Capture-Time Indexing Paradigm, Authoring Tool, and Browsing Environment for Digital Broadcast Video[1]

Michael Carreira,[2] John Casebolt,[3] Gerard Desrosiers,[4] and T.D.C. Little

Multimedia Communications Laboratory
Department of Electrical, Computer and Systems Engineering
Boston University, Boston, Massachusetts 02215, USA
(617) 353-9877, (617) 353-6440 fax
tdcl@bu.edu

MCL Technical Report 01-08-1995

Abstract–Historically, Multimedia Video-on-Demand (VOD) systems have considered stream indexing as an authoring activity, decomposing monolithic streams containing no explicit indexing information. This paper suggests a scheme for "up-front," capture-time indexing of digital video streams, whereby the indexing information logically becomes part of the stream. This approach takes advantage of the sequential, temporal nature of capture and the knowledge of the stream recorder to empower further manipulation and playout of the stream. We explore the impact of capture-time indices, and implement a sample format in a Segment Definition File (SDF).

The Video Broadcast Authoring Tool (VBAT) is the focus of our paper. Taking a video stream and an SDF as input, VBAT provides a means for authors to create, delete, modify and annotate segments of that stream. VBAT also integrates existing technology such as World-Wide Web's HTTP links and the Motion Picture Parser application.

Creation of various-format stills for browsing is supported. VBAT provides for post-processing of the SDF to various playout environments; we implement and describe a post-processor for World-Wide Web browsing and playout. Finally, we discuss VBAT's position in an integrated digital video broadcast environment and areas of future work.

[1]In *Proc. Multimedia Computing and Networking*, IS&T/SPIE Symposium on Electronic Imaging Science and Technology, SPIE Vol. 2417, February 1995, pp. 380-388.

[2]Quantum Corporation, Shrewsbury, MA 01545, *carreira@leds.enet.qntm.com*

[3]United States Air Force, Draper Laboratory, Cambridge, MA 02139, *casebolt@bu.edu*

[4]Naval Undersea Warfare Center Division Newport, RI 02841, *desrosiers@v22c.npt.nuwc.navy.mil*

Keywords: Multimedia authoring, indexing, browsing, digital video.

1 Introduction

Recently, we had the chance to review over 100 papers representing both current and classical thinking in the field of Multimedia. In the process of distilling all this information, while trying to gain a 'big picture' view of the current state-of-the-practice in digital video, we became intrigued by a couple of topic areas, namely indexing and browsing.

This paper documents research efforts leading out of that interest, and presents two different, yet inter-related ideas: a capture-time indexing paradigm, and an extensible, portable authoring tool implementing that paradigm.

In the majority of literature on continuous-media stream indexing, the subject is viewed as a post-capture/post- creation activity, based on decomposing monolithic streams containing no explicit indexing information [1, 2, 3]. This 'downstream' indexing has quite a few disadvantages, including computationally-intense processing and, in many cases, hardware-specific display requirements. Couple these drawbacks with the published fact [2, 3] that much of authoring is just flat tedious work – both error-prone and boring – and we wonder if there is a better way to do the business of authoring.

Interesting work has been done by Deardorff, et al., to alleviate some of the tedium involved with authoring by introducing the Motion Picture Parser [4], a tool designed to detect scene/shot transitions by operating mathematically on a histogram of the individual frame sizes. Although reducing the need for laboriously cycling fast-forwards/rewinds to find scene changes, the MPP algorithms themselves can be quite compute-intensive, perhaps more appropriate to a non-interactive/off-line environment. We believe that something can be done to give the author a 'head start' on the event/scene transition identification process. In particular, there exists a way of presenting to the author an idea of what the recorder of the stream had in mind as ideal browsing points, highlight frames, audio marks, or content transition points.

In the end, we struck on a very simple idea, one which we have not seen in the literature to date. Why not add indexing activities to the front-end of the process, either capture-time (for live streams) or creation-time (for pre-produced streams)?

Exploring this concept a bit further, it becomes clear that this scheme has many advantages. Perhaps most importantly, it allows the introduction of a line-of-communication – a bridge – between the recorder of the stream and the author of the stream, perhaps two different people. Additionally, depending on the amount of indexing information captured, very little overhead would be involved in the logical (perhaps even physical) inclusion of indexing information in the stream itself. As a return on this investment, the author would receive a stream pre-indexed – providing the head start we want, and perhaps saving many hours of traversing the stream looking for certain events.

Concentrating for a moment on the capture of live streams, we can argue that the greatest gains could be made in this area (as opposed to prepared, pre-produced streams). Since the capture of live streams is essentially sequential in its temporal nature – i.e., it takes 300 seconds to capture 300 seconds of a live event, no more, no less – we have an opportunity to take advantage of this time to provide indexing information. The point being that since the 300 seconds in the above example must be spent to capture the event, why not also use it to index the event, rather than wasting another person's time downstream to do the same work?

A perfect example would be the capture of a sporting event. Taking a baseball game, say we wanted an index at the end of every inning, after every out, and even after each pitch. Without capture-time indexing, someone must watch the game in its entirety at a later date to mark the stream with the appropriate information, doubling the amount of time necessary to have a stream with this information embedded. Why not mark the stream as the game progresses?

Another important part of the information transfer along our conduit from recorder to author is domain expertise. It is easy to imagine a scenario where the recorder has information or knowledge which would allow the recognition of an important event during the capture of a stream; an author lacking the same information may not understand the significance of the event. Using our baseball game example again, stipulate that a player steals a base, setting the all-time record for bases stolen in a single season. To the announcer and/or recorder of the game, as well as to future viewers of the stream, this represents a major event. To an author preparing this stream for viewing who happens to not follow baseball, the event may just be another stolen base - nothing of any consequence - and the event may not be indexed. This is an example of information loss due to an author unfamiliar with the domain of the stream which is being indexed.

Having defined the indexing paradigm, the paper will present an example of capture-time indexing implementation. Next, we will relate our design and implementation experiences with an authoring tool based on the paradigm. In conclusion, we will summarize our experiences, relate some lessons learned, and discuss future work in the areas covered.

2 Capture-Time Indexing Implementation

Implementation of a capture-time indexing paradigm could take many forms. After presenting a few of these theoretical possibilities, we will examine the Segment Definition File, our chosen format for implementing capture-time indexing.

2.1 Capture-Time Indexing Implementation - In Theory

It is convenient to establish two categories of indexing information inclusion: physical and logical. Physical inclusion dictates that the indexing information is actually embedded in the stream itself. Logical inclusion implies a relationship between the stream and the indexing material based upon a projection provided by an external mechanism; for example, an authoring tool.

An example of physical inclusion would be immersion of the index mark in an MPEG stream, perhaps as a sequence of contiguous I-frames, or, more elegantly, as a logical function layer in the bit stream [5]. Various encoding methods (H.261, J-Movie, etc.) could be adapted to provide a place to 'hide' the indexing information. In short, an entire paper could be devoted to studying the current and future encoding schemes with a mind for extension to include this type of out-of-band information.

For the purposes of this research, we chose to implement a logical inclusion scheme. The reasons for this choice are three: simplicity, portability, and utility. Addressing simplicity first, we did not have to modify any existing encoding method to carry out our research. Next, portability; by choosing a logical inclusion method, we did not need to worry about portability problems between varied coding schemes, which complemented one of our motivations for building the Video Broadcast Authoring Tool, described below. Finally, addressing utility concerns, it is much easier to access and manipulate the indexing information outside of the stream in a concise format, leveraging available operating system tools. In our case, the information is even human readable, residing in an ASCII file.

Drawbacks of the logical inclusion scheme which are immediately obvious deal with configuration management and performance. As we shall see, steps were taken to deal with configuration concerns, and performance, in our experiments, did not seem to be an issue due to the relatively small set of information dealt with (on the order of 100 or so indices for a given stream).

2.2 The Segment Definition File (SDF)

The vehicle we have chosen to implement our capture-time indexing database is the Segment Definition File (SDF). The SDF is simply a formatted ASCII file containing stream information and segment information. The SDF may be automatically generated at capture time; in our research, however, the initial SDF was created by a test program or by hand.

The contents of the file were designed to support the types of user queries found in some of the literature [6, 7], as well as provide information necessary to perform configuration management and extended annotation. We decided to let the SDF provide both our capture-time indexing information and authoring information. Since a capture-time index implementation may provide either minimal information for an indexed event (perhaps only the frame number) or a more robust collection of data, our SDF provides the notion of optional data fields. As portrayed in Fig. 1, the SDF is composed of exactly one header section and one-or-many segment sections. Of particular interest to Capture-Time Indexing is the fact that each indexable event is marked by the creation of new segment entry in the SDF, minimally with a start frame number and nothing else. See the first segment entry in Fig. 1 for an example of a capture-time index segment. The second, more fully-populated segment section will be explained below in the context of the Video Broadcast Authoring Tool.

```
  ———————————-Header—————————
Source Media File: stout.mv
Session Name : team-intro
Still File Format: GIF
File Produced By : Video Broadcast Authoring Tool
Program Version : PreBeta
Date/Time : Mon Aug 15 20:06:41 1994
———————————Segments————————-
Segment Name :
Parent Name :
Children :
Start Frame : 5
Stop Frame :  Rep Frame :
Narrative :
External Refs:
x Position :
y Position :
————————————————————————

Segment Name : Stout
Parent Name : NONE
Children : 3
Jerry
John
Mike
Start Frame : 0
Stop Frame : 0
Rep Frame : 94
Narrative : This is our favorite segment
External Refs: 2
Name: BU MCL
URL : http://spiderman.bu.edu
Name: Quantum
URL : http://www.qntm.com
x Position : 1
y Position : 0
```

Fig. 1 - Example Segment Definition File

3 The Video Broadcast Authoring Tool

The Video Broadcast Authoring Tool (VBAT) is a tool for segmenting and annotating previously-captured digital video streams. Its main purpose is to break a video stream into logical segments and then arrange those segments in a hierarchical manner for subsequent browsing and playout. The following sections describe our implementation of VBAT. The motivations for creating VBAT will be outlined as will some interesting design and implementation details that were encountered developing VBAT.

3.1 Motivations

The Video Broadcast Authoring Tool (VBAT) was developed primarily as a vehicle for demonstrating the usefulness of capture- time indexing by taking advantage of a "capture time" SDF. Secondary motivations include the integration of existing capabilities (such as the Motion Picture Parser and Hyper Text Transfer Protocol) as well as a way in which to automate the creation of HyperText Markup Language (HTML) files.

A "capture time" SDF may be generated as a by-product of the initial video capture. This SDF will define certain frames as marked frames, allowing VBAT to initialize the user's display with a still image representing each marked frame. At this point, we define this representative frame as the frame from which a still image is extracted for purposes of visual identification of a segment during authoring and browsing. Notice that at this time, each marked frame is considered a segment of length one – i.e., a segment with identical start and stop frames (and, for now, an identical representative frame). Each such single-frame segment will be positioned in the user's display as the root node of an n-ary tree. The tool will now allow the author to manipulate each segment by modifying start, stop, and representative frames as well as entering narrative data and hypermedia links to related information. An internal, VCR-like media player is provided to aid in the start, stop, and representative frame selection. The player provides forward and reverse play, single step forward and reverse play, and fast forward and reverse play.

As we alluded to earlier, the sequencing of segments is derived from their position in one or more n-ary trees; leaf segments are traversed in a depth-first, left-to-right manner.

Segments may be "dragged" and "dropped" to different hierarchy levels or locations in the main window grid and attached to parent segments to edit their playout relationships. Although somewhat simplistic, this sequencing relationship more than adequately provides us with a framework in which to carry out our research.

The results of all these modifications are stored in a new SDF. As seen in Fig. 1, the SDF includes data fields for each segment parameter (e.g., Parent Name, Start Frame, Representative Frame, Grid Position, etc.). In addition to the modification of existing segments, new segments may be added or existing segments deleted at any time during an authoring session.

VBAT maintains a separate SDF for each authoring session. A session may be suspended and resumed any number of times with the SDF containing all pertinent session information. Multiple authoring sessions are allowed for each media file. Thus, multiple views of a captured audio/video sequence may be maintained, possibly providing alternate perspectives of the media stream. Segmented media files and still image files may be generated for leaf node segments as well as a HyperText Markup Language (HTML) file to facilitate browsing and playout. VBAT integrates existing capabilities such as the Motion Picture Parser [4], Hyper Text Transfer Protocol (HTTP), and related World-Wide Web browsers to provide a robust authoring environment.

In the event that a "capture time" SDF is not available, a useful feature is provided by VBAT to determine all shot transitions in the media file. This capability is provided by integrating VBAT with the Motion Picture Parser [4] which detects shot transitions. This provides the author a starting point in the editing process in the absence of "capture time" markings. VBAT also uses the MPP algorithm in VBAT's VCR-like player feature. In this setting, it is used to jump to the next shot transition, allowing the user quick navigation of the video stream. This effort also capitalized on the Hyper Text Transfer Protocol (HTTP) to provide external links to supplementary information for a particular segment. VBAT allows the entry of multiple links for each segment and will utilize Mosaic to support resolution of reference link entries. The back-end of VBAT will generate an HTML file which represents the playout tree generated by the author. This final product will allow an end user to browse the video in an organized fashion.

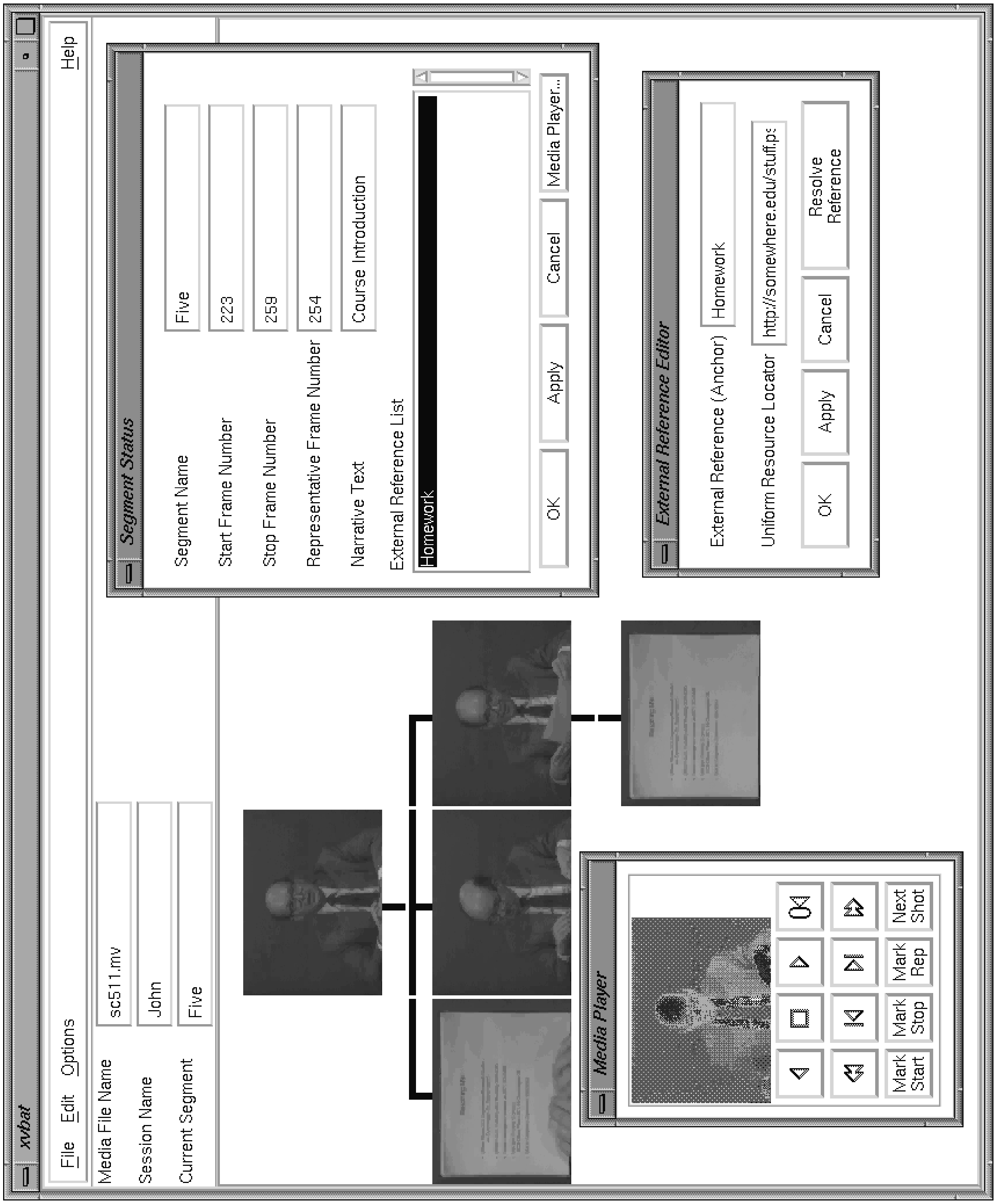
3.2 Design

VBAT fits into the system environment depicted in Fig. 2. VBAT inputs include the monolithic video stream to be arranged and, optionally, a capture-time or authoring-session SDF. VBAT outputs are an updated SDF (capturing the state of the authoring session at save time) and, optionally, physically-segmented video stream files representing each segment defined during the authoring session. A post-processing back-end for VBAT will utilize the physical output of VBAT (SDF and media segments) to prepare and format the logical output (segments, annotation, playout order) for browsing and playout in a particular environment. For demonstration purposes, we chose to implement a post-processor capable of creating an HTML file for subsequent use by World-Wide-Web browsers such as Mosaic. This post-processor also creates a set of stills, one for each representative frame, in a format configured by the author (current supported formats include XPM, JPEG, and RGB). These stills are displayed as imagemaps in the HTML output.

VBAT has been designed in a modular fashion with well defined internal interfaces to support future extensions. VBAT is composed of the following modules: User Interface, Segment Data Manager, Audio/Video Interface, Authoring Services, File services, Command Processing, and HTML-generating post-processor. VBAT was modularized in this manner to accommodate anticipated future enhancements. In our judgment, some system components that are likely to change are the video encoding type (e.g., MPEG vs. SGI format), the required back-end (the browsing/playout system may not require HTML), and target windowing system (e.g., Windows vs. X). Any or all three of these potential environment changes can be accommodated independently by changing the Audio/Video Interface module, the post-processor, or the User Interface module, respectively.

3.3 Implementation

A bit more attention was paid to the process of implementation for VBAT than is usual for research projects. We did this to encourage future enhancements of VBAT as well as to satisfy external project requirements. VBAT was implemented using a tailored Waterfall [9] software development process model which was extended to include a Human Computer Interface (HCI) prototype phase. A Software Project Management Plan was developed per IEEE standard [8], and other documents (including a Software Requirements Specification, Software Design Document, Software Test Plan, User's Manual, System Administrator's Manual, and Software Portability Guide) were produced.


xvbat

File Edit Options Help

Media File Name: sc511.mv

Session Name: John

Current Segment: Five

Segment Status

Segment Name: Five

Start Frame Number: 223

Stop Frame Number: 259

Representative Frame Number: 254

Narrative Text: Course Introduction

External Reference List: Homework

OK Apply Cancel Media Player...

External Reference Editor

External Reference (Anchor): Homework

Uniform Resource Locator: http://somewhere.edu/stuff.ps

OK Apply Cancel Resolve Reference

Media Player

Mark Start, Mark Stop, Mark Rep, Next Shot

Figure 1: VBAT System Environment

The prototyping effort was conducted using the ICS Builder Xcessory (BX) tool and served to clarify requirements associated with the VBAT HCI. The BX tool proved to be very useful for a novice Motif developer to rapidly create static displays. The tool was abandoned upon completion of the prototyping phase primarily due to its consistent hard coding of child widget size resources. Afterwards, it was determined that a BX default parameter could have been altered, thus allowing widgets to manage their own size resources. Although the bulk of the code generated by BX was retained throughout the development effort, certain modifications were performed to enhance portability and modularity. These modifications included utilization of the variable-argument Motif functions to minimize the number of source statements, the removal of redundant BX "helper" code (embedded XPM code, specifically), and remodularization of the generated code into smaller, more manageable files. VBAT was developed in ANSI C, compiled with GNU GCC version [2, 5, 6] and SGI's unbundled C compiler.

VBAT's development took advantage of code reuse by utilizing existing public domain software wherever feasible. Video software included XPM, NetPBM, SGI libraries and SGI demo software. Some of the tools used in development included Builder Xcessory, CVS, Imake, RCS, Schmit Rene's Memdebug and Conor P. Cahill's dbmalloc. In all, VBAT consists of 8800 new non-comment source lines, and 3200 modified source lines. A full-page screen capture of a VBAT session is presented on the following page. The main VBAT window contains information such as the Media File Name, Session Name, and Current Segment, as well as a five-segment tree containing three leaf nodes and one root node. In addition to the main window, windows for the Media Player, Segment Status, and External Reference Editor are displayed. The Media Player window is described above in the text.

Segment Status displays information pertaining to the currently selected segment (in this case, Segment Five). The External Reference Editor allows interactive definition, modification, and resolution of Uniform Resource Locators (URL's) for our current segment.

4 Concluding Remarks

In summary, we have forwarded the idea of capture-time indexing as a tool for enhancing the quality and efficiency of the authoring and subsequent browsing and playout of digital video streams. We have briefly discussed our implementation of an authoring system utilizing capture-time indexing, the Video Broadcast Authoring Tool (VBAT).

We have presented VBAT as a modular, extensible authoring tool which leverages existing code and modern software engineering tools and practices to produce a solid foundation for a digital video authoring environment. Emphasis was placed on VBAT's adaptability, and the concept of independent post-processors was explained. We will conclude with a presentation of lessons learned, and discuss future work.

4.1 Lessons Learned

This project provided the team with an opportunity to learn much about Multimedia in general, and even more about authoring and digital video/image implementation in specific. First, and most importantly to us, we have been able to successfully implement a system capable of taking advantage of capture-time indexed media streams. Second, we feel encouraged that it possible to build a high-quality, useful authoring tool in an Open-Systems environment using very little other than freely-available software libraries and a minimum of hardware-specific code. Once again, our experiences prove that it is always wise to explore the Internet extensively prior to implementing anything - standing on the shoulders of existing work allows creation of useful new capabilities in a robust framework.

4.2 Future Work

The life of VBAT (and capture-time indexing) is far from over. Starting in the first few months of 1995, VBAT will be extended by an entirely new team of researchers. Although their work will primarily focus on enhancing VBAT functionality, porting efforts are also planned to allow VBAT to utilize Parallax video boards on Sun workstations, Indy Video boards/COSMO Compression boards on SGI workstations, and perhaps some as-yet-undetermined format on Pentium workstations running Linux.

On a system level, VBAT figures to play an important part in a new fast-access digital video initiative in the Multimedia Communications Laboratory. Adaptation to this initiative's front-end capture and back-end browsing/playout environments will provide another challenge to VBAT's modular design.

Finally, although not currently scheduled, a large amount of work remains in the area of capture-time indexing implementation methods. This work needs to explore not only the encoding and encapsulation of the indexing information (logical and physical approaches), but also the methods for actually marking the desired frames, including manual (pushbutton) and automatic (pattern- or color-matching) paradigms.

5 Acknowledgements

The authors wish to thank Quantum Corporation, Naval Undersea Warfare Center, Draper Laboratory, and The United States Air Force Institute of Technology for funding while conducting this research. Also, special thanks to Dinesh Venkatesh for his help with integrating Motion Picture Parser functionality. Equipment was provided by the Multimedia Communications Laboratory of Boston University. Finally, thanks to everyone who contributes quality software to the Internet community.

References

[1] W. Mackay, and G. Davenport, "Virtual Video Editing in Interactive Multimedia Applications," *Communications of the ACM, July 1989*, Vol. 32, No. 7, pp. 802-10.

[2] L. Hardman, G. van Rossum, and D. Bulterman, "Structured Multimedia Authoring," *ACM Multimedia 93 Proceedings*, June 1993, pp. 283-9.

[3] G. Drapeau, and H. Greenfield, "MAEstro - A Distributed Multimedia Authoring Environment," *USENIX Summer '91 Proceedings*, pp. 315-28.

[4] E. Deardorff, T.D.C. Little, J.D. Marshall, D. Venkatesh, and R. Walzer, "Video Scene Decomposition with the Motion Picture Parser," *IS&T/SPIE Symposium on Electronic Imagery Science & Technology (Digital Video Compression and Processing on Personal Computers: Algorithms and Technologies)*, San Jose, February 1994, SPOE Vol. 2187, pp. 44-55.

[5] D. LeGall, "MPEG: A Video Compression Standard for Multimedia Applications," *Communications of the ACM*, April 1991, Vol. 34, No. 4, pp. 47-58.

[6] E. Oomoto, and K. Tanaka, "OVID:Design and Implementation of a Video-Object Database System," *IEEE Transactions on Knowledge and Data Engineering*, August 1993, Vol. 5, No. 4, pp. 629-643.

[7] L. Rowe, J. Boreczky, and C. Eads, "Indexes for User Access to Large Video Databases," IS&T/SPIE Symposium on Electronic Imagery Science & Technology (*Digital Video*

Compression and Processing on Personal Computers: Algorithms and Technologies), San Jose, February 1994, Vol. 2187, pp. 1-12.

[8] IEEE Std 1058.1-1987, IEEE Standard for Software Project Management Plans.

[9] W.W. Royce, “Managing the Development of Large Software Systems: Concepts and Techniques,” *WESCON Proceedings*, August 1970.