

This is also critical in allowing capture at the highest rates possible. When capturing a stream, it is desirable to capture in as high a quality as possible to provide the best source material for conversions. To accomplish this, typically sophisticated hardware-assisted compression and capture boards need to be employed. It is highly unreasonable though to require that such hardware be present in all clients. Such hardware might not even be available for all clients. A technique for converting from such high-performance hardware-assisted formats to more widely available software-only codecs is thus extremely valuable. By the same token, different versions to support other hardware assisted formats (such as provided by some Motion JPEG and Indeo boards) can also be generated as well.

3.1.1.5.2 Support for preview and focus

It should be noted that adapting to limited resources is not the only reason for having different quality versions of streams. The client interface will typically present preview versions of streams and allow the user to decide what to emphasize in a "normal" focused version. The preview versions, which might be thumbnail views of a stream suitable for conveying basic information about activity in a stream, can be generated using the same versioning scheme on the capture system.
3.1.1.5.3 Streams & versions

Streams have a number of versions associated with them. Each of these versions represents the same information, but at varying quality levels. When we speak of a stream, we are implicitly referring to the capture version of the stream, which is necessarily the highest quality version\(^6\). All versions possess the stream format, handler, and rate attributes.

In addition to these attributes, each version can also give information about its bandwidth requirements. At this point, no consideration is given to computational requirements. The bandwidth requirements, combined with the format and handler, can be used to determine whether or not a client can view a stream in a recorded event.

3.1.2 Events

Events are the unifying thread throughout the entire STREAMS system. The capture system records events and prepares the events for use. The server takes the events from the capture system and allows clients to request them for playback. An event keeps track of explanatory information such as date and location, the streams within the event, and the participants that make up the event.

3.1.2.1 Event as a collection of streams

First and foremost, an event is a collection of streams. There is no absolute limit on the number of streams in an event. The makeup and types of the streams are completely specifiable by the user and are only limited by the capture hardware used to record the event.

3.1.2.1.1 Canonical forms of events

There are several standard configurations of events which can be helpful to think of as a starting point. A configuration of an event is a default template which specifies a standard number of streams, their type and format, duration, etc. These templates can be valuable in speeding up the event specification process. They are also especially valuable for novice operators of a capture system. Typical event configurations include:

---

\(^6\) This is assuming that no form of media enhancement is used. Media enhancement technologies, while applicable, are beyond the scope of this research.
LECTURES - A lecture is an asymmetrical form of information exchange. A typical lecture would have a medium quality video stream of the speaker and a low frame-rate video stream of the audience. There would also be an audio stream of the speaker, and an audio stream of the audience to identify audience questions. Finally, there could be a viewgraph stream containing high-resolution images of the viewgraphs used.

MEETINGS/DESIGN REVIEWS - Such events are typically much more symmetric in their information exchange paradigm. By default, there would be two medium quality video streams to capture all of the participants in a room. There would be a single audio stream - although others will typically be added if possible. In addition, a standard feature in such events would be an electronic whiteboard stream.

In addition to these canonical forms of events, others can be defined which might be tailored to the specific needs of an organization, to incorporate some other standard visual tools, or to match the capabilities of a specially-outfitted room. For example, consider a special room that has an electronic whiteboard, several PDAs for note-taking, a viewgraph projector, two video cameras and two microphones located at opposite ends of the room to aid in speaker identification. A template could be defined for such a room so that anytime an event was taking place in that room, that template could be used as a default configuration.

3.1.2.2 Event information

There are several bits of information that may be useful to maintain about an event. This information can be used to determine whether or not a user wishes to view a given event.

NAME - A string used to identify the event in user interfaces - e.g. “A Primer on Video On Demand”.

DATE - The date that the event took place. This is important to keep track of for informational purposes and to help identify a past event. It is also useful in constructing event browsers - where it can be used as a sort or selection criteria on a group of events.
DESCRIPTION - A longer, more detailed, synopsis of the event. This is also useful for user interface construction.

NUMBER OF STREAMS - This refers to the number of streams that makeup the event. This does not include any additional stream versions generated.

LOCATION - Where the event took place
DURATION - How long the event is

3.1.2.3 Participant involvement
One of the most important things to keep track of with the event is some kind of record of the participants involved in the event. In addition to the number of participants it is useful to maintain more detailed information about each participant. For example, along with a participant's name, it is useful to include information on how to contact the participant after the event - typically an email address or phone number.

3.1.2.4 Implementation issues for events
In order to represent events within the capture system, a C++ class, Event, was designed and implemented (see discussion in section 3.3.1). There were several requirements on these Event objects that needed to be considered. The most obvious requirement is that the Event objects need to maintain all of the information about the events. This followed easily and naturally. However there were some other issues that arose.

3.1.2.4.1 Persistent event storage - atomic vs. distributed
Some mechanism for saving event specifications to disk needed to be devised. The first issue that needed to be addressed was whether or not the actual media content of the streams would be incorporated into the same file. This approach yields a single file for an event which is advantageous because it reduces housekeeping duties and minimizes organizational complexity. However, there are a couple of problems with this. Considering the large size of the files containing the multimedia data for the streams, incorporating all of the stream data into a single event file would produce a mammoth file. Such a file would be more difficult to handle because it would necessarily have to be on a single filesystem. It is quite possible that all of the combined storage requirements of an event could exceed a user's single filesystem. Another reason that the
monolithic file approach is unattractive is that it prevents the use of existing tools to view and edit media files. For example, consider AVI files which are generated by the capture system to store video. There are many standard utilities and applications that can view and manipulate AVI files. It was considered very important to remain compatible with such tools because it is beyond the current scope of STREAMS to provide functionality such as video editing, compositing, etc.

Finally, there is a great deal of overhead involved with traversing a single event file. A user would never play back all of the streams, and stream versions, in an event at the same time. Even if stream data was interleaved within such a file, much effort would be spent skipping over unwanted stream samples. For these reasons, an event format was chosen which maintains links to the actual multimedia data which may be spread among several filesystems.

A related issue is the location of additional information needed for the construction of searching tools. Some media files do not provide adequate information about stream activity relative to the entire event. For example, video and audio media files provide no support for embedding info on which participant in an event is “active” at any given moment. Such data needs to be maintained in addition to the event information and stream data. This information is incorporated into the event files. Another approach would be to create an additional “search tool” file for each stream, however this would add considerable effort to file management and housekeeping chores. Some media file formats (such as AVI) do allow embedding additional information in the files which could be useful for storing search tool data. However, this is not the case in general, and most media file formats will be opaque to the capture system anyway (see section 3.1.1.3).

3.1.2.4.2 Extensibility

Another issue which was important to the design of an event persistence specification was future extensibility. It is difficult to anticipate all of the information that will be useful to maintain about events so a file format that is flexible is crucial. In addition, since searching tool data will be stored in the event files, it needs to be easily maintained. For example, as technology to generate searching tool data improves, new information would need to be tracked in the event files to make use of the new technology, without obsoleting the old techniques. Such might be the case when technology exists to provide voice recognition or speaker identification. The common thread to all of these concerns is that the new file formats must not obsolete the old ones - older formats should continue to work transparently.
A proprietary binary file format was considered but eventually rejected because it was deemed too constraining on future growth. A tagged file format was adopted instead [EAIFF85]. A tagged file format is a file where data are organized by a “tag” - a label which is used to identify a particular chunk of data. Tagged file formats have the property that new tags can be added to the files at any time to support future extension. If tagged file readers are designed properly, in that they ignore unknown tags, the new tags do not render the file unreadable.

Consider an example of an application A that can read a tagged file foo. If application B, which can read foo adds some new tags to keep track of application B-specific information, application A will still be able to read the file. Application A will just ignore the extra tags that were added.

### 3.1.2.4.2.1 The RIFF file format

The Resource Interchange File Format (RIFF) was chosen as the standard for event files. RIFF is a tagged file format, designed by Microsoft, loosely based on the Tagged Image File Format (TIFF) used for storage of bitmaps. RIFF is supported, although only at a low level, in the Microsoft Windows environment through a small set of APIs. RIFF is not a specific file-format per se, but is a general style of organizing files. Examples of some RIFF files are AVI files (Audio/Video Interleaved) and WAV files (digital audio).

RIFF organizes its files into chunks. All data is stored in chunks. There are three types of chunks: RIFF chunks, LIST chunks, and data chunks. RIFF and LIST chunks are special because they can contain other chunks, referred to as subchunks. A RIFF chunk defines a form specification and is the top-level chunk for a file. All chunks are subchunks of the RIFF chunks. Chunks are identified by a FOURCC.

### 3.1.2.4.2.2 The Event RIFF specification

See Appendix 1 for a description of the current event file RIFF specification.

### 3.1.2.4.3 Cross-platform compatibility

One critical requirement on the event files is that they be readable on a variety of platforms. This is important because while event files will be generated on Windows capture machines, they need
to be read on servers that may be any other type of platform. Such a requirement might argue for
the use of text files as the format for event persistence. However, text files would be too
cumbersome for maintaining search tool information, and too inefficient as well.

One problem with using a binary file format such as RIFF is deciding how much resolution to use
in defining the chunks. For example, if we know that we need to keep track of three integers, is it
best to put each integer in its own chunk or combine the three integers into a single chunk. In the
interests of simplifying parsing complexity, the latter solution seems like a better approach.
However, there is a problem with that technique. The problem manifests itself when we try and
write out entire structures of data in a single step. Consider a hypothetical structure of type
StreamInfo with 4 elements, as in Figure 5. Such a technique as described in the figure is
straightforward to use, but is potentially incompatible across platforms. To understand why, we
must consider how compilers organize structures in memory.

typedef struct _streamInfo {
    unsigned short streamNo;
    unsigned long startTime;
    unsigned long duration;
    char streamName[40];
} StreamInfo;

TO WRITE TO DISK:
write(file, &streamInfo, sizeof(streamInfo));

TO READ FROM DISK:
read_chunk_data(file, buffer, &dataSize);
memcpy(&streamInfo, buffer, dataSize);

Figure 5 - Reading and writing a structure

Ideally, a StreamInfo structure will take up exactly 50 bytes in memory - 2 bytes for the ID, 8 for
the times, and 40 for the name. However, on some processors, it might be more efficient to lay out
words on 2 or 4 byte boundaries. For example, in a machine where 4-byte alignment is used, the
streamID would be followed with 2 bytes of padding to align the startTime on a 4 byte boundary.
If a StreamInfo chunk in an event was written on such a machine, in the way described in Figure 5,
the StreamInfo chunk will not be portable to all machines. If such a structure was read as in this
figure on a machine with 1-byte alignment, the program will mostly likely crash and return a bus
error or similar fault.
There are several possible solutions. Structure serialization could be avoided altogether and each chunk could contain a single datum. This option is undesirable. If it is possible to programmatically ensure a standard byte-alignment (such as 1-byte alignment) for event serialization, then we could avoid this problem. Currently, such techniques are available via inline pragmas but are not universally available. A simpler technique is to just force all words to be 4-byte unsigned longs. In this way, the structure alignments are universal, and there is no real penalty in program complexity. The only real drawback to this approach is that space is potentially wasted. However, relative to the mammoth amounts of space required by the media files themselves, the extra space is considered insignificant.

3.1.3 Capture Devices

Capture devices are the physical devices that are used in the actual recording of a stream. Capture devices can be divided into two broad classes - source devices and sink devices. Examples of source capture devices are video capture boards, digital audio capture cards, electronic whiteboards and personal digital assistants. Examples of sink capture devices are different kinds of hard disks and displays. Networks could also be thought of as a sink device - for remote observation of capture, for example. Currently, the only sink capture devices supported are hard disks.

3.1.3.1 Basic capture device attributes

Capture devices possess a range of attributes that are useful throughout for the construction of capture-system user interfaces. For example, capture devices have a name which can be used to identify the device. This name is typically something like “Microfield Graphics Softboard”. In addition to the device name there is also usually a version attribute. This is used to account for different versions of drivers used to support capture devices. Capture device drivers also provide an icon to provide a pictorial representation of the device to an operator of the capture system.

In addition to user-interface related information, capture devices also specify a stream type that is captured. This stream type is the same as the stream type described in section 3.1.1.2 and is used as such. Examples of stream types that a capture device might support are ‘vids’ and ‘auds’.

---

7 4-Byte alignment is sufficient to ensure alignment on 32-bit machines, the most prevalent type of machine available today. On 64-bit machines, 4-byte alignment may or may not be sufficient.
Capture devices are either uninitialized, initialized, or shutdown. A capture device must be initialized to initiate capture on the device. The initialization process varies and is dependent on the particular capture device. For example, initialization of a video capture card involves ensuring that a valid video signal is present, and that the card is synched with the signal. Initialization of an electronic whiteboard may require downloading of DSP code to prepare the whiteboard scanners for capture.

3.1.3.1.1 Networked capture devices

Capture devices can be located in different capture systems connected via a network. Each capture system is identified with an ID that is also associated with each capture device. A master capture system can present the operator with a list of all capture devices available not only locally, but also in slave capture systems as well.

3.1.3.2 Source capture devices

Source capture devices add additional functionality to that present in all capture devices. Source devices have a number of sources, or distinct inputs. A source capture device can have one or more simultaneous capture channels. Most video capture cards have a single capture channel (i.e. only one frame can be captured and compressed at a time). Stereo audio cards have two simultaneous capture channels as they can capture both a left and a right channel simultaneously - these can be used to capture two monaural audio streams at the same time.

Source capture devices can provide descriptions of the sources. Sources are numbered from 0 to the number of sources - 1. More descriptive names for these sources are useful in constructing the capture systems user interface. For example, a Creative Labs VideoBlaster RT300 has 4 sources - "Composite In 1", "Composite In 2", "Composite In 3", and "Svideo In".

3.1.3.2.1 Interleavable Sources

A capture device may be able to do interleavable capture among its sources. This means that during real-time capture, it is possible to switch between sources. In this manner, a card that has 3 sources can "simultaneously" capture 3 different streams by switching between the sources during...
capture and “stitching” the captured data together. It should be noted that the interleaved sources are still sharing a single capture channel and as such are limited to the maximum bandwidth on that channel.

3.1.3.2.2 Capture format

A source device has a capture format which specifies the particular representation used for the captured data. For example, capture format for a video capture card may be represented by “320x240x24 Motion JPEG”. Capture devices usually have a number of different formats available, to be chosen by trading off quality and bandwidth requirements.

The capture format is also used as the stream format for a captured stream as described in section 3.1.1.3. A capture device always has a particular format that it is set to capture with, although this can be changed - if a capture format is not set by the user, a default format will be used. When it starts capture, it captures data in the format that it happens to be set. The capture format applies to a capture channel and to all sources being captured on that channel.

A capture format, like a stream format, is an opaque data type and basically looks like a sequence of bytes. The size of the format, in bytes, is obtainable however. Text descriptions of a format such as “320x240x24 Motion JPEG” can be obtained for use in user interface construction.

3.1.3.2.2.1 Source-specifiable devices

Most capture devices do not allow a change of capture format once capture has begun. However, some capture devices do allow switching the capture format during real-time capture. Such devices are said to be source-specifiable. This capability is most useful when combined with interleavability. In this case, you can “simultaneously” capture a number of streams each from a different source and with a different format. In the case where interleavability is not available, source-specifiable capture devices might still be useful if some knowledge about stream content is available. If so, then intelligent decisions on capture format can be made throughout the life of a stream. It should be noted however that in the current version, a stream has a single format throughout its entire duration. Thus, source-specifiable devices are not useful without interleavability as well.

3.1.3.2.3 Capture rate
A capture device can specify a list of possible capture rates. A video card may specify a range of capture rates from 1 to 30 frames per second. This list may be empty, as in the case where we are dealing with aperiodic streams. Audio streams may specify their capture rate in the capture format and may or may not provide a list of rates. The list of capture rates is used to setup capture devices and within the capture system user interface.

3.1.3.2.4 Capture bandwidth

Given a capture format, and possibly a capture rate, an estimate of the required bandwidth necessary to sustain capture on the device can be calculated. This estimate can be used to allocate resources or to determine whether sink capture devices such as hard disks can bear the load.

3.1.3.3 Interacting with Capture Devices - the STREAMS prototype

A critical problem with the STREAMS prototype capture system was its strict reliance on specific capture hardware. There were two different versions of the STREAMS prototype, each with major differences in the code base. The first version ran on SunOS 4.1.x systems equipped with VideoPix frame grabbers. A newer version of the prototype was designed to run on the Sun Solaris Operating System with SunVideo video capture cards. The code to capture from each particular frame grabber is statically linked within the prototype. This had the unfortunate side-effects of difficulty in incorporating new hardware and potential for obsolescence as better cards with possibly incompatible interfaces are developed.

3.1.3.4 Device Independence

The new capture system intends to be device-independent in the sense that the code for the capture system should be separated as much as possible from specific hardware interactions with capture devices. The device-independence is required to include video and audio capture hardware as well as any other types of hardware that can potentially be considered useful to STREAMS.

3.1.3.4.1 Video capture devices

At the time of this research, a relatively new standard for interacting with video capture hardware was emerging for the Windows/PC platform - the chosen platform for the new capture system. This is the Video For Windows standard. Video for Windows ("VFW") provides an interface for capture applications and video capture devices that allows the ideal of device-independence to be achieved. Essentially, VFW defines a standard device-driver level interface for video capture
cards. As long as capture applications adhere to this interface, and vendors of video capture cards provide drivers for their boards, device-independence (at least for video) can be realized. The number of VFW-compliant boards on the market is increasing and currently stands at about 60% of the available boards.

It should be noted that there are some disadvantages involved with choosing VFW as the abstraction layer for video capture devices. The most significant of these is the overall quality level and performance hits. The quality level that VFW provides is adequate for most purposes but is noticeably less than ideal. The reason for this is the tradeoff involved with accommodating a wide range of capture devices versus optimizing for a few. Similarly, performance in areas such as frame rate is also negatively impacted by the generalization.

For these reasons, other systems that deal with digital video such as the Avid MediaSuite opt for custom video abstraction solutions and forego VFW. However, for the purposes of STREAMS, VFW presents the best option. Using VFW is a much simpler proposition since much of the work of defining interfaces has been done already. This allows for more time on STREAMS-related issues. In addition, since VFW is continually evolving and improving, and there are already many organizations working at improving it, it is reasonable to assume that the effort should be done elsewhere as opposed to within the work of this thesis. Finally, while the performance and quality levels of VFW are not sufficient for broadcast-level work, it is more than adequate for STREAMS.

3.1.3.4.2 Audio Devices

Microsoft Windows has provided services for device-independent audio since version 3.1. These services are part of a well documented standard and are sufficient for STREAMS purposes. There are some limitations in the system though. Most notable is the limitation of a single capture channel per audio device. This means that if an audio capture device can digitize several independent channels of audio simultaneously, only one channel can be utilized at a time. This is a significant issue for STREAMS since it is likely that several channels of audio may be desired for an event. In addition, the Windows audio standards restrict sampling rates to one of three values, 11.025 kHz, 22.05 kHz, and 44.1 kHz. Since STREAMS is primarily dealing with speech quality audio, all of these rates can be considered overkill. Despite these limitations, the Windows audio standards will be used for STREAMS. The reasons are largely because they exist and are the
easiest approach. In addition, compression techniques such as Adaptive Differential Pulse Code Modulation (ADPCM) can be used to mitigate the effects of the overkill in the sampling rates.

3.1.3.4.3 Other Devices

This category of capture devices includes anything that is not encompassed by video and audio standards, such as electronic whiteboards, personal digital assistants, etc. As a result of the broad scope of these devices, there exists no standard to effectively deal with them. For example, consider the Microfield Graphics Softboard, an electronic whiteboard used extensively during thesis research. The Softboard connects to a PC via a serial port and has a custom protocol used to control the whiteboard and communicate information about stroke activity. Microfield Graphics provides a library of C-language routines to deal with the Softboard. These routines are designed to be incorporated into capture applications. Other vendors of electronic whiteboards provide similar toolkits. The STREAMS prototype was retrofitted to work with the Softboard via the library of routines. This was simple enough, however it resulted in a system that is dependent on the Softboard. If a different whiteboard was needed, the code would have to be rewritten to use a different whiteboard toolkit.

These problems are not limited to electronic whiteboards. Similar issues arise when trying to interface to PDAs such as the Apple Newton or the Casio Zoomer. Clearly what is needed is a generalized notion of a capture device. This interface for dealing with capture devices needs to specify exactly what a capture device is, how to control it, how to capture from it, all while being sufficiently generic to encompass a wide range of devices. Such an interface has been designed and will be discussed in the next section.

3.1.3.5 Design for device independence - The STREAMS interfaces to capture devices

There are already preexisting standards to support device-independence for audio and video capture devices. However, these standards do not provide all of the information that is necessary for the devices to be optimally utilized in the STREAMS capture system. For example, a VFW video capture driver has no facilities to programmatically control what source the card is capturing from. The same is true for audio drivers.

In order to provide information to the capture system that is necessary for STREAMS in a device-independent way, a two-tiered driver approach is proposed. The first driver ("level 1") is the
regular VFW or audio driver that directly interacts with the hardware. The second driver is called a *STREAMS* driver and provides additional device-dependent information to the capture app via a device-independent API. The *STREAMS* drivers interact with level 1 drivers and also return handles to the level 1 drivers so that the capture application can interact with them as well. This is for performance reasons. It is considered unacceptable to add an extra level of indirection between the drivers during real-time capture since this will introduce unnecessary function call overhead. A depiction of this two-tiered driver architecture is given in Figure 6.

**Figure 6 - Two-tiered driver architecture**

3.1.3.5.1 *STREAMS*Video

The function of the *STREAMS*Video driver is to provide additional information to the capture app that is not available from the standard VFW drivers. One oversight is a lack of obtaining the maximum frame rate available with the capture card. The most significant omission in VFW driver functionality, however, is the lack of multi-source support. More specifically, there is no way to programmatically determine the number of sources a video capture card has and to switch between them. This capability is currently provided through the use of *driver-supplied dialogs*\(^9\) which a VFW driver provides to allow the user to make the changes. This works fine for most applications but is not acceptable for the *STREAMS* capture system. The reason that this is unacceptable is that the capture system needs to have intimate control over what source the card is capturing from. Consider a capture card with 4 inputs for cameras and a maximum capture rate of 30 frames per second. Using the *STREAMS* capture system, it would be nice to be able to simultaneously capture up to 4 streams of video. For example, a video stream of a speaker talking at 15 fps, a stream of the crowd at 10 fps, and a stream of the viewgraphs at 1 fps. In order to

---

\(^9\) A *driver-supplied dialog* is a dialog that a driver can be instructed to display to obtain information from the user.
accomplish this with a single capture card, we must resort to interleaving video frames from different sources \(^\text{10}\). In this case, we could set up the card to capture at 26fps, capture 15 fps of the speaker, 10fps of the crowd, and 1 fps of the viewgraphs. The technique for selecting which source to capture from given a set of streams and their fps requirements is discussed later. In order to do this interleaving, we must have a way to programmatically switch to another source during real-time capture. Such a requirement is obviously not satisfied by a driver-supplied dialog. The method for switching also must not introduce any significant delay because it is potentially in the critical path during capture.

It is quite possible that a later revision of the VFW driver standard will introduce the functions necessary to support interleaving of video. However, in the meantime, these APIs will be provided by the STREAMSVideo specification. STREAMSVideo will also provide information about the maximum frame rates available for a capture card.

The STREAMSVideo driver is intended to be vendor-supplied with the capture card. The idea is that vendors who would like their capture cards to provide interleavable video support for STREAMS will write a STREAMSVideo driver and ship it along with the VFW driver. However, a STREAMSVideo driver is simple enough to write that is more likely, at least in the early stages, that the user of the card will have to write it. Most VFW drivers do in fact provide techniques for switching input sources. However, these techniques are device-dependent and vary from driver to driver. The instructions on how to access these APIs are all that are necessary to write the STREAMSVideo driver. These device-dependent APIs are usually accessed through the videoMessage(functionID, clientData) function interface - where functionID specifies the device-dependent function such as SWITCH_SOURCE, and clientData might specify the source number to switch to.

It is possible that this entire problem could be avoided if it was just required that there be a separate frame grabber for each stream of video that you might want to capture. This however is an impractical restriction. First, it is currently almost impossible to get two capture cards to coexist in the same system together. It is definitely impossible to get more than two. The reasons

\(^{10}\) If a video card existed with two simultaneous capture channels, we could avoid the need to interleave. However, no such cards currently exist.
Capture System Design and Implementation

for this have to do with interrupt conflicts and bus-contention issues that are present in current day operating systems such as MS-DOS and Windows 3.1. Future OS's such as Windows 95 may eliminate such difficulties. However, even if such difficulties are eliminated, free expansion bus slots are typically at a premium and it is unlikely that users of the capture system will want to sacrifice many slots just for video capture cards\textsuperscript{11}. Finally, it is not cost-effective to require multiple capture cards. Consider that each card may support 30fps of compressed-video capture. Much of this will be wasted since current bus bandwidth levels are being pushed to the limit. It is already a fairly demanding task to capture 30fps over a standard ISA PC bus. This may also change in the future, as more sophisticated busses such as PCI become widespread.

3.1.3.5.2 STREAMSAudio

The function of the STREAMSAudio device driver is almost exactly analogous to the function of the STREAMSVideo driver. STREAMSAudio drivers are designed to work with audio cards that have multiple sources. However, at the time of this research, no such cards were available within a reasonable price range. As a result, no STREAMSAudio drivers were implemented in the current realization of the capture system.

3.1.3.5.3 STREAMSGeneric

STREAMSGeneric is a driver specification for a generic capture device. It is intended to be used by all non-audio and non-video capture devices and streams. The distinction is made because audio and video are typically relatively high-bandwidth stream types. STREAMSGeneric can be made to handle such data but is better suited to relatively low-bandwidth situations. The reason for this is that STREAMSGeneric drivers cannot offer the performance guarantees necessary to ensure smooth operation of the capture system. This issue is discussed fully in section 4.2.1. Examples of applications where a STREAMSGeneric driver is appropriate are Electronic Whiteboards and Personal Digital Assistants. These are low bit-rate streams where the data rate is very bursty.

STREAMSGeneric allows the capture system to interact with devices in a uniform manner. It provides an API of functions that are used during real-time capture and for accessing captured data. The capture system can support multiple STREAMSGeneric devices in a system.

\textsuperscript{11} Consider the typical PC-compatible computer has 7 slots. Three slots are typically used for a graphics display card, I/O port card, and disk drive controller card. A SCSI card (for large disks) takes up another slot. A sound card takes up a fifth slot, leaving two more slots available. These slots can be used for video capture cards, network cards, etc.
3.1.3.5.3.1 The STREAMSGeneric Capture API

The STREAMSGeneric API is a standard set of messages that define the interactions with generic capture devices. These messages are described in Appendix 1.

3.1.3.6 Implementation of STREAMS Drivers - A realization of device independence

A STREAMS Driver is a special kind of Dynamic Link Library (DLL). A DLL is a software module that is dynamically loaded in at run-time and then acts just like a library of routines that might have been compiled into an application via a statically-linked library. By deferring the linking of calls to run-time, we introduce the possibility of software plug-ins. This is the mechanism that the STREAMS Capture system uses to realize a plug-in architecture for capture devices and stream types.

STREAMS Driver DLLs follow the Windows Installable Driver Interface standard. This is a specification that is designed to provide a uniform way of accessing and controlling devices in the Windows operating environment. It requires a base level of functionality that all installable-drivers must provide such as opening, closing, power-on, initialization, installation, and configuration. In addition to these base-level functions, the installable driver interface also allows for custom-APIs that are more specifically relevant to the particular device.

There are several advantages to using the installable driver interface. The most significant of these is that it allows device drivers to be dealt with by the user of a system in a uniform manner. For example, DLLs that obey the installable driver interface can be installed and configured via the Windows Control Panel applet. Another advantage is that it is a well-defined standard for writing drivers which simplifies driver writing. There is plenty of sample-code on how to write an installable-driver. Conforming to the standard is also playing it safe for the future by ensuring compatibility with as-of-yet undeveloped tools.

There are some disadvantages to using the installable driver interface. Some of the functionality required of the interface is unnecessary for STREAMS. Most significant however is that it forces an extra level of indirection between the client of the driver, the capture application, and the driver itself. This extra level of indirection is manifested through the use of an extra Windows configuration file and introduces potential operator error. However, this disadvantage is heavily
outweighed by the advantages especially since much of the installation of new drivers into a STREAMS capture system will be largely automated.

3.1.3.6.1 Problems with and alternatives to using DLLs

There is one major drawback to using DLLs. DLLs and the installable driver interface are specific to the Microsoft Windows operating system. Thus the code to interact with capture devices and the streams that they generate can only run in Windows. While this may not seem like a major restriction on the capture system, since it is by design a Windows only solution, it does in fact introduce a deeper problem. The problem arises when you need to access the captured stream on a non-Windows platform - something that the STREAMS server might potentially need to do. A vendor of a capture device will provide a STREAMS driver to control the device. This driver is also what manages storing a captured stream to disk and all of the peculiar file formats that it might use. The different STREAMS subsystems have no idea about the particulars of these file formats - they access the streams through the standard interface of STREAMS drivers. If STREAMS drivers are DLLs then streams can only be accessed through Windows platforms. This issue is still being addressed.

Using a DLL is not the only approach to isolating device-dependent code, while maintaining a plug-in architecture. Another possibility involves the use of separate *device-driver processes*. These are distinct executables that run in a separate application space and communicate with client applications via some form of inter-process communication (IPC) such as RPC or shared-memory. The method of IPC must also specify a protocol or API for communication. By taking this approach, the device-dependent code that is used to deal with streams can be utilized on all platforms. A comparison of the different approaches is given in Figure 7.
While this technique seems to solve the DLL problems, it introduces a number of other issues. First, in order for it to be feasible at all, the form of IPC must be standardized. Such a standardization is not available, or commonplace, among all platforms. More importantly, running several independent processes incurs a significant overhead. This is especially significant during real-time capture where the capture application needs to maintain tight time-synchronization and control over the streams being captured. The penalty resulting from crossing application boundaries has the potential to be too large and will interfere with the capture process. Of course, lightweight IPC mechanisms can be used to mitigate the cost, however, these are not readily available.

There is also a practical consideration that argues against the device-driver process approach. Windows 3.1, the version of Windows that the capture system is currently being developed for, is a cooperative multitasking environment. The lack of preemption means that a single errant process can bring down the entire system. As a result of the need to trust many distinct modules, the process approach introduces much more potential for failure. Windows also does not have a clean "process" metaphor like UNIX, for example.

### 3.1.3.7 Use of STREAMS drivers

A key point with the STREAMSVideo and STREAMSAudio drivers is that their existence is not required for the operation of video and audio capture devices. If they are available they allow for more functionality in the form of interleavable streams. However, if they are not present, a default
level of functionality is still provided. For example, consider a STREAMS capture system set up with a relatively new video capture card that does not yet have a STREAMSVideodriver. This capture card, as long as it is VFW-compliant can be used with the capture system. However, the system will only be able to capture a single video stream irrespective of the number of video sources available on the card. In addition, no extra work beyond installing the standard VFW drivers is necessary - the capture system will automatically recognize the card. This fallback compatibility is considered important because it promotes the goal of device-independence, even when specialized drivers are not available. If at some point, a STREAMSVideodriver is written for the new card, it can easily be incorporated into the capture system. Once the driver is available, the capture system will be able to capture as many simultaneous video streams as the capture card has inputs - limited only by the maximum capture bandwidth of the card.

A STREAMS driver is a special file of the form “filename.drv” which is associated with a particular piece of capture hardware. Typically, STREAMS drivers are provided by vendors of capture equipment and are installed into the STREAMS directory - the directory in which the STREAMS capture application executables exist. They are incorporated into the capture system via the Windows system.ini file and the capture system’s capture.ini file. These are two files which specify configuration information for Windows and STREAMS capture. There is a section in capture.ini which lists the names of all of the STREAMS drivers that will be used in the capture system. These names are then used by Windows installable-driver API’s to look up the associated driver files in system.ini. The capture system requests a specific handle to a STREAMS driver by giving Windows the name of the driver. The system then uses this handle to communicate with the STREAMS driver.

It should be noted that there is no information about what kind of STREAMS driver a particular entry in the INI files may be referring to. This knowledge is obtained directly from the STREAMS driver via a standard API that all STREAMS drivers support.

3.1.3.8 Plug-In Device Support

Because of the use of dynamic linking to support capture devices, the STREAMS capture system supports a plug-in architecture for new capture devices. This plug-in architecture is realized through the use of STREAMSGeneric drivers and the STREAMSGeneric API. A consequence
of this is that future devices that can not even be foreseen can be incorporated into the capture system without any change to the source code.

3.2 Capture System User Interface

The capture system's user interface is a critical component of the capture system. The user interface acts as a control console for the entire system and provides a level of integration that was wholly absent from the STREAMS prototype capture utilities. The interface is a GUI designed within the Microsoft Windows environment. The GUI consists of a main working area which consists of all of the major views and windows used to specify, set up, and capture events. A view of the main window of the capture interface is given in Figure 8. In addition to the main window, the interface consists of a pull-down menu, toolbar for quick access to functions, and a pictorial view of capture devices available in the system.

Figure 8 - Capture Application Main Workspace
3.2.1 The Event Window

The event window is the central feature of the main workspace. It is a view of the current specification of an event. The capture application is a Multiple Document Interface (MDI) Windows application which means that it can handle multiple open documents simultaneously. The documents in this case are the events that the capture system is working on. The event window conveys information that is given about the event such as event date, name, description, etc. In addition to this, the event window also gives a view of the streams within an event.

3.2.1.1 The Event Hierarchy View

An event can be thought of as having parameters and streams. The parameters are things like name, date, duration, etc. The streams also have parameters. In addition to parameters, streams also have a capture configuration and a number of versions. Organizing such a topology of data in an intuitive fashion is a difficult task. A hierarchical view of the data was chosen as the organizing metaphor for event specification. At the top level is the event. “Children” of the event include the event parameters and all of the streams. Stream parameters are under streams as are stream versions. Although not yet implemented, an entry for generated search tools could also be children of a stream.

3.2.1.1.1 The CHierListBox Hierarchical list box

To realize a hierarchical view of an event, a new Windows control was designed and implemented. This control, the CHierListBox provides a hierarchical view of generic data. Entries that have children can be collapsed or expanded by the user by double-clicking. This allows users to see detail when they need to, but not be forced to deal with all of the clutter in general. Each entry in the listbox can be associated with an icon to provide graphical flair. The bitmap can be changed dynamically to reflect changes in state.

3.2.1.1.2 Icons in the Event hierarchy

Associated with each entry in the event hierarchy is an icon. The event entry has a group of three people as its icon. The first person is initially red and the other two are grayed out. When an event is captured, the second person becomes yellow, indicating that the event has been successfully recorded. After an event has been captured, the event needs to undergo a preparation process which shall be described in more detail later. After preparation, the third person becomes green.
At this point all of the people are lit up and the event is ready for playback or migration to a STREAMS server.

Stream entries also have an icon to convey state information. An uncaptured appears as an empty net. After a stream is captured and the event is prepared, the net is filled conveying that the stream is ready for use. Stream versions are represented with a group of differently-sized gray squares. When the versions are generated, the squares are colored in, indicating availability.

3.2.1.2 Adding streams to an event

Adding streams to an event is accomplished through the stream dialog which is accessible via the “Options/Add Stream” menu option. Selecting this brings up a dialog box like the one in Figure 9. The stream dialog lets you enter a stream’s name (a unique ID within an event), start time and duration. It also allows the user to select the type of stream from the stream types available in the system. The dialog will default to a video stream if video capture is available in the system.

![Figure 9 - Adding a stream to an event](image)

Based upon the stream types and the capture device selected to use, there may or may not be a user selectable rate to set. If there are rates to set, the rates combo-box will present the user with available options, otherwise it will be grayed out. All streams will initially be set to a default format. The user can select a different format by pressing the format button. The format button will bring up a device-dependent format selection dialog from which the user makes a choice.

Figure 11 shows a format dialog for a video capture card and Figure 10 shows one for an audio
sound source. Both of them are shown to emphasize the point that these are device-dependent dialogs that are not generated by the capture applications, but rather by the plug-in drivers.

After all parameters are selected, the user can accept the stream specification and it will be added to the event. An entry in the event hierarchy will be created and the user will be returned to the main application window. Stream specifications can be modified after they are added by double clicking on the stream parameters entry of a stream.

3.2.2 The Capture Device Gallery

In addition to the event window another significant display in the main workspace is the Device Gallery. The Device Gallery provides a pictorial view of the available capture devices within the system. Along with an icon for each capture device, the gallery also provides relevant information about the device useful for making decisions in the event specification process. For example: for a video capture device, the gallery lists the maximum number of input sources, the number of sources being used, and the maximum frame rate on the input channel along with the allocated

![Figure 11 - Video Format Dialog](image1)

![Figure 10 - Audio Format Selection Dialog](image2)
frame rate. The gallery might also display the currently selected capture format (see section 3.1.3.2.2) that the card will be capturing with. The Gallery can currently maintain entries for source capture devices only at this point - sink capture devices such as disks have not yet been implemented.

The primary intent of the Device Gallery is to provide feedback to the user about the state of the capture system and the event specification process. The Device Gallery is useful in determining whether or not the capture system is appropriately configured by displaying all of the capture devices and drivers that have been successfully recognized. It also provides useful feedback about how capture resources such as bandwidth and input sources are being used. The Gallery also serves as a springboard into device configuration dialogs which may be provided by STREAMSVideo, STREAMSAudio, or STREAMSGeneric drivers. The configuration dialogs can be called up by double clicking on the device’s icon in the gallery. Figure 12 shows a sample configuration dialog for an electronic whiteboard. In addition to the gallery interface, configuration dialogs can also be called up via the Windows Control Panel driver’s applet.

3.2.2.1 Device Gallery implementation

Another Windows control was designed and developed to support this functionality - CBitmapList. Like the CHierListbox, CBitmapList can be used like a standard Windows control and is fully compatible with all development tools. CBitmapList provides for a list of bitmaps, each with a name and description.

3.2.3 Preview windows

The other major components of the main workspace are the device preview windows. These windows are used to get an idea of what the captured streams will look like on a capture device. The main point of the preview window is to allow the user to see what the streams might look like before starting capture so as to enable interactive adjustment of parameters. The STREAMS
capture system currently supports preview windows for video and audio. Video preview windows display the captured video, in the format that the card is currently set to (as described in the device gallery). Audio preview windows allow the user to adjust microphone and line levels.

3.2.3.1 Video preview - Overlay vs. unassisted preview

The capture system will utilize a video capture's overlay capability if it is present. Overlay is a high-end feature on some video cards that allows you to preview video without loading the CPU or graphics display hardware. Overlay generally results in much higher preview quality than is achievable via unassisted preview - typically 30fps smooth video. Unassisted preview essentially involves capturing frames from the card, passing them through the expansion bus, and displaying them using the graphics display hardware. Unassisted preview performance depends upon CPU power. On a 66MHZ 486 DX2 the maximum unassisted preview rate attainable was approximately 7 frames per second. This however loads down the computer a great deal. The capture system normally performs unassisted preview at 1 frame per second so as not to interfere with interactive response of the application. However, if the preview window is given explicit focus, it will preview at the maximum rate possible.

3.2.4 Capturing events

After an event has been specified in the event window, and preview windows have been checked and adjusted, the operator of the capture system can proceed to capturing the event. To capture an event, the operator selects the "Capture\Start Capture..." menu option. The capture system will attempt to initialize any capture devices that need initialization by invoking a device-dependent, driver-specified, initialization routine. Some capture devices will put up an initialization progress dialog such as in Figure 13.

A pre-capture summary dialog is then presented to the user for confirmation. If acceptable, the user can start event capture. Once event capture begins, the cursor changes to an hourglass, and
the system assumes total non-yielding control of the Window operating system until a mouse click
or keypress is registered. At that point, the capture is completed, all pending writes are executed,
and the event icon in the event hierarchy indicates that the event has been successfully recorded.
During capture, status about the progress of capture are reported in the applications status bar -
information such as number of frames captured, effective frame rate, audio bytes sampled, etc. At
this time no provision is made to provide for a predefined duration or starting time for an event,
however this feature could easily be incorporated.

3.2.5 Event preparation

After an event has been successfully recorded, it needs to be prepared for playback. The
preparation process takes care of some post-processing that needs to be done before the event can
be played back or migrated to a server. To prepare an event, select the “CapturelPrepare
Event...” option. The system will display some information about the preparation process (such as
an estimate for how long the operation may take) and ask for user confirmation. If OK, the system
will proceed and after some time, the event will be available.

3.2.6 Saving events

Events can be saved to disk and restored at a later time. Events are written to the event RIFF
specification file format described in section 3.1.2.4.2.2. By default, events are all stored in their
own subdirectory off of the master event directory. The master event directory is specified with the
EventFilesPath property in the capture system’s configuration files. Streams associated with
events are also, by default, stored in the same subdirectory as the event files themselves.
To save an event to disk or to restore a previously saved event from disk, select “FilelSave” or
“FilelOpen”, respectively. A dialog such as that in Figure 14 appears. The event browser dialog
is analogous to the standard windows File dialogs but provide a more useful display for events.
The browser displays event names, date, and a description to aid in the browsing process.

3.2.7 Configuration files - Setting up the capture system

The capture system is set up through the use of a number of configuration files. These files are
standard Windows INI files and basically consist of a number of “property=value” entries,
arranged by categories.
3.2.7.1 CAPTURE.INI

CAPTURE.INI is the capture system’s own configuration file. A sample CAPTURE.INI file is given in Figure 15. The file is organized into a couple of sections which group configuration options by category. The most significant section is the [CaptureDevices] section. In this section, identifiers for the STREAMS drivers are located. Each entry is of the form "DevicelD=Description". The DeviceID is used to identify the driver with the Windows driver API’s. The description is for readability only and does not affect the operation of the capture system. In the Figure 15 example, there are two devices with STREAMS drivers present in the capture system, the ASLMegamotion and the Softboard. These names are used to index into the other significant configuration file - SYSTEM.INI.

Other sections in CAPTURE.INI include the defaults section which provides default values for some parameters to the capture system. In addition to these two sections, devices may add their own sections, as the Softboard has done in Figure 15, to store device-specific configuration information.
3.2.7.2 SYSTEM.INI

SYSTEM.INI is a standard Windows configuration file that controls much about the operation of
Windows itself. It is usually an extremely large file, and as such, does not warrant complete
inclusion here. However, there is only one section used by the capture application in SYSTEM.INI
- the [Drivers] section. This section provides the actual mappings from the device names specified
in CAPTURE.INI to the DLLs that house the actual code to deal with the devices. An example of
the relevant portion of SYSTEM.INI is given in Figure 16.

```
[CaptureDevices]
ASLMegamotion=Alpha Systems Lab Megamotion
Softboard=Microfield Graphics Softboard

[Defaults]
EventFilesPath=C:\MSVC\SOURCE\STREAMS\CAPTURE\EVENTS
DefaultVideoFPS=5
DefaultAudioSPS=11000
DefaultAudioSampleSize=8

[Softboard]
ComPort=COM2
DownloadDSPOnStartup=FALSE

Figure 15 - Sample CAPTURE.INI
```

```
...[Drivers]...

ASLMegamotion=C:\MSVC\SOURCE\STREAMS\ASLMEGA\ASLMEGA.DLL
Softboard=C:\MSVC\SOURCE\STREAMS\SOFTBD\SOFTBD.DLL

Figure 16 - Sample SYSTEM.INI drivers section
```
3.3 Software architecture

This section will consider some of the software architecture and design issues that were encountered during the construction of the capture system. Algorithms used will be discussed as well.

Overall, over 5000 lines of code were written to achieve the current realization. Most of this code (about 75%) is in the main capture application. The balance consisted of code for the device drivers and assorted tools that were written. All of the main application code is written in C++. The device drivers are written in straight C due to difficulties in writing C++ DLL's. C++ name-mangling interferes with the dynamic linking process and is not straightforward to use.

The application is heavily based upon the Microsoft Foundation Classes version 2.5 (MFC). MFC is an applications-framework designed to make developing Windows applications easier. At times, the lower level APIs of the Windows Software Development Kit were used - especially for some of the more multimedia oriented tasks.

A 66MHZ 486 DX2 was used as the development machine. It was equipped with an Alpha Systems Lab Megamotion video capture card, Sound Blaster 16 audio board, and serial ports for devices such as electronic whiteboards. The Microfield Graphics Softboard was used extensively throughout development both as a capture device, and a brainstorming tool.

3.3.1 The use of C++ - STREAMS and object-oriented organizations

C++ was used as the primary development language primarily because of the object-oriented approaches that it supports. The entire STREAMS system, and especially the capture system, lends itself to using object-oriented methodologies. The primary reason for this is that much of the intelligence involved with handling streams needs to be kept with the stream data so as not to expose any specific stream-dependent details. This also applies to interacting with capture devices in a generic and extensible fashion. Such requirements are classic examples of encapsulation opportunities and the need to keep data with the functions that operate on that data.
3.3.2 Capture devices

For each capture device in a system there is an associated capture object that is used by the capture system to interact with the device. There is a base class CaptureDevice which serves as a parent for an entire hierarchy of capture objects. The class hierarchy is presented in Figure 17.

![Figure 17 - The CaptureDevice C++ Class Hierarchy](image)

The capture objects provide a consistent interface to capture devices and isolate the code from device dependent details.

3.3.3 Capture Device Manager

Handles to capture objects are obtained via a capture device manager. The CaptureDeviceManager class coordinates the creation of all of the capture objects within the capture system - including those with and without STREAMS drivers. It calls different constructors for capture objects based upon this. The capture device manager acts as an intermediary to the capture objects and provides such functions as ensuring that resources are not overallocated.

3.3.4 Event documents and views

The application is structured around events. MFC applications are typically organized using a document-view metaphor - i.e. an application manipulates documents that can be accessed through views. In order to fit events into this approach, two classes were created - CEventDoc, and CEventView. A CEventDoc handles storage of Event objects. A CEventView provides the view of the event visible in the event window.
3.3.5 MrCapture - the man behind the scenes

After events have been specified, and submitted for capture, the event objects are passed off to the MrCapture object. MrCapture parses the event, ensuring that resources have been properly allocated, and decides on the best way to capture the event. There are a number of ways to capture an event, depending upon the streams that are being captured.

3.3.5.1 Capturing events with video

The initial approach to capturing video was to use the low level videoMessage API provided by Video for Windows. These APIs are at the level of grabbing individual frames only and it is left up to the programmer to manage writing the captured frames to disk. The videoMessage API does provide a streaming mode to allow frames to be automatically captured at a specified rate, simplifying timing efforts. This approach worked, however was prone to great difficulty stemming from the low-level of the calls, and only achieved maximum capture rates of 13-15 fps on a 30fps card. The performance degradation was due to poor buffering and disk I/O management—something that could have been enhanced with further effort.

Microsoft released a higher level API for capturing video at an opportune time in the writing of the application. The new API, called the AVICap interface, abstracted away most of the details of video capture and even took care of disk I/O to an AVI file automatically. In addition, it guaranteed portability to the Win32 API for easy use on future operating systems such as Windows NT or Windows 95. The AVICap was vastly superior to the videoMessage API and was adopted.

3.3.5.1.1 Using AVICap

AVICap allows you to specify parameters for event capture, such as frame rate, whether or not capture audio, and what file to capture to. Then an AVICap object can be told to start capturing with those parameters. For STREAMS however, capture is not quite that simple. The capture application also needs to manage capture of data from Generic capture devices simultaneously with the video and audio capture. In addition, video capture is not straightforward when doing interleaving of video sources. In that case, a choice must be made after each frame is captured on whether or not to switch video input sources.
The AVICap provides hooks into the capture process that the capture application uses to take care of its other capture devices and video source switching. Specifically, the capture system uses a VideoStream callback which is called for each frame of video that is captured.

The VideoStream callback used will first determine whether or not the input video source needs to be changed to support interleaving of video. If it is determined that a switch is necessary, the video input is switched through a negotiated switching mechanism. There are two styles of switching for video inputs - a videoMessage API and a CaptureDevice’s SwitchToSource method. The former is a more efficient way of switching sources that some video capture cards provide. After the video source processing is done, the callback proceeds to check if any of the Generic capture devices in the system need to be handled. If so, the appropriate real-time capture methods are called on their capture objects.

3.3.5.1.1 Video buffers and AVICap

A workaround that was necessary to ensure the proper performance of the capture system with AVICap interface was limiting the number of video buffers to 1. Normally, this number is around 5-6 to allow for smooth capture of video. However, in order for the VideoStream callbacks to be called immediately after each frame is captured, there must only be a single video buffer. If there are multiple buffers, several frames can be captured before the VideoStream callbacks are invoked. Even though the correct number of callbacks are invoked, they must be called immediately after each frame is captured to allow the source switching to take place. As a result of this buffer limitation, about 10% of video frames get dropped during 30fps capture. This however is a small penalty compared to the benefits of using AVICap.

3.3.5.1.2 Capture file issues

A single AVI file is used for capture of audio and video stream data. A single file was chosen to capture multiple streams because it simplifies capture complexity, and eliminates many of the disk seeks that would be necessary if several files were used. As a result of this design decision, a separate post-processing step is required to create the individual stream files (discussed in Section 3.3.5.1.3). Before a capture file is used for capture, the capture system ensures that it is defragmented - fragmentation is a phenomenon present on many file systems, including the MS-DOS FAT - the file system in use here. Fragmentation causes a severe performance penalty during real-time capture.
Multi-source Video Interleaving Algorithm

In order to maximize the number of video streams that can be captured simultaneously with a single video capture card, it is proposed that a multi-source video interleaving algorithm be used. By using such an algorithm, you can capture as many video streams as a capture card has input sources. The only limitation is that the combined frame-rates of the video streams must be less than or equal to the maximum frame rate of the video card. This interleaving is essentially a form of sharing a scarce resource, namely a video capture card, among several streams. It is analogous to time-slicing in the sense that each stream gets control of the capture channel for a short amount of time and then relinquishes control. The requirements of video interleaving are slightly different from time-slicing though.

Consider a situation where we would like to capture 3 video streams from a video card that has a maximum of 30 fps capture bandwidth. Video stream A is a fairly high quality stream and is 20 fps. Video stream B is less demanding and is 8 fps. Video stream C is a low quality stream at 2 fps. The question is how do we best schedule the use of the video capture card to capture these three streams?

One approach is to do multiplexing at the stream level. This means that we set up the card at a capture rate of 30fps and capture 20 frames of video stream A, 8 frames of video stream B, and 2 frames of video stream C. Assume that the time to switch between input sources is negligible. It can be seen that this approach will in fact produce streams that contain the correct number of frames as determined by their frame rates and length. However, the captured video will be unsatisfactory because they actual time delays between frames will vary greatly. Consider video stream B, the 8 fps stream. The captured stream will contain 8 frames each captured 33 ms apart in absolute time. The 9th frame will be captured 726 ms after the 8th frame since 22 intervening frames need to be captured for the other streams. A similar problem arises for the other streams.

Clearly, what is needed is a technique that will multiplex the capture channel in as fine a resolution as possible. We need to multiplex at the frame level, not the stream level. An interleaving algorithm is proposed to accomplish this.

Requirements for interleaving algorithm
The algorithm must have the following properties:

**DETERMINISTIC**

This algorithm must be deterministic. In other words, given a set of streams to interleave and their frame rates, the algorithm must multiplex the sources the same way each time. The reasons for this are not immediately obvious. It may seem that it is only necessary to interleave the frames as they are being captured. This would be true if you could physically save the different frames to distinct files during capture. However, in order to optimize performance it is better to capture to a single file and then construct the individual streams later by extracting the relevant frames. This is better for performance because it reduces the amount of file I/O necessary to capture. In order to extract the frames later, we must be able to determine the sequence that the frames were interleaved. We can obtain this information in exactly the same way that it was obtained for use during capture if the algorithm is deterministic.

**LOW SWITCHING OVERHEAD**

Another stringent requirement for the algorithm is that it must not introduce significant delay. Because switching sources must be done in real-time during capture, the technique used to select the source to switch to must not interfere with actual capture. This delay can be thought of with the actual hardware delay involved in switching sources. These two combined must be negligible compared to the capture rate.

3.3.5.1.2.2 Details of The Algorithm

This algorithm is designed to be a low-cost deterministic solution to the above requirements.

Consider a set of streams, \( s_1, s_2, \ldots, s_n \). Each stream has corresponding sampling rate of \( r_1, r_2, \ldots, r_n \).

For each stream \( s_i \) and corresponding rate \( r_i \), form the set of timing pairs \( T_i \). The set of timing pairs \( T_i \) is obtained in the following way:

\[
T_i = \{ \frac{1}{r_i}, i, \frac{2}{r_i}, i, \ldots, \frac{r_i}{r_i}, i \}
\]

Each element in the set is a timing pair - consisting of a sample time and a stream number. Now take the union of all of the sets \( T_i \) and sort the elements in ascending order. The sort is done first on sample time and then on stream number. The resulting sequence of stream numbers in the
sorted set of pairs now gives an ordering for how to multiplex the sources. The ordering is only valid when the total sampling rate for the capture channel is set to the sum of \( r_1 \ldots r_n \).

Example

Let’s consider an example similar to the one that was mentioned before. We have 3 streams, A, B, and C, as before. The frame rates are 8, 4, 3 fps, respectively. Form the set of timing pairs \( T \) for each stream.

Stream A - 8 fps

\[
T_a = \{ <.125, 1>, <.25, 1>, <.375, 1>, <.5, 1>, <.625, 1>, <.75, 1>, <.875, 1>, <1, 1> \}
\]

Stream B - 4 fps

\[
T_b = \{ <.25, 2>, <.5, 2>, <.75, 2>, <1, 1> \}
\]

Stream C - 3 fps

\[
T_c = \{ <.33, 3>, <.66, 3>, <1, 3> \}
\]

Taking the union of \( T_a \), \( T_b \), and \( T_c \) and sorting the result we obtain:

\[
T = \{ <.125, A>, <.25, A>, <.25, B>, <.33, C>, <.375, A>, <.5, A>, <.5, B>, <.625, A>, <.66, C>, <.75, A>, <.75, B>, <.875, A>, <1, A>, <1, B>, <1, C> \}
\]

Now just extracting the sources we are left with:

Chapter 3

The resulting sequence can be used to select which source to switch to during the real-time capture process. The sequence can be repeated over and over again. This sequence interleaves the frames more evenly than the stream level approach and produces much better quality video streams.

It can be seen that this algorithm is obviously deterministic. In addition, since the information about the streams will be available prior to the start of real-time capture it is possible to calculate the switching sequence beforehand as well. This means that during real-time capture, the only action that must be taken to determine what source must be switched to next is a lookup. In addition, since the period of the sequence generated is also known beforehand, and it is also possible to know exactly where in the sequence the switching process is, a simple array lookup can be used. This is a nominal cost and can be considered insignificant.

3.3.5.1.2.3 Implementing the algorithm

In order to use this algorithm within the capture application a C++ class was created. This class, SwitchTable, encompasses the functionality discussed here. The SwitchTable accepts any number of input streams and their integral frame rates. The SwitchTable must then be initialized. Initialization enacts the algorithm on the input stream and prepares the SwitchTable for real-time capture. After initialization, the SwitchTable is indexable from 0 to the combined frame rate less 1. Each value in the SwitchTable indicates the source that should be captured from at that frame in the sequence. For example, if SwitchTable[4] = 3, it means that the fifth frame of the sequence should be captured from source 3.

3.3.5.1.2.4 Caveats and Future directions

Possibly interesting avenues of further investigation might include considering a non-negligible hardware switching time. In the case of the hardware used throughout this work, the switching time was for all intensive purposes, instantaneous. However, other less sophisticated video capture cards might have minor, but not negligible delays. Such delays would interfere with the proper operation of the interleaving algorithm. The algorithm could potentially be modified to incorporate such delays.

There is also the possibility that it might not be possible to specify exactly the rate at which a channel captures. For example, some capture boards only allow capture at 30 fps. If our stream
requirements add up to only 17 fps then the algorithm will not produce correct results. One possible solution to this problem is to insert a “dummy” stream that will take up the extra 13 fps.

3.3.5.1.3 Post-processing events with video

Events are captured during real-time capture into a single AVI file. All video and audio streams are stored in this file. If there are multiple audio or video streams in an event, their stream data will be interleaved together (interleaved within stream type, not between types) and must be processed to create their individual stream files. The MrCapture post processing methods handle this. These methods will split interleaved video into the right number of stream files, and split stereo audio into two monaural audio stream files, if there were two streams specified in the event. In any event, the original capture AVI file is discarded after post-processing.

3.3.5.2 Capturing events without video

In order to capture events without video, a timing loop is used instead of callbacks. This loop essentially just keeps track of time and calls the appropriate capture methods on the capture objects when their update periods are expired.
4. Results and Conclusions

4.1 What was implemented

An integrated capture environment for specifying and capturing events was designed and implemented. These events could be comprised of any number of streams of different types. The types available to capture with were dependent on hardware available and were determined at runtime via a plug-in architecture for capture devices. This plug-in architecture ensured that devices could be incorporated into the system by supplying an external STREAMS driver only. Support for the Microfield Graphics Softboard was provided in this fashion. In addition, the capture environment also provided for event management after event capture. For example, different stream versions and representations could be created. Finally, provisions were made in the captured streams to obtain information about the streams useful for generating data for search tools, although this information was only available on the simplest of levels.

4.2 Problems encountered - Future opportunities

A number of problems were encountered during the design and construction of the capture system. Some of these problems can potentially be addressed with better operating systems, while solutions for others are not so obvious.

4.2.1 Guaranteeing performance - a problem of trust

The most significant problem encountered had to do with guaranteeing a certain level of performance to streams during capture. This problem was intimately related to the Generic interface design for plug-in architecture - the feature that supported extensibility and device-independence in the capture system. Because capture from these Generic devices is handled within their STREAMS drivers, and not within the capture application itself, the STREAMS drivers have to be trusted to perform correctly. This not only had ramifications on capturing from Generic devices, but also on the entire capture system as a whole. Because of the nature of the Microsoft Windows operating system, there was no possibility for preemption during capture. As a cooperative-multitasking operating system, Windows has absolutely no way of interrupting an errant STREAMS driver. It is completely up to the STREAMS drivers to return control to the capture application so that it may continue to capture from other devices.
This problem was noticed during the first integration of a Generic device into the capture system. When the Softboard driver was used for electronic whiteboard capture, there was a tendency for video frames to be dropped as people were writing on the whiteboard. The problem was eventually traced to a function within the Softboard interface logic. When there was data available from the Softboard, it would read as much data as possible until there was no more data. This potentially introduced large delays during which no other streams could be handled. The problem was rectified by ensuring that the Softboard driver would return within a certain specified time interval. Even with this solution, the dependence of the capture system on trusting the plug-in drivers to return control is clearly evident. If the drivers are poorly written, the entire operation of the capture system is affected.

4.2.1 Possible solutions

The best approach to solving this trust problem is simply to ensure that the capture application can always grab control back from capture applications. This can best be accomplished on a true preemptive multitasking operating system by using separate processes to capture from each device. An even better approach would be to utilize separate threads of control within the capture process to capture from each device. Unfortunately the only operating systems that support these kinds of functionality on PCs are Windows NT and OS/2. These operating systems do not have good hardware support though, and are thus unsuitable for a capture system. It should be also be noted that separating the capture into threads or processes also introduces potential problems of time synchronization. Such issues could be addressed by using RPC or LRPC mechanisms. However, their potential effect on capture performance, as a result of the cross-process communication, is unknown.

4.2.2 Loss of information

With some capture devices, the capture process fails to capture all of the important properties of the stream. For example, consider an electronic whiteboard stream. Such a stream accurately captures all of the writing on the whiteboard. However, information about where participants might be pointing is not kept. This is especially important because users of an electronic whiteboard may spend a good deal of time pointing and gesturing at written notes to emphasize certain points.
4.2.2.1 Possible solutions

One way to capture this information is to simultaneously capture video of whiteboard activity along with the electronic whiteboard stream. This approach is simple, however not very desirable. It would be irritating to have to constantly shift focus between two different streams to get a good overall view of whiteboard activity.

Another possible technique is to try and extract pointing information from the video of a whiteboard stream and somehow incorporate that into the electronic whiteboard stream. It is possible to obtain information about where participants may be pointing by processing individual frames in the video. Trying to incorporate this information into another stream, or even generating another stream from the electronic whiteboard stream and the pointing information stream, introduces a host of questions and problems that could not be addressed within the scope of this thesis. However, this could be a potentially interesting avenue of further research, as it considers how to link different streams of data together, based on stream content and semantics, rather than timing.

4.3 Conclusions

It has been said that one of the most important qualities in successful organizations is good interpersonal communications [Kraut95]. Interpersonal communications refer to interactions between people in both formal and informal contexts - such as group meetings, design reviews, spontaneous discussions, and lectures. With this in mind, it is easy to see how a system like STREAMS really has the potential to improve the quality of formal interpersonal communications. STREAMS can improve the quality of such interpersonal communications by assuring the participants that their ideas and interactions are being recorded in a way that preserves the essence of the event. By allowing people to replay such recorded events at their convenience, and enabling powerful ways of accessing the information within the events, STREAMS introduces a new paradigm into the realm of group interactions.

This research has addressed some of the issues involved with making a system like STREAMS a reality. While the ideas presented are specific to a capture system within the STREAMS environment, the concepts can be applied in a number of similar capture situations. This thesis has demonstrated the importance of digital representations for the storage of media and provided a
compelling realization of a system that uses such representations to richly record events. Multimedia, the base upon which STREAMS is built, is still a new technology. However, as the technology matures and becomes accessible to more people, applications such as STREAMS will change the way people and organizations think about group interactions. This research has shown that in the case of interpersonal communications events, some of that change is already underway.
Appendix 1 - Specifications

A1 - Event RIFF Specification

The event file generated by the capture system is a RIFF file of form type 'EVNT'. This file obeys all of the conventions for RIFF files [MMREF90] and has the following specification:

```
<EVNT-form>  →  RIFF( 'EVNT'
         <eventInfo-list>
         <Streams-list>
          [<Partcpt-list>]}

<eventInfo-list>  →  LIST( 'einf'
         {{<Desc-ck>}
          [<Comment-ck>]
          [<Date-ck>]
          [<Duration-ck>]
          [<Location-ck>]
         }

<Streams-list>  →  LIST( 'stms'
         {<numStrm-ck>
          [<Stream>]...}}

<Partcpt-list>  →  LIST( 'parl'
         {{<numPart-ck>
          [<PartcptName-ck>]...}}

<Stream>  →  LIST( 'strm'
         {<streamInfo-ck>
          <versionInfo-ck>})

<Desc-ck>  →  'desc' (<ZSTR>)

<Comment-ck>  →  'comm' (<ZSTR>)

<Date-ck>  →  'date' (<LONG>)

<Duration-ck>  →  'dura' (<LONG>)

<Location-ck>  →  'locn' (<ZSTR>)

// STREAM CHUNKS

<numStrm-ck>  →  'nstm' (<USHORT>)
```
<streamInfo-ck> \rightarrow 'sinf' (<StreamInfoStruct>)

<versionInfo-ck> \rightarrow 'vinf' (<VersionInfoStruct>)

// PARTICIPANT CHUNKS

<numPart-ck> \rightarrow 'nump' (<BYTE>)

<PartcptName-ck> \rightarrow 'pnam' (<ZSTR>)

// STRUCTURES

<StreamInfoStruct> \rightarrow struct {
    unsigned long structSize;
    unsigned short streamType;
    unsigned long startTime;
    unsigned long duration;
    BOOL isCaptured;
    unsigned short numVersions;
    unsigned long ofsStreamName;
    unsigned long ofsCaptureDevice;
    unsigned long ofsStreamTypeDesc;
    char stringData[1];
}

<VersionInfoStruct> \rightarrow struct {
    unsigned long structSize;
    unsigned short numVersions;
    char versionData[1];
}

A1.1 Description of structures used

A StreamInfoStruct is a variable length structure containing information about the logical characteristics of a stream. The structSize member contains the length of the structure. There are several variable length null-terminated strings packed into the stringData field. The ofsXXX members give the index of the first character of the XXX string inside the packed stringData field. For example, to obtain the a description of the stream type, read the null-terminated string at stringData+ofsStreamTypeDesc.
The VersionInfoStruct contains information about a particular scalar version of a logical stream. For example, if a video stream has 160x120, 320x240, and 640x480 versions, then numVersions will be 3, and each one will have a separate StreamVersionStruct contained in the versionData variable length field. Version information is packed into the versionData field of the VersionInfoStruct in the following manner:

\[<\text{len}>\text{StreamVersionStruct}<\text{len}>\text{StreamVersionStruct}>...\]

where \(<\text{len}>\) indicates how many bytes of StreamVersionStruct immediately follow it.

A1.2 - The STREAMSVideo Driver Specification

STREAMSVideo drivers respond to all of the standard installable-driver messages, and a few additional messages:

- SDRV_VID_QUERY_INFO_SIZE
- SDRV_VID_GET_INFO
- SDRV_VID_QUERY_SOURCE_TABLE_SIZE
- SDRV_VID_GET_SOURCE_TABLE

The SDRV_VID_QUERY_yyy_SIZE messages return the amount of memory needed to hold the message for the corresponding SDRV_VID_GET_xxx message.

The SDRV_VID_GET_INFO returns a STREAMSVideoInfo structure which gives general information about the capture card that is unavailable through other means. It is defined as follows.

```c
typedef struct _STREAMSVideoInfo {
    unsigned long structSize;
    unsigned short maxFrameRate;
    unsigned short msVideoIdx;
    unsigned short numSources;
    unsigned long ofsName;
    unsigned long ofsVersion;
    unsigned long ofsFileExtension;
    unsigned long ofsErrorMessage;
    char stringData[1];
} STREAMSVideoInfo;
```
The SDRV_VID_GET_SOURCE_TABLE message returns information on how to switch the current video source via a videoMessage() API. The videoMessage() function accepts three parameters, a message ID which is used to instruct the video card to switch sources, and two arguments specifying the source. These parameters are returned in a SwitchableSourceInfo structure along with a string used to identify the source to the operator in the capture interface. The format for this structure is:

```c
typedef struct _SwitchableSourceInfo {
    UINT messageId;
    DWORD dwParam1;
    DWORD dwParam2;
    char sourceName[20];
} SwitchableSourceInfo;
```

A1.3 - The STREAMSGeneric Capture Driver Specification

STREAMSGeneric drivers respond to all of the standard installable driver messages and a number of additional ones. These additional messages are used by the capture system to control generic capture devices and to obtain information about them.

- **SDRV_GENERIC_QUERY_INFO_SIZE**
- **SDRV_GENERIC_GET_INFO**
  Obtains the size of the information message, and gets the information about the capture device respectively. The information is returned in a STREAMSGenericInfo structure.

- **SDRV_GENERIC_INITIALIZE**
- **SDRV_GENERIC_SHUTDOWN**
  Used to initialize and shutdown the capture device respectively. Initialization encompasses one time operations that typically need to be done after power-on.

- **SDRV_GENERIC_SHOW_SETUP_DLG**
  Displays a driver-dependent dialog used to configure and setup the capture device.

- **SDRV_GENERIC_SHOW_FORMAT_DLG**
  Displays a driver-dependent dialog used to select a capture format for the capture device.

- **SDRV_GENERIC_SHOW_SOURCE_DLG**
  Displays a driver-dependent dialog used to select the input source used to capture from.

- **SDRV_GENERIC_SHOW_SSFORMAT_DLG**
Displays a driver-dependent dialog used to select a capture format for Source-specifiable capture devices.

SDRV_GENERIC_PREPARE_TO_CAPTURE

Sent to the driver to indicate that capture is about to begin. Used to perform capture-time initialization.

SDRV_GENERIC_START_CAPTURE
SDRV_GENERIC_STOP_CAPTURE
SDRV_GENERIC_PAUSE_CAPTURE
SDRV_GENERIC_RESTART_CAPTURE

Starting capture initiates recording on the capture device. The driver will accept SDRV_GENERIC_CAPTURE_UPDATE messages until a stop or pause message is received.

SDRV_GENERIC_SWITCH_SOURCE

Switches the capture input source, either before or during capture. Programmatically switches the source - not via a dialog box.

SDRV_GENERIC_SWITCH_FORMAT

Switches the capture input format, either before or during capture. Programmatically switches the format - not via a dialog box.

SDRV_GENERIC_CAPTURE_UPDATE

Repeatedly sent to a STREAMSGeneric driver during real-time event capture to give the capture device some time to capture.

The STREAMSGenericInfo structure returns information to the STREAMS capture application about the capabilities of a capture device. It also contains information useful for construction of user interfaces such as bitmaps. Its format is as follows:

```c
typedef struct _STREAMSGenericInfo {
    unsigned long structSize;
    DWORD streamType;
    unsigned short updateFreq;
    unsigned short numSources;
    DWORD capabilities;
    HBITMAP bitmap;
    unsigned long ofsName;
    unsigned long ofsVersion;
    unsigned long ofsStreamDesc;
    unsigned long ofsErrorMessage;
    char stringData[1];
} STREAMSGenericInfo;
```
References

Aarons94  

Berlin92  

Blackketter92  

CraigHill93  

Cruz94  

EAIFF85  

Elliot93  

Gabbe94  

Isaacs94  

Ishii90  

Katseff94  
References

Kraut95  

Moore93  

Mills92  

RAD94  

Softboard94  

Tonomura93  

Ueda93  

Whittaker94  