Content-Based Access to Algebraic Video

by

Ron Weiss

B.A. Brandeis University (1992)

Submitted to the Department of Electrical
Engineering and Computer Science in Partial
Fulfillment of the Requirements for the Degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Signature of Author ______________________________
Department of Electrical Engineering and Computer Science
May 17, 1994

Certified by ______________________________
David K. Gifford
Associate Professor of Computer Science
Thesis Supervisor

Accepted by ______________________________
F. R. Morgenthaler
Chairman, Departmental Committee on Graduate Studies

Content-Based Access to Algebraic Video

by

Ron Weiss

Submitted to the Department of Electrical
Engineering and Computer Science on May 13, 1994
in partial fulfillment of the requirements for the
degree of Master of Science in Electrical
Engineering and Computer Science

Abstract

Algebraic video integrates fundamental access methods for digital video: composition, search, navigation and playback. Video presentations are composed using a *video algebra* that consists of a set of basic operations on video segments to produce a desired video stream. The video algebra contains operations for temporally and spatially combining video segments as well as for attaching attributes to these segments. Algebraic video access methods also include query and navigation operations. Query and navigation allow users to discover video presentations of interest by describing desired attributes and exploring a presentation's context. Unlike previous approaches, algebraic video permits video expressions to be nested in arbitrarily deep hierarchies. It also permits video segments to inherit attributes by context. Experience with a prototype algebraic video system suggests that algebraic video offers a complete, integrated framework to access and manage video, is easy to use, and that satisfactory performance is obtainable. The prototype system is used to discover video segments of interest from existing collections and create new video presentations with algebraic combinations of these segments.

Thesis Supervisor: David K. Gifford
Title: Associate Professor of Computer Science

Acknowledgments

I would like to thank my advisor, David K. Gifford, for his advice, encouragement, and support. He has been extremely helpful in guiding me, helping focus my thoughts, and comprehend the important issues.

I would like to acknowledge Dr. Andrzej Duda for his many contributions and important insights.

I would like to thank my group members, Brian Reistad, Mark Sheldon, and James O'toole for their many insightful comments.

I would like to thank my father, Dr. Zvi Weiss, for introducing me to the world of computers, and for my entire family for their encouragement and support.

I wish to acknowledge Professor Gerald Winer who provided me with helpful comments about my writing style.

I would like to thank my roommate Mike Ashburn whose endless hours of work and encouragements when the going got tough have motivated me to continue on.

And finally I'd like to thank my fiancee Kim Winer, that was always there for me, and although physically far away, I was never closer to.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The video and audio capabilities of current computer systems are advancing rapidly. As digital video becomes ubiquitous and as more video sources become available, applications will need to deal with digital video as a new data type. However, since video has both temporal and spatial dimensions, it places different requirements on applications than existing data types such as text. Moreover, the volume and unstructured format of digital video data make it difficult to manage, access and compose video segments into video presentations. When creating new video presentations, it is essential to reuse existing video segments and presentations, because the sheer volume of the data makes copying prohibitive. It is therefore necessary to provide a new digital video data type with content-based access that will alleviate these problems and facilitate broader use of video resources.

Many existing digital video abstractions rely on the traditional view of video as a linear temporal medium. They do not take full advantage of either the logical structure of the video or of hierarchical relationships between video segments. Moreover, flexible associative access based on the structure and the hierarchy is not supported. For these reasons, *algebraic video* allows the user to:

- create video presentations that
  - model nested video structures such as shot, scene and sequence,
  - express temporal compositions of video segments,

  - define output characteristics of video segments,
  - specify multi-stream viewing.
- integrate content-based access to video:
  - associate content information with logical video segments,
  - provide multiple coexisting views and annotations of the same data,
  - provide associative access based on the content, structure and temporal information.
- playback video presentations

The goal of algebraic video to help users efficiently access and manage digital video. To test this goal, a prototype system that implements algebraic video has been built. Algebraic video and a system that implements this model must meet the following criteria:

1. **Complete, Integrated Framework**: The model must support the fundamental access methods to video in an integrated fashion. Users should be able to efficiently find relevant video presentations, playback these presentations, and compose new presentations of their own. In supporting the creation of new video presentations, the data model must be sufficiently flexible and powerful to model many diverse types of video presentations, and also allow the association of content with video presentations for subsequent content-based access.

2. **Ease of use**: The data model and the system that implements the data model must be easy to use in the creation of algebraic video, during the search process, and during the playback of video presentations.

3. **Performance**: Query and video playback should provide satisfactory performance. Although absolute performance is not a goal of this thesis, the system must provide acceptable video viewing and support interactive query access.

The rest of this chapter describes motivating examples and the requirements they place on algebraic video (Section 1.1), provides a design overview of the *algebraic video* data model (section 1.2), and summarizes the contributions of this research (Section 1.3).

1.1 Possible Applications

The algebraic video data model can provide the foundation for a digital video system where users efficiently access and manage video information. For example, consider a user that wants to compose a digital video presentation about the latest developments with economic reforms. First, the user searches a large collection of TV broadcasts for all video segments reporting on economic reforms and Smith, who is a notable economist. Then, the user may want to examine the context in which the segments have appeared: in the headline news or in a talk show. Finally, the user chooses some segments and combines them such that they form a new video presentation that can be played out, stored, or exchanged.

Another example is a user who wants to automate the daily task of composing and viewing a short video presentation that includes the day's interesting events. The user encodes his or her preferences once, and then allows the computer to automatically construct a relevant video presentation on a daily basis. A system that supports this scenario must allow the user to easily express certain desired characteristics of the video presentation. First, the user encodes the preferred content of the video footage, such as economic news, latest basketball scores, speeches by the president, and the current weather forecast. Second, the user must be able to express temporal constraints on the duration of the entire video presentation or any of its subsegments. Third, the user must be able to express the spatial and temporal composition of the relevant video segments, possibly in a multi-window configuration that includes several concurrent video streams. For instance, more important footage such as the economic news will be placed in larger windows and closer to the beginning of the presentation. Fourth, the user will not want to view the same video footage again if it occurs earlier in the video presentation. Finally, a system supporting this interaction must be easy to use by anyone, including non-programmers.

The above examples require a new video data type that integrates both the content attributes and the semantic structure of the video data. For content-based access, the video data type may need to describe the people in a scene, the associated verbal communication for each video segment, and the relationships between segments. Automatic content extraction, such as image and speech recognition should be used when possible. Because these methods are not yet generally feasible, other forms of information extraction can be employed. Text captions or image features such as color, texture, and shape, may be associated with the video footage. The user can also associate personalized descriptions with any component of the video presentation. Finally, the data model allows users to express temporal and structural relationships between video segments. Along with content information, these semantic structure attributes are also indexed for content-based access.

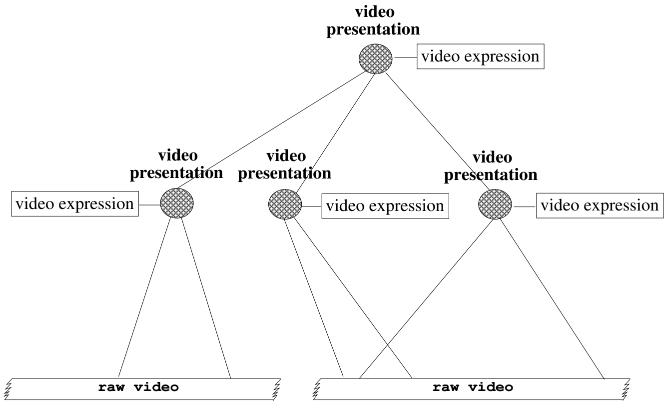

Figure 1-1: Video Expressions Denote Video Presentations

1.2 Design Overview

The *algebraic video* data model allows users to create video presentations using hierarchical compositions of *video expressions* with high-level semantic descriptions. A video expression denotes a *video presentation*, which is multi-window, concurrent spatial and temporal combination of video segments (see Figure 1-1). The video expressions are constructed using *video algebra* operations. A video algebra is introduced as a means for combining and expressing temporal relations, for defining the output characteristics of video expressions, and for associating descriptive information with these expressions. The algebraic video abstraction provides an efficient means of organizing and manipulating video data by assigning logical representations to the underlying video streams and their contents. The model also defines operations for flexible associative access to the video information. The algebraic video preserves the correspondence between video segments so that all relevant segments and their neighbors can be efficiently found. Video expressions define media-independent output characteristics, and therefore the rendering can adjust to the available resources.

Users access algebraic video via query, navigation, and playback of video presentations. These access methods to an algebraic video collection are grouped together as the *interface operations*. Users can search for relevant presentations with queries that describe desired attributes of video expressions. The invocation of a query results in a set of video expressions that can be played back, reused or manipulated by a user. In addition to content-based access, the interface operations allow users to browse and explore the structure of the video expressions. These activities help users understand the surrounding organization and context. For example, the user can find an interesting expression and then examine the encompassing video segments. Furthermore, users can create their individual interpretations of existing video footage by composing new video expressions from the existing video components.

1.3 Algebraic Video Contribution

The algebraic video data model allows users to compose concurrent video presentations by structuring raw data into logical video segments and then describing the temporal relations between these segments. Hierarchical relations between the video expressions allow *nested stratification* — overlapping segments are used to provide multiple coexisting views and annotations of the same data and enable the user to assign multiple meanings to the same footage. Segments can be organized hierarchically so that their relationships are preserved and can be exploited by the user. In addition to simple stratification, the algebraic video model preserves nested relationships between strata and allows the user to explore the context in which a stratum appears (see discussion in Section 2.1).

The algebraic video data model offers the following important advances over previous digital video representations:

- It provides the fundamental functions required to deal with digital video: composition, reuse, organization, searching, and browsing.

- It models complex, nested logical structure of video using video algebra. The video algebra is a useful metaphor for expressing temporal interdependencies between video segments, as well as associating descriptions and output characteristics with video segments.

- The model allows associative access based on the content of the video, its logical structure and temporal composition.

The *Algebraic Video System* is a prototype implementation of the algebraic video model and its associated operations. It facilitates experimentation with the algebraic video data model, and helps evaluate the feasibility of the design goals in the construction of a real production system. The system allows users to compose and playback algebraic video presentations. It extracts video attribute information and supports content-based access, as well as video playback. The system offers an integrated environment that includes a text based video expression editor, query based interface for searching, methods for browsing, and an algebraic video player for playback of relevant video presentations. The algebraic video player uses the logical representation of the video data to provide viewing methods based on the ascribed temporal characteristics of the video.

The remainder of this thesis discusses related work (Chapter 2), the design of the algebraic video data model (Chapter 3), the prototype implementation (Chapter 4), the user interface to the system (Chapter 5), and experience with the system and conclusions from the work (Chapter 6).

Chapter 2

Related Work

This chapter describes related work that includes video authoring and annotation tools (Section 2.1), systems that provide content-based access to video (Section 2.2), modeling of unstructured video for content-based retrieval (Section 2.3) and modeling the temporal component of information (Section 2.4). The author is not aware of any system that currently supports a complete, integrated framework for managing and accessing digital video as offered by algebraic video.

2.1 Video Authoring and Annotation

Video authoring and annotation tools provide facilities for composing and annotating complex video presentations. Davenport *et al.* [4, 3] implemented a video annotation system that uses the concept of *stratification* to assign descriptions to video footage, where each stratum refers to a sequence of video frames. The strata may overlap or totally encompass each other. They are stored in files and can be accessed using simple keyword search. Figure 2-1 shows an example of video footage annotated by strata. In the stratification system, a user can find a sequence of interest, but cannot easily determine the context in which the video sequence appears. due to the absence of relationships between the strata. The algebraic video data model provides a full hierarchical organization of video footage that permits flexible browsing. A user creates strata with video nodes that reference portions of video segments. In addition to modeling simple stratification, algebraic video preserves the nested relationships between strata and allows the

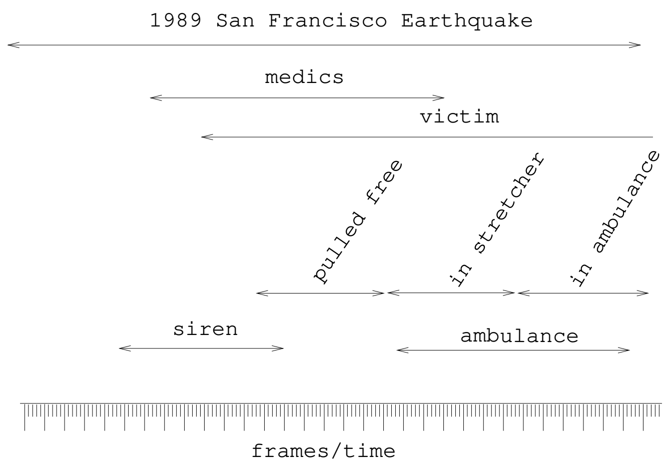

Figure 2-1: Annotation of Video Footage using Stratification

exploration of the context in which a stratum appears. Furthermore, the algebraic video data model allows the association of arbitrary, possibly non-textual, attributes with the video data. Section 3.3 discusses in detail the advantages of algebraic video over linear stratification.

Adobe Premiere [2], DiVA VideoShop [11], and MacroMind Director [25] that are built upon the QuickTime [12] video support system, and Avid Media Composer [6] that relies on a proprietary video system, are commercially available video authoring tools. They allow the user to create movies using audio and video tracks, and also enable the user to specify special effects during video segment transitions. These commercial systems are based on two distinct paradigms: *timelines* and *scripts*. In the timeline approach, video and audio objects are placed on a line representing time flow. Video objects are normally a sequence of frames that may have an associated audio stream. Prerecorded audio streams can also be independently placed in the timeline. Normally, a direct manipulation graphical editor similar to the one illustrated in Figure 2-2 (artificially created for the purposes of this discussion), presents the video author with video and audio tracks, and a special effects track for combining the two video tracks. Synchronization between any two objects is achieved by carefully placing the video and audio objects on tracks that are marked by time indices. The indices reflect the elapsed time since the beginning of the video presentation. Any presentation created on a timeline can be easily mapped into an algebraic video presentation. However, some algebraic video presentations, such as ones that include choices, cannot be modeled using a simple timeline metaphor.

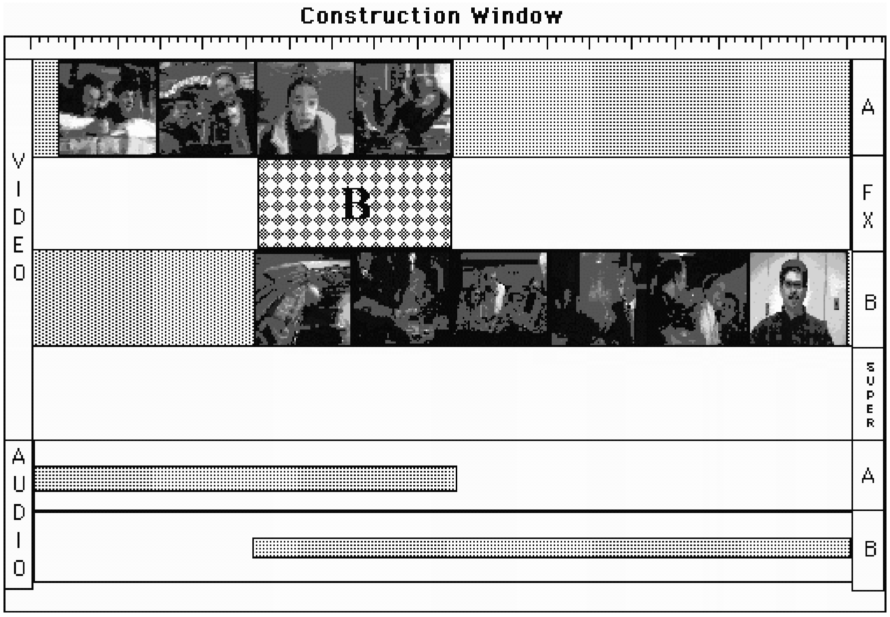

Figure 2-2: Timeline Editor

The script (or *flowchart*) approach requires the video author to explicitly program timing and placement information. The toolkits allow the user to edit video data in essentially the same manner as film makers edit analog movies. They arrange shots on a temporal linear axis by cutting, pasting and making transitions. The computer merely simplifies the previously mundane task of searching for a sequence of frames from a video source such as a video tape, and then copying the frames onto the video target. Digital video is unique because it is not restricted by the linearity of traditional media. It possesses a dynamic element, where the video display may be determined during runtime and may not follow a strictly linear progression determined *a-priori*. The toolkits do not take advantage of this distinctive feature. Moreover, the toolkits lack methods for specifying the elaborate logical structure of video data and do not address content-based access. Because perusing through a large video collection in search of some footage is inherently time-consuming, content-based access should be incorporated as an integral part of the video creation and access process. The algebraic video approach allows structured, multi-stream composition using video algebra operations and content-based access.

Multimedia authoring systems such as CMIFed [36, 19] propose structuring primitives for multimedia documents. They define an *atomic multimedia presentation* in terms of *events*, where an event is usually a small fragment of video, audio or text. Then, *composite* presentations contain other, possibly nested *presentations*. Events are mapped into *channels*, which are a media specific abstraction for a group of events. Synchronization is achieved using parallel and sequential composition, as well as synchronization arcs that specify constraints between two events in the same presentation. The model fails to address the structure of the video data itself because video is still treated as an unstructured linear stream. Although the model contains a structuring mechanism for the multimedia presentation that can be examined with the hierarchy view, it does not allow multiple coexisting views of the data that enables flexible annotation and content-based access.

Hamakawa and Rekimoto [18] propose a multimedia authoring system that supports editing and reuse of multimedia data. Their system is based on a hierarchical and compositional model of multimedia objects. It allows the user to mark objects with a title at a certain point in time. However, it does not support a fully functional free form annotation mechanism that enables subsequent content-based access. Similar to the CMIFed system, it does not allow multiple coexisting views of the same data.

Media Streams is an iconic visual language that enables users to create multi-layered, iconic annotations of video content [10]. Icons denoting objects and actions are organized into cascading hierarchies from levels of generality to levels of increasing specificity. Additionally, icons are organized across multiple axes of descriptions such as objects, characters, relative positions, time or transitions. The icons are used to annotate video streams represented in a Media Time Line. An icon that is placed on the timeline annotates the video segment from its insertion point to a specified end point, that could be a scene break or the end of the video stream. Currently, around 2200 iconic primitives can be browsed. However, this user-friendly visual approach to annotation is limited by a fixed vocabulary. For instance, textual data such as close-captioned text can not be readily associated with the video stream. Moreover, the approach suffers from the same limitations of the timeline approach discussed above. It is still restricted to a linear medium, rather than exploiting the non-linearity that characterizes digital video. While the iconic primitives can be combined to produce compound annotations, the annotations themselves do not reveal the logical structuring of the hierarchical and possibly nested video presentation.

VideoScheme [26] is a programmable video editing system that combines the metaphor of direct manipulation video editing and a programming language. The system can be used to automate mundane editing tasks, as well as easily programmed to perform more complex tasks such as object and media recognition by efficient prototyping of novel algorithms. At the core of the system is a simple direct manipulation video editor that includes video and audio tracks. The Scheme programming environment that is embedded in the interactive editor allows users to type and evaluate expressions, which in turn affect the editor. However, rather then incorporating video as a fundamental data type within a programming language, VideoScheme chooses to provide a library of procedures that manipulate a timeline based video editor. The system does not directly support the logical structuring of video, and ultimately suffers from the same disadvantages of other timeline representations.

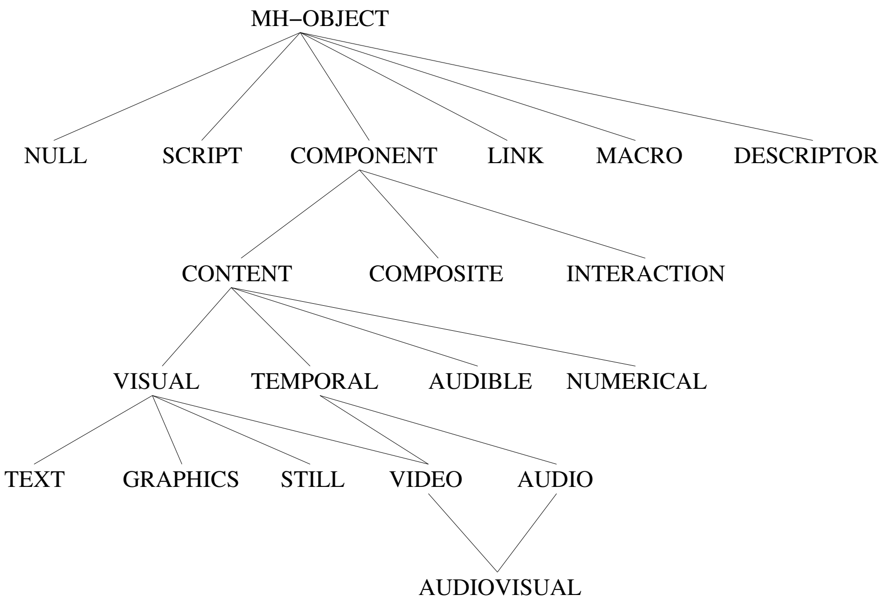

Figure 2-3: The MHEG Object Inheritance Tree

The MHEG [29] standard is intended for "*coded representation of final form multimedia and hypermedia objects that will be interchanged across service and applications*". At the core of the standard are the MHEG objects (represented in Figure 2-3) that play a federated role between interacting applications. MHEG defines the formats used at the interchange point between applications that want to exchange multimedia data. The objects are synchronized and composed to form complex presentations using four mechanisms: script, conditional activation, spatio-temporal, and close system synchronization.

The HyTime [31] hypermedia standard provides a mechanism to specify hyperlinks and schedule multimedia information in time and space. It is based on the Standard Generalized Markup Language (SGML), using "architectural forms" to express rules for hypermedia structuring information. These architectural forms and attributes of information objects are grouped into six modules: Base module, Measurement module, Location address module, Hyperlinks module, Scheduling module and a Rendition module. The scheduling module allows events, which are occurrences of information objects, to be scheduled in "finite coordinate spaces" (fcs). The user expresses spatial and temporal positions of objects in fcs's using coordinate axes or relationships. The rendition module maps the fcs representation to its "real-world" counterpart to achieve playback of the multimedia information. As stated in their description, both MHEG and HyTime are intended for final formatted documents, and lack mechanisms for content-based access, editing and annotation of the multimedia data. Koegel *et al.* implemented HyOctane [20], a multimedia information system based on the HyTime standard. They provide a sample HyTime document type definition to model multimedia slideshow presentations.

2.2 Systems with Content-Based Access to Video

Content-based access systems provide facilities to discover video segments of interest. These systems normally define a data schema to represent a video presentation, and support user queries formulated against information associated with elements of this data schema. Little *et al.* [23] implemented a system that supports content-based retrieval and playback of video footage. They defined a specific data schema composed of *movie, scene* and *actor* relations with a fixed set of attributes. The system requires manual feature extraction, and then fits these features into the data schema. Queries are permitted on the attributes of movie, scene and actor. Once a movie or a scene is selected, a user can scan from scene to scene beginning with the initial selection.

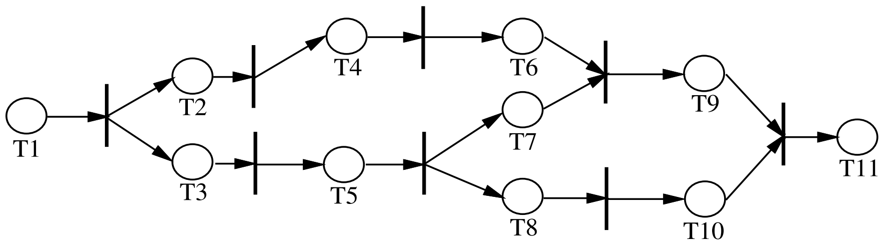

Figure 2-4: Timed Petri Nets

The system implemented by Little *et al.* defines a fixed temporal schema for representing the logical structure of a motion picture. The schema includes a hierarchical organization of *shots* that combine to form *scenes*, where several *scenes* are arranged to form a *movie*. This data schema is used to construct a Timed Petri Net (TPN) representation of the movie, such as the one in Figure 2-4. The system uses the TPN for synchronization of the video and audio objects during playback. The TPN representation itself does not include any logical structuring of the presentation. In addition, it does not support multiple coexisting views of the same data, and is therefore not sufficiently expressive for annotation and subsequent content based access. In addition, the data model and the system's Virtual Video Browser are limited for several reasons. First, descriptions cannot be assigned to overlapping or nested video sequences as is accomplished in the stratification model. Second, the system is focused on retrieving previously stored information and is not suitable for users that need to create, edit and annotate a personally customized view of the video footage. For example, users cannot create a new movie from the collection of scenes that are returned as a result of a query. Moreover, the browser does not support queries based on the temporal ordering of scenes.

Electronic Scrapbook is a system for home-video video annotation and editing [9], where the annotations can later be used for content-based access. The user can attach descriptions to video clips and use a modified form of case-based reasoning to edit and create personalized video stories. The user can query a database of video clips and also filter, sort, or remove overlapping segments from the results. The system uses a small, special-purpose taxonomy that can be used in descriptions, but does not exploit the logical structure of video. For example, the user cannot describe hierarchical relationships where video segments are nested.

Gibbs *et al.* [14] propose an object-oriented approach to video databases. An *audio/video database* can be viewed as a collection of *values* (audio and video data) and *activities* (interconnectable components used to process values). The temporal and flow composition mechanisms allow aggregation of values and activities. The model supports a flow composition of a video presentations using queries against the database to specify the audio and video values. The system evaluates a query, and selects to play back the video results of the queries. Since these database values are linear sequences of data elements, their logical structure is not represented. Also, the temporal composition mechanism is essentially equivalent to the timeline paradigm.

2.3 Modeling of unstructured video for content-based retrieval

When video is captured into digital format from an analog source, it initially exists as an unstructured sequence of video frames and audio segments. Several proposed systems extract information from these unstructured streams and then provide a data model that is used for content-based access. Swanberg *et al.* [32, 33] defines such an architecture for parsing data semantics from the video stream. The system manages a fixed data schema for representing information about the video stream, where a shot is defined as a sequence of frames without a scene change, and an *episode* is a sequence of shots that are somehow related. The system provides tools and models to aid in the analysis of a video stream, including support for identification of *shots* and *episodes*. A knowledge module maintains the information about the segmentation of the video footage, information about the objects and features in the video, and information to facilitate query optimization. The data schema used for this system is not sufficiently flexible and therefore not suitable for free form modeling of the complex relations between video segments.

Nagasaka [27] implemented a system that automatically indexes video by detecting cuts and associating a small icon of a representative frame with each subpart. The list of icons is used as an index of the video. Additionally, the system supports full-video searches for frames in which a specified object appears. Queries are accomplished using an image of the reference objects.

2.4 Modeling the temporal component of information

The consideration of the temporal aspect of information is not restricted to the field of multimedia. Earlier work examined the problem of representing temporal knowledge with sufficient expressive power, while still allowing efficient deductive performance. Allen [5] introduced an interval-based temporal logic that supported an efficient reasoning algorithm. The primitive element in the knowledge representation is the *temporal interval*. The representation then defines a method for expressing relationships between temporal intervals in a hierarchical fashion. Efficient reasoning and deduction are based on constraint-propagation techniques. As illustrated in Figure 2-5, the work also defines the possible relationships between two primitive temporal intervals. Including the inverses of each relationship described in this figure (where *equal* is its own inverse), there is a total of thirteen possible mutually exclusive relationships between any two temporal intervals.

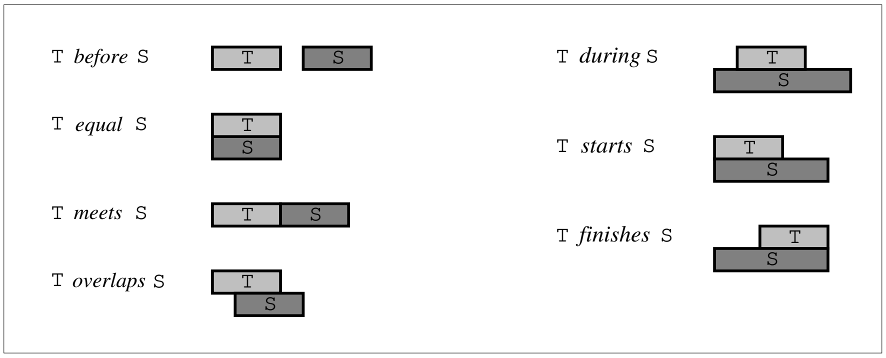

Figure 2-5: Relationships Between Temporal Intervals

Fiume *et al.* [13] defines a temporal scripting language for object-oriented animation. A user of their system begins with a library of animated objects that have autonomous spatio-temporal behavior. When constructing an animation, the user globally coordinates the activities of these objects using the proposed temporal scripting language. The approach allows abstraction over animated objects, reuse of objects, as well as extensibility to other media types. The language defines an object protocol consisting of two messages that enables the expression of a binary general synchronization operator between objects, as well as other forms of temporal scheduling.

Chapter 3

Design

The design goal of algebraic video is to provide a high-level abstraction that models complex structuring and content information associated with digital video data and supports content-based access. Motion video is more that just a sequence of frames randomly pieced together. Video is created from a collection of raw footage through editing and logical structuring to convey some message to the viewer. To achieve this narrative goal, video is generally arranged into different story units such as *shots, scenes and sequences* according to some logical structure [24]. A *shot*, which is the basic organizational entity, is a collection of video frames recorded sequentially. One or more related shots are combined in a *scene* and a series of related scenes forms a *sequence*. The screenplay defines the logical structure of the video by organizing the different story units. It provides additional content and structural information through detailed descriptions of the scenes and sequences.

The video footage itself also contains complex content information that can be extracted and associated with the video story units. For example, attributes such as close-captioned text and key frames that characterize a shot can be associated with video. Other attributes that can be extracted from the footage and provide information about the video include dominant color in a shot, estimation of motion, and the audio stream. Thus, video has two components that define it: the logical structure of the video entities, such as shots, and the actual content of the footage.

To obtain the design goal, the data type must be able to:

- model nested video structures such as shot, scene and sequence,
- express temporal relationships between video segments,
- associate content information with logical video segments,
- provide multiple coexisting views and annotations of the same data,
- define output characteristics of video segments.
- provide associative access based on the content, structure and temporal information.

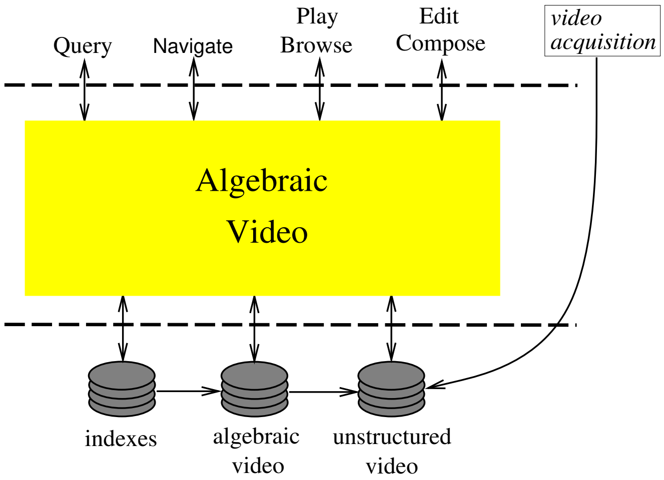

Figure 3-1: Interface and Facilities of Algebraic Video

Interaction with algebraic video is accomplished via four activities as illustrated in Figure 3-1: *edit* and *compose* video presentations, *play* and *browse* existing algebraic video presentations, *navigate* the video hierarchy and *query* to find relevant video. The operations that support playback, navigation and content based access are grouped together as the *interface* operations. The storage subsystem includes raw, unstructured video, a representation of algebraic video, and indexes to support content based access. The process of video acquisition that involves digitizing video or acquiring digital video from other sources is not specified by the algebraic video data model.

The fundamental entity of the algebraic video model is a *presentation.* A presentation is a multi-window spatial, temporal, and content combination of video segments. Presentations are described by *video expressions.* The most primitive video expression involves the creation of a single-window presentation from a raw video segment. These segments are specified by identifying the raw video and a range within the raw video. Compound video expressions are constructed from simpler ones using *video algebra* operations. Video expressions can be named by variables, composed to reflect the complex logical structure of the presentations, and share the same video data. A video expression may contain composition information, descriptive information about the contents, and output characteristics that describe the playback behavior of the presentation. Video expressions can be played back, searched and browsed. An *algebraic video node* provides a means of abstraction by which video expressions can be named, stored and manipulated as units. An algebraic video node contains a single video expression that may refer to children nodes or raw video segments (see Figure 1-1).

The remainder of this chapter introduces the algebraic video operations for editing and composition (Section 3.1), presents the interface operations for managing and accessing video presentations (Section 3.2), describes nested stratification as an effective means for content-based access (Section 3.3 and concludes by considering the model as a programming language (Section 3.4).

3.1 Editing and Composing Using Video Algebra Operations

The video algebra operations for editing and composing video presentations are classified into the following categories:

- **Creation**: defines the construction of video expressions from raw video.
- **Composition**: defines temporal relationships between component video expressions.
- **Output**: defines spatial layout and audio output for component video expressions.
- **Description**: associates content attributes with a video expression.

	Creation
create	`create` *name begin end* creates a presentation from the range within the identified raw video segment
delay	`delay` *time* creates a presentation with empty footage for duration *time*
	Composition
concatenation	$E_1 \circ E_2$ defines the presentation where E_2 follows E_1
union	$E_1 \cup E_2$ defines the presentation where E_2 follows E_1 and common footage is not repeated
intersection	$E_1 \cap E_2$ defines the presentation where only common footage of E_1 and E_2 is played
difference	$E_1 - E_2$ defines the presentation where only footage of E_1 that is not in E_2 is played
parallel	$E_1 \parallel E_2$ defines the presentation where E_1 and E_2 are played concurrently and start simultaneously
parallel-end	$E_1 \wr\wr E_2$ defines the presentation where E_1 and E_2 are played concurrently and terminate simultaneously
conditional	(`test`) ? $E_1 : E_2 : \ldots : E_k$ defines the presentation where E_i is played if `test` evaluates to i
loop	`loop` E_1 *reps* defines *reps* repetitions of the video expression E_1 (can be *forever*)
stretch	`stretch` E_1 *factor* the duration of the presentation is equal to *factor* times duration of E_1. This is achieved by changing the playback speed of the video expression.
limit	`limit` E_1 *time* the duration of the presentation is equal to the minimum of *time* and the duration of E_1, but the playback speed is not changed.
transition	`transition` E_1 E_2 *type time* defines *type* transition effect between expressions E_1 and E_2; *time* defines the duration of the transition effect
contains	`contains` E_1 *query* defines the presentation that contains component expressions of E_1 that match *query*
	Output
window	`window` E_1 $(x_1, y_1) - (x_2, y_2)$ *priority* specifies that E_1 will be displayed with *priority* in the window defined by (x_1, y_1) as the bottom-left corner, and (x_2, y_2) as the right-top corner such that $x_i \in [0,1]$ and $y_i \in [0,1]$
audio	`audio` E_1 *channel force priority* specifies that the audio of E_1 will be output to *channel* with *priority*. If *force* is true, override `audio` specifications of the component expressions.
	Description
description	`description` E_1 *content* specifies that E_1 is described by *content*
hide-content	`hide-content` E_1 defines a presentation that hides the content of E_1

Table 3.1: Video Algebra Operations

Table 3.1 presents the video algebra operations. The arguments denoted by E_1, $E_2, \ldots, E_k$ are video expressions. The result of a video expression is a presentation. A video expression defines the temporal and spatial composition of its *presentation* arguments using the operators defined in the table. For the examples given in this chapter, expressions of the form: $(E_1 \odot E_2 \odot E_3 \odot \ldots E_n)$, where $\odot$ is any specific binary operation, denote an expression of the form: $(\ldots((E_1 \odot E_2) \odot E_3) \odot \ldots E_n)$. Also note that the binary video algebra operations are inherently not associative because they include a temporal component.

3.1.1 Composition

Complex scheduling definitions and constraints can be expressed using the composition operators. *Concatenation* is the simplest temporal combination of presentations. The video expression $E_1 \circ E_2$ defines the compound video presentation where the presentation E_2 follows the presentation E_1. The *union* operation is similar to *concatenation* in the sense that it also creates a compound video presentation by sequentially ordering the argument video presentations. However, the resulting presentation is not merely a temporal ordering of the arguments. Rather, the *union* operator eliminates some redundancy between the arguments in a manner similar to the mathematical union set operation (and unlike the the multi-set union operation). The resulting video presentation is a combination of the video presentation arguments where common footage is not repeated. Section 4.2.1 offers a precise definition of one possible interpretation of the common footage elimination process as implemented in the algebraic video system. The *union* operation allows the user to easily construct a non-repetitive video stream from overlapping segments. It preserves the temporal ordering of the component presentations. Note that if these expressions do not contain overlapping segments, then *union* is equivalent to *concatenation.* Figures 3-2 and 3-3 present an example of an algebraic video node that uses the *union* operation in the composition of a video expression. In this example, one raw video file is annotated by three overlapping nodes. The *union* of the three overlapping nodes yields one video stream with no redundancy in the playback.

The *intersection* operation defines a new video presentation that includes only footage that is contained in both of the arguments. Thus, it enables the user to easily construct video presentations that incorporate only the footage that is the overlap of multiple video segments

C_1 = **create** Cnn.HeadlineNews.rv 10 30
C_2 = **create** Cnn.HeadlineNews.rv 20 40
C_3 = **create** Cnn.HeadlineNews.rv 32 65

(**description**
  $(C_1 \cup C_2 \cup C_3)$
  (title = "CNN Headline News"
   text = "Smith proposes economic reform ..."))

Figure 3-2: Union Operation in an Algebraic Video Node

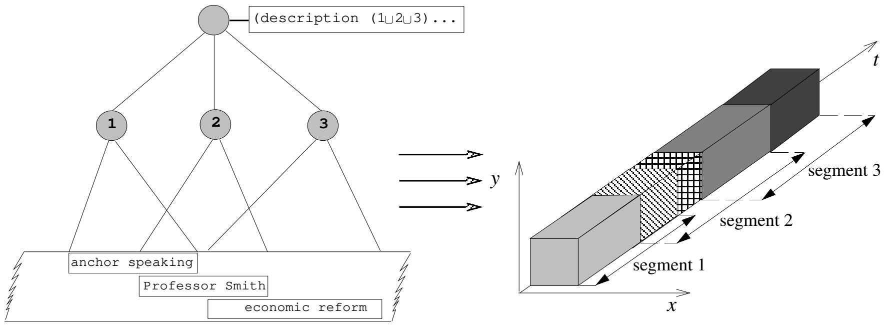

Figure 3-3: Graphical Representation of Union

or presentations. For example, the *intersection* operation can be used to efficiently determine whether a video presentation contains any portion of a particular video segment. The *difference* operation defines a video presentation that is similar to its first argument, except that any video footage contained in its second argument is not included. The *union*, *intersection* and *difference* operators do not only determine scheduling constraints for combining video presentations. Rather, they also examine the contents of these component presentations, and produce new presentations whose contents and scheduling depends on the contents of the video presentation arguments. These operations are different from the normal mathematical set operations because they also have a temporal component and are therefore not associative.

The *parallel* and *parallel-end* operations allow the user to compose multi-window, concurrent video presentations. The *parallel* operation defines a compound video presentation where the argument presentations are scheduled to start concurrently. The *parallel-end* operation allows an alternate synchronization mechanism for composing compound video presentations where the

```
C1 = create Hoffa.uv 0:10 3:50
C2 = create ScentOfAWoman.uv 0:00 5:40
C3 = create ScoobyDoo.uv 20:12 40:00
C4 = create Alladin.uv 12:00 17:25

(concat
  C4
  (conditional
    % Beginning of test expression written in TCL
    puts stdout "Please select movie ending"
    puts stdout " (1 = sad, 2 = happy, 3 = Scooby-Doo): "
    set c [read stdin 2]
    return "$c"
    %
    C1 C2 C3)
```

Figure 3-4: An example of the Conditional Operator

video presentation arguments are scheduled to terminate simultaneously. The video algebra can be extended to include a generalized synchronization operator that allows the specification of temporal relationship constraints based on intermediate time points between the start and end of a video expression.

The *conditional* operation defines a video presentation where the choice of what to display is made at runtime. Thus, the same video presentation can result in completely different video streams depending on dynamic expression evaluation. The test expression in the conditional operation must evaluate to an integer. However, it is easy to map non-integer test expressions, such as a user's environment variable, time-of-day, weather patterns, and user interaction, to valid integers. The *conditional* operation can be used in the domain of interactive movies where a user creates her own story by choosing to explore different possible plot threads. This can be accomplished by logically structuring the video and allowing the user to choose segments based on interaction or an *a priori* specification. Figure 3-4 illustrates how the *conditional* operation can be used for personalized viewing or other viewing that can be affected by external sources. In this example, the user chooses the ending of the presentation. The *conditional* operation can also introduce non-determinism into the playback of video presentations by instructing the *test* expression to return the result of a random number generator.

The *loop* operation creates a compound video expression that repeats the video presentation argument a specified number of times. The *stretch* operation changes the playback speed of the video presentation, but does not alter the playback speed of other presentations. The *limit* operation creates a video presentation whose duration does not exceed the minimum value of the *time* argument and the duration of the video presentation argument. An author of a video presentation can easily constrain the playback duration of video presentations with the limit operation to fit some prespecified temporal constraint, such as a thirty second time slot for a news clip. The *transition* operation combines two video expressions using a transition effect of duration *time*. The transition *type* is one of a set of transition effects, such as dissolve, fade, and wipe. Note that *concatenation* is a simple *transition* with $time = 0$.

The *contains* operation permits the user to define a video presentation based on the results of a query on a video expression argument. The operation combines the subexpressions that match the query into one video expression, while preserving the hierarchical relations of the video expression argument. The syntax and semantics of the *query* argument in a *contains* operation is explained in section 3.2.1. The following video expression is an example of the usage of the `contains` operator:

(`contains` "text:mafia" (JimmyHoffa.av ∘ FleshAndBone.av ∘ BusinessSense.av))

In this example, the resulting presentation includes only those video subexpressions which have the "mafia" text attribute. This could include the entire `JimmyHoffa.av` video node, as well as selected portions of `BusinessSense.av`.

3.1.2 Output Characteristics

Because multiple video streams can be scheduled to play at any specific time within one video presentation, the playback may require multiple screen displays and audio outputs. Therefore, video expressions include output characteristics that specify the screen layout and audio output for playing back children streams.

All video expressions are associated with some rectangular screen region in which they are displayed. A video expression constrains the spatial layout of its components. As expressions can be nested, the spatial layout of any particular video expression is defined relative to the parent rectangle. The parent rectangle is the screen region associated with the encompassing

C_1 = `create` Cnn.HN.127.Intro.rv 35 70
C_2 = Cnn.HN.3.14.Anchor.av
C_3 = Bosnia-7-14-93.av

P_1 = `window` C_1 (0,0) - (.7,1) 10
P_2 = `window` C_2 (0,0) - (1,1) 20
P_3 = `window` C_3 (.7,0) - (1,1) 30

$(P_1 \parallel P_3) \circ P_2$

Figure 3-5: An Algebraic Video Node with Parallel and Concatenation Operations

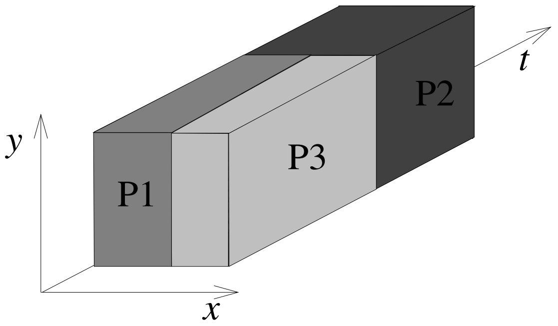

Figure 3-6: Playback of the Algebraic Video Node

expression. The *window* operator defines a rectangular region within the parent rectangle where the given video expression is displayed. The rectangular region is specified by two points in a relative coordinate system, the top-left (x_1, y_1) and bottom-right (x_2, y_2) corners, such that $x_i \in [0, 1]$ and $y_i \in [0, 1]$. By default, a video expression is associated with a square that fits in the parent rectangle. Figures 3-5 and 3-6 give a simple example of an algebraic video node and illustrate the playback characteristics of this node using spatial and temporal coordinates. Figures 3-7 and 3-8 give a more elaborate example of an algebraic video node with nested window specifications and a snapshot captured during playback.

Window priorities are used to resolve overlap conflicts of screen display. The *window* operation establishes the video priority of the associated window region with the *priority* parameter. The window with the higher priority overlaps the window with the lower priority. For example, assume that the two windows W_{c1} and W_{c2} are children of the same parent window region. If

C_1 = `create` hoffa.rv 30:0 50:0

P_1 = `window` C_1 (0,0) - (0.5,0.5) 10
P_2 = `window` C_1 (0,0.5) - (0.5,1) 20
P_3 = `window` C_1 (0.5,0.5) - (1,1) 30
P_4 = `window` C_1 (0.5,0) - (1,0.5) 40
$P_5 = (P_1 \parallel P_2 \parallel P_4)$
$P_6 = (P_1 \parallel P_2 \parallel P_3 \parallel P_4)$

(P_5 ||
 (`window`
  (P_5 || (`window` P_6 (0.5,0.5) - (1,1) 60))
  (0.5,0.5) - (1,1) 50))

Figure 3-7: An Example of Nested Windows

the priority of W_{c1} is greater than the priority of W_{c2}, then W_{c1} and all its video subexpressions will overlap W_{c2} and all its video subexpressions. Figures 3-9 and 3-10 give an example of a video presentation with nested window specifications that uses video priorities to resolve overlap conflicts.

The *audio* operation directs the audio output of the video expression to *channel*, which can be any logical audio device. If the *force* argument is true, then the *audio* operation overrides any *channel* specifications of the component video expressions. The *priority* parameter is defined in a manner analogous to the *priority* parameter of the *window* operation.

3.1.3 Associating Descriptions with Video Presentations

The model permits the association of arbitrary *descriptions* with a given video algebra expression. It allows textual, as well as non-textual descriptions such as key frames, icons, salient stills [35], and image features like color, texture, and shape. The *description* operation associates *content* information with a video expression.

The *content* description of an expression is not fixed by the model. However, for the purposes of this thesis and the prototype implementation, a *content* is a boolean combination of *attributes*, where each attribute consists of a *field* name and a *value*. An example of an attribute is `title = "CNN Headline News"`. Some field names have predefined semantics, e.g. `title`, and other

Figure 3-8: A Playback Snapshot of the Node with Nested Windows

fields can be defined by the user. Values can assume a variety of types, including strings and algebraic video node names. Field names or values do not have to be unique within a description. Therefore, a description can have multiple titles, text summaries, and actor names that are associated with a video expression. For example, a description may contain recorded close-captioned text. The user may add other attributes, such as actor, characters, and the scene summary. The components of a video expression inherit descriptions by context. This implies that all the content attributes associated with some parent video node are also associated with all its descendant nodes.

The *hide-content* operation defines a video expression E that does not contain any descriptions. The *contains* and *search* operations (described in section 3.2.1) on E do not recursively examine the components of E. The *hide-content* operation provides a method for creating abstraction barriers for content-based access. Thus, an author of a video node can provide a pointer to the node (i.e. it's name), while disallowing content-based access that examines the node's components. The node acts as a "black box" that can be combined and played together with other video expressions (including operations that examine it's footage contents, such as *union*, *intersection*, and *difference*). However, it does not allow the *contains* operation to select

$$
\begin{aligned}
C_1 &= \texttt{create}\ \text{Cnn.HeadlineNews.rv 10 50} \\
C_2 &= \texttt{create}\ \text{Cnn.Entertainment.rv 25 55} \\
C_3 &= \texttt{create}\ \text{Cnn.Sports.rv 12 27} \\
C_4 &= \texttt{create}\ \text{Nightline.rv 5 15} \\
C_5 &= \texttt{create}\ \text{LateNight.rv 60 65} \\
\\
W_4 &= \texttt{window}\ C_4\ (0.5,0)\ \text{-}\ (1,0.5)\ 5 \\
W_2 &= \texttt{Window}\ C_2\ (0,0)\ \text{-}\ (0.6,0.6)\ 10 \\
W_3 &= \texttt{window}\ ((C_3 \parallel W_4) \circ C_5)\ (.4,0)\ \text{-}\ (1,0.6)\ 20) \\
\\
&(C_1 \parallel W_2 \parallel W_3)
\end{aligned}
$$

Figure 3-9: Example of Nested Overlap

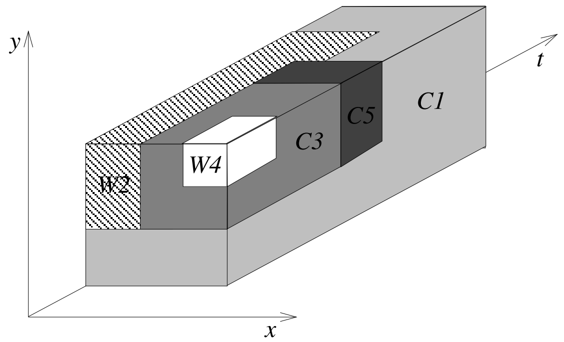

Figure 3-10: An Example of Nested Window Priorities

certain subcomponents solely based on their content description. For example, the *hide-content* operation allows the author of a presentation to include personal annotations that may not be used by others, unless they manually examine the node and reconstruct it. It also allows the video author to mark certain video subpresentations using the *hide-content* operator to disallow content-based access after the author has determined that the description information associated the presentation is not suitable for further consideration. This operation is not intended as a fool-proof security mechanism, but rather as a convenience feature when editing or searching for video.

Content-Based Access	
search	`search` *query* searches a collection of nodes for video expressions that match *query*
Browsing	
playback	`playback` *video-expression* playback the video expression
display	`display` *video-expression* display the video expression
Navigation	
get-parents	`get-parents` *video-expression* returns the set of nodes that directly point to *video-expression*
get-children	`get-children` *video-expression* returns the set of nodes that *video-expression* directly points to

Table 3.2: Interface Operations

3.2 Interface Operations

The interface operations on video expressions fall into three main categories: content-based access, browsing and navigation. Table 3.2 defines these operations. They are also discussed in the following subsections.

3.2.1 Content-Based Access

Associative access to video expressions is accomplished with the *search* operation, in which the user specifies desired properties of the expressions. A *search* is performed within the context of a persistent collection of algebraic video nodes. For querying within the collection, a simple predicate query language is used. A *query* is a boolean combination of attributes. When a query is applied to the collection in the *search* operation, the hierarchy of every node in the collection is searched recursively and the operation returns the *query result set* of nodes that satisfy the query. A node that can be revealed in more than one way is not searched more than once.

A description of a video expression is implicitly inherited by its subexpressions (which can be descendant nodes). The scope of a given algebraic video node description is the subgraph that originates from the node. Matching a query to the attributes of an expression must take into account all of the attributes of that expression, including the attributes of its encompassing expressions. In the case where the expression is an algebraic video node, this also includes the attributes of the ancestors. However, if a node's ancestor is in the result set, the descendant node is removed from the result set. This is done to ensure that complete sub-hierarchies of algebraic nodes are not returned in the result set of a query that matches some ancestor node. For example, consider the query `text:smith and text:question` applied to a collection that contains the node described in Figure 3-12. The result of the query is the node with the description `text:"question from audience"`, because this node implicitly contains the description `text:smith`. The node with the description `text:question` is not returned because it is a descendant of a node already in the result set.

Once a query result set is generated, the user can then playback any of the expressions in the set or browse and explore the video context and composition using the operations described in the previous section. For example, the user can inspect the encompassing video segment by examining the parent nodes.

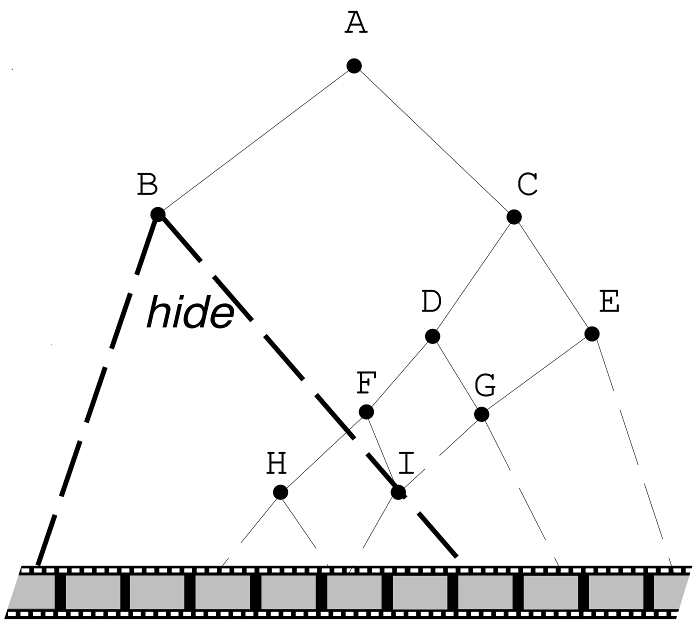

Figure 3-11: A Hierarchy of Nodes with Descriptions and *Hide-Content*

Note that the *search* operation does not examine sub-hierarchies of the components of ex-

Query	Result Set
$pred : h$	$null$
$pred : b$	$null$
$pred : h$ & $pred : c$	$null$
$pred : c$ & $pred : e$	E
$pred : e$ & $pred : b$	$null$
$pred : f$	F
$pred : f$ & $pred : e$	$null$
$pred : e$ & $pred : d$	G

Table 3.3: Example Queries on Node Hierarchy

pressions constructed by the *hide-content* operation. If any ancestor node A specifies that the descendant node D's content is hidden, then this node is excluded from examination by the *search* operation regardless of whether other nodes also include node D as their child. This behavior respects the decision of some author of a video presentation that deemed the contents of node D as unsuitable for associative access. Figure 3-11 is a schematic representation of a hierarchy of algebraic video nodes that depicts only the ancestral relationships between the nodes, and does not include the temporal composition. Assume that $pred : x$ represents a predicate that is true for node x. Assume that the descriptions specified in each node are mutually exclusive, but the node's attributes are still inherited according to ancestral relationships. Node B uses a *hide-content* in its video expression. Table 3.3 gives some example queries and their respective result sets.

A *contains* operation may examine the attribute contents of node D if it does not use node A in its video subexpression that includes D. Thus, a handle on a video node guarantees that you can still use the content-based composition features of the data model with the given node.

Temporal Queries

Temporal queries are handled using temporal predicates defined similarly to the definitions by Snodgrass [30]. He defined a *temporal predicate operator* that takes time intervals as arguments and returns a Boolean value. The three temporal predicate operators are: `precede`, `overlap` and `equal`. A *temporal predicate* is an expression containing the temporal operators. For example, if interval I_1 precedes interval I_2, then predicate I_1 `precede` I_2 is true.

These operators are adapted for the attribute based query language. The following temporal attribute field names are defined: `precede, follow, overlap` and `equal`. The user can include names of algebraic video nodes in these attributes.

- *precede*: if there is an expression where E_2 follows E_1, E_1 has attribute `precede:`E_2.
- *follow*: if there is an expression where E_2 follows E_1, E_2 has attribute `follow:`E_1.
- *overlap*: if there is an expression where E_2 overlaps E_1, E_1 has attribute `overlap:`E_2 and E_2 has attribute `overlap:`E_1.
- *equal*: if E_1 completely overlaps E_2 and E_2 completely overlaps E_1, then, E_1 has attribute `equal:`E_2 and E_2 has attribute `equal:`E_1.

For example, if $E_1 \circ E_2$, then E_1 has attribute `precede:` E_2 and E_2 has attribute `follow:`E_1. A query `follow:Anchor-talk.av` will return the node that comes after the node with the anchor talking. For simple video expressions, it is straightforward to determine the temporal predicates. Now consider the expression $A \circ ((B \circ D) \parallel (C \circ E))$. Inspection of the video expression by itself is not sufficient to determine the temporal ordering between D and E. Whether they overlap, or one precedes the other, depends on the actual durations of B and C, as well as the durations of D and E. The following four cases depict how each of the temporal predicates could be true for D with respect to E for the above video expression.

- **Case 1**: $duration(B) = 5, duration(C) = 6, duration(D) = 4, duration(E) = 7$. Then, D has the attribute $overlap : E$.
- **Case 2**: $duration(B) = 2, duration(C) = 10, duration(D) = 4, duration(E) = 7$. Then, D has the attribute $precede : E$.
- **Case 3**: $duration(B) = 10, duration(C) = 2, duration(D) = 6, duration(E) = 5$. Then, D has the attribute $follow : E$.
- **Case 4**: $duration(B) = 5, duration(C) = 5, duration(D) = 5, duration(E) = 5$. Then, D has the attribute $equal : E$.

For the above example and other simple cases, an inspection of the durations of the video subexpressions can help determine the temporal relationships between these subexpression. Note that with the *loop* operation, an expression E_1 can have all temporal predicates true with respect to some other expression E_2. The claim is true in the following compound expression, where the durations of E_1 and E_2 are both 5: $((\texttt{loop}\ E_1\ 100)\ \|\ ((\texttt{delay}\ 10)\ \circ\ E_2))$.

When the playback of the video expression can be mapped onto a timeline schedule, it is possible to determine the temporal relationships between any two subexpressions. However, whereas any timeline representation of video segments can be rewritten using video algebraic operations, not all video algebra expressions can be rewritten in the timeline metaphor. For example, the *conditional* operator requires a runtime evaluation of some test expression, and therefore the playback schedule of any video expression containing this operation can not be determined *a-priori*. In this case, the temporal relationships between two subexpression is not necessarily known. Assuming that the expression B denotes $(\texttt{conditional}\ (random(3))\ E_1\ E_2\ E_3)$ in the video expression $A \circ ((B \circ D)\ \|\ (C \circ E))$ makes it impossible to determine the temporal relationship between D and E. Finally, in the video expression $E_1 \cup E_2$, the subexpressions E_1 and E_2 do not have a direct temporal relationship, because the resulting presentation contains a non-trivial composition of the two expressions.

3.2.2 Browsing and Navigation

The browsing and navigation operations enable the user to inspect the video expression and to view the presentation as defined by the expression. The user can *playback* any expression. The *playback* operation results in the invocation of an algebraic video browser that supports some basic presentation flow control, such as `play, stop`, and `goto-beginning`. For any given expression, the user can browse and traverse the organizational hierarchy with the *get-parents* and *get-children* operations. Notice that *get-parents* of a video expression that is not a node will yield an empty result set. Finally, the user can *display* the expression associated with a video expression. As discussed above, the expression includes description, composition, and output characteristics. If the argument is not a node, the operation is simply the identity function.

3.3 Nested Stratification Permits Multiple Coexisting Hierarchical Descriptions

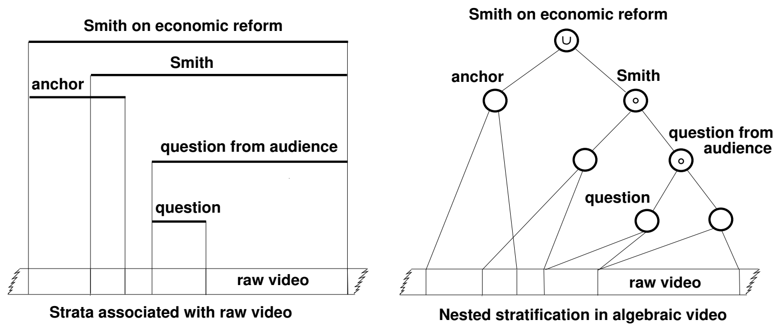

Figure 3-12: Nested stratification with algebraic video

Hierarchical relations between the algebraic video nodes allow *nested stratification*. Davenport *et al.* [4, 3] defines a *stratification* mechanism, where textual descriptions called *strata* are associated with possibly overlapping portions of a linear video stream. In the algebraic video data model, linear strata are just algebraic video nodes. To create a simple strata as in the Davenport model, a user specifies the raw video file and the sequence of relevant frames with the *create* operation.

Nodes that refer to the same video data are used to provide multiple coexisting views and annotations, and enable the user to assign multiple meanings to the same footage. Moreover, algebraic video nodes can be organized hierarchically so that their relationships are preserved and can be exploited by the user. In addition to simple stratification, the algebraic video model preserves nested relationships between strata and allows the user to explore the context in which a stratum appears. Figure 3-12 presents an example of algebraic video that utilizes *nested stratification.* As illustrated by the example, linear overlapping strata are not sufficient to model the content complexities and structure found in a video presentation. The nested algebraic video nodes preserve the structural composition of the presentation, and also allow coexisting content and structural interpretations for overlapping footage.

The nested stratification is used primarily for annotation and editing purposes, but it can also be used when browsing, searching or playing back video. An author of a video presentation can search for a particular attribute and discover the nodes that contain that attribute. The author can then explore the organization of the encompassing presentation with the nested stratification, and possibly discover other relevant information. Finally, the author will include only a selected set of the nodes and footage in the new video presentation. The *union* operator can be used for combining overlapping nodes and to guarantee that there will be no repetition of video footage during playback.

3.4 Video algebra as a programming language

Video algebra contains elements of a programmatic interface for specifying video presentations. Existing programming languages offer primitives for combining elementary data types, means of abstraction, control structures and input-output management. The algebraic video model defines a stand alone system that provides operations for a language for composing and describing video. The language of video expressions includes variables, operations, and expressions for combining these operations. It incorporates video presentations as a primitive data type, whereby all operations and all results of compound operations are video presentations. Thus, any syntactically valid combination of operations results in some video presentation (which may be empty). The algebraic video data model presents a programmatic interface that is functional in nature, and does not include the notion of state. In fact, the variables within a node are analogous to macro definitions rather than variables with state that support assignment. Also note that within the context of a node, variables can only be assigned video expressions.

Our Video algebra can be used to examine the characteristics of video as a primitive data type analogous to numbers and sets. It operates in separation from the syntax or semantics of any particular programming language, and concentrates on the video presentation data type. The model is targeted for users who wish to create video presentation and who do not necessarily possess programming skills. The model must therefore be simple enough to use for those who only understand the characteristics of video. In addition, the model can be used as a basis for a direct manipulation graphical video editor.

The rest of this section discusses the merits of incorporating video algebra into existing programming languages, and illustrates how means of abstraction can be added to video algebra

Video Templates	
template	`template` *formal-parameters tmpl-body* creates a *video template* where *tmpl-body* is a video-expression that can have variables specified in the *formal-parameters*.
apply	`apply` *video-template arg-list* defines a video presentation which is the result of evaluating the body of the *video-template* where the formal-parameters are bound to the corresponding values in the *arg-list*. These values can only be video expressions.

Table 3.4: Video Templates

using *video templates*.

3.4.1 Video Algebra as an Extension to Existing Programming Languages

The algebraic video data model described in this thesis can serve as the foundation for an extension of existing programming languages (such as TCL or Scheme), where the algebraic video operations are added as new primitive functions that manipulate the video data type. In terms of expressive power, there are definite advantages of including algebraic video as an extension library within a production level programming language. The author of video presentation, who must now possess programming skills, will be able to use variables, assignments and procedural abstractions. Programming languages support a multitude of other data types, such as numbers, strings, and possibly time values, that could be included in video expressions and aid in the construction of complex video presentations. For instance, it becomes possible to create a presentation with an undetermined and unlimited number of display windows and raw video segments that would depend on some runtime evaluation of arbitrary conditions.

3.4.2 Abstraction Using Video Templates

The model as presented lacks a good means of abstraction and reuse over existing video presentations. The video algebra can be extended to include a primitive form of first-class *video templates* using the substitution model of procedure application as defined in [1]. Table 3.4 describes the extensions for incorporating templates into the video algebra. Templates are used for

ElevenNewsTemplate.av

```
(template (intro date anchor headline followup)
  (($intro
    || (window $date (0,0) - (.2,.2) 10)
    || (window (limit $headline 0:10) (.6,0) - (1,.4) 20))
  ∘ $anchor ∘ $headline ∘ ($followup − $headline)))
```

Figure 3-13: Eleven O'clock News Template Node

```
(apply  ElevenNewsTemplate.av
  GenericIntro.rv   Date-May17.rv   AnchorMay17.av
  EconomicRecovery.av   May17StockMarketRebound.av)
```

Figure 3-14: Customized Newscast for May 17th

abstraction in a manner analogous to macros, but can also be passed around to other templates and be returned as a result, i.e. they are first-class. Since the only fundamental data type in algebraic video is the video presentation, templates only accept video expressions as arguments. With these extensions, the language provides variables, expressions and "procedure-like" templates that operate on expressions in the language. Note that these templates can be named within the context of a node, or a template can have a global name if it it a video node.

The *tmpl-body* may use names specified in the *formal parameters* of a template that refer to the corresponding arguments of the video template. The *tmpl-body* is a video expression that results in a presentation upon replacement of the formal parameters of the video template with the arguments supplied during the template application. Thus, the result of applying a video template to an argument list is the body of the template where each formal parameter is replaced by the corresponding argument. This application yields a video presentation. Note that a video template by itself is a *NULL* video presentation, because the template body has not been applied to yield a result. Also note that it is an error to apply a non video-template to an argument list. The model does not allow for mutable data as part of the template definition.

Figures 3-13 and 3-14, illustrate the usage of templates in simplifying the task of creating the late night news presentation. Given the template node that captures the structure of the news presentation, one can easily create a new video presentation for that day's newscast. The video author merely has to *apply* the corresponding subpresentations to the template to arrive at the finalized multi-window presentation.

Chapter 4

Implementation

The *Algebraic Video System* is a prototype implementation of the algebraic video data model and its associated operations. The prototype system facilitates experimentation with the algebraic video data model, and helps evaluate the feasibility of the design goals in the construction of a real production system. The system provides support for composition and content-based access to algebraic video. The creation of video expressions involves the specification and combination of raw video segments. Video expressions also serve as repositories for attribute information extracted from the video segments. In the prototype, the units of storage and indexing are the algebraic video nodes. These nodes are textually represented by human-readable, semi-structured, algebraic video files. The system has a graphical interface for managing a collection of raw video segments and algebraic video nodes, and includes query and browsing tools. Figure 4-1 presents the architecture of the implementation. This chapter describes the system level components that support query processing and video playback. Chapter 5 introduces the user interface components of the Algebraic Video System.

The algebraic video system provides the following functions:

- acquisition of video data from external sources (such as TV broadcasts, or other video collections),
- parsing the raw unstructured video to algebraic video files,

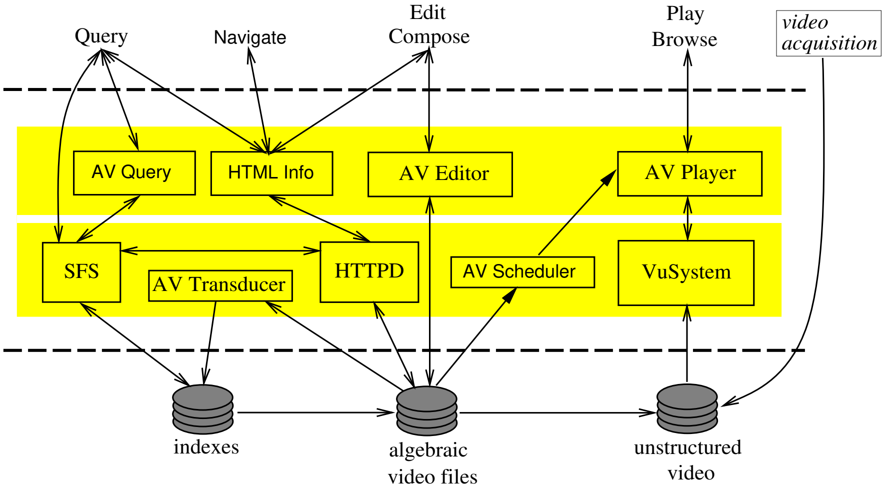

Figure 4-1: Algebraic Video System Implementation

- indexing of algebraic video nodes,
- content-based access to the data,
- playback and browsing of the video expressions,
- user composition, reuse and editing of more complex video expressions.

The implementation is built on top of three existing subsystems: the VuSystem [34], the Semantic File System (SFS) [15], and the World-Wide-Web's HTTPD [7]. The VuSystem provides an environment for recording, processing and playing video. A set of C++ classes manage basic functions such as synchronizing video streams, displaying in a window, and processing video streams. TCL [28] scripts control C++ classes and offer a programmable user interface that can be customized. The VuSystem is used for managing raw video data and for its support for TCL programming. The Semantic File System is used as a storage subsystem with content-based access to data for indexing and retrieving files, which represent algebraic video nodes. The HTTPD server supports the world-wide-web's graphical interface to the system that provides facilities for query, navigation, editing and composing video, and invoking the video player. The *HTML info* module that relies on the HTTPD module includes static HTML documents and a set of TCL scripts that dynamically create HTML documents in response to user interaction. Currently, the parsing of raw video to algebraic video nodes is carried out manually.

The system implements algebraic video by compiling algebraic video files into TCL scripts. These files contain semi-structured, textual specifications of the algebraic video nodes. There is one such file per node. Algebraic video files and raw video files can be named using the Semantic File System naming conventions, as well as the UNIX pathname conventions. The system recursively parses such nodes and produces TCL scripts that interpret video algebra operations. The scripts also offer an extendable, low-level programmatic interface for composing algebraic video.

The system supports indexing, searching, playback and browsing of algebraic video. All video algebra operations in Tables 3.1 and 3.2 except *delay*, *limit*, *parallel-end*, *transition* and *hide-content* have been implemented. The temporal attributes have not been implemented yet. The acquisition of video data, the associated close-captioned text, shot segmentation and parsing uses the VuSystem support. Figure 4-2 illustrates two different snapshots of the browser playing the same algebraic video file. The first snapshot contains the main window with a segment from CNN Headline News which is overlayed with a preview of a basketball game. Below the main windows are previews of two popular films. The second snapshot is taken some time later. The original main window has disappeared and the configuration of some of the windows has changed. However, the basketball preview and an excerpt from the movie "Hoffa" are still present.

The rest of this chapter discusses the implementation of the search facilities (Section 4.1) and playback of algebraic video (Section 4.2). This chapter also includes an in-depth discussion of the playback support for the *union, intersection* and *difference* operations. Chapter 5 introduces the user interface components of the Algebraic Video System.

4.1 Content-Based Access

The system extends and interfaces with SFS to provide content-based access to an algebraic video collection. The *AV Query* module, the direct user query mechanism, and the HTTPD server all use the pathname interface to invoke queries in the SFS search server. The *algebraic*

Figure 4-2: Browser Snapshots

video transducer extends SFS to manage algebraic video by providing a mechanism to extract attributes from the descriptions stored in algebraic video files. The system indexes the video files to create the correspondence between attributes and algebraic video nodes. The transducer is used in the indexing process to associate attributes and values with the algebraic video files. This indexing process allows efficient querying and retrieval of relevant video segments. Individual video nodes that overlap are indexed by the system separately.

The system automatically associates attribute information with the algebraic video files, and allows the user to manually add more attributes. To support content-based access, close-captioned text associated with the video stream is extracted if available and entered into segment descriptions as the `text` attribute. The user can add more attributes such as `title, author,` and `actor`, and organize nodes into a desired hierarchy using video algebra operators. Since the algebraic video files are stored in a human-readable, semi-structured text file format, the user can edit and create algebraic video files using any available text editor. Future work includes support for the automatic segmentation of raw video footage using the VuSystem shot detection module and *a-priori* knowledge of the video stream structure. For example, an half-hour CNN headline news presentation can be automatically segmented to the headline news, business, sports and entertainments portions by relying on CNN's fixed schedule.

The transducer uses a three-phase process to associate all the relevant attributes with the algebraic video nodes and their descendants in the video collection. As described in Section 3.2.1, a description of a video expression is implicitly inherited by its subexpressions (which may include descendant nodes). In the first phase of the indexing process, the attributes of all nodes are stored in temporary files that are associated with each node. Then, the transducer parses the video expressions of the nodes and recursively examines the descendant nodes. For each of the descendant nodes, the transducer appends the attributes of the original ancestor node to the descendant node's temporary attributes file. In the final phase, the transducer associates all the attributes found in the temporary attributes file with the corresponding algebraic video node.

To support navigation of the node hierarchy and pruning of result sets, the transducer also associates ancestral relationship attributes with the video nodes. For example, consider a node p that has children $c_0, c_1, \ldots, c_n$ and descendants $d_0, d_1, \ldots, d_m$. Then, all children $c_0, c_1, \ldots, c_n$ have the attribute `parent:p`. In addition, all descendants $d_0, d_1, \ldots, d_m$ have the attribute `descendant-of:p`. A node with attribute `parent:p` also has the attribute `descendant-of:p`. The implementation of the *get-parent* operation and the query interface relies on the property that given the above ancestral attributes, the query `parent:p` will result in $c_0, c_1, \ldots, c_n$.

The *AV Query* and *HTTPD* modules, as well as the user, communicate with SFS directly via the pathname query interface. SFS interprets a given file pathname as an attribute query. It then returns the result in a dynamically created *virtual directory* that contains the set of matching algebraic video nodes. Section 5.1 discusses in more detail the syntax and semantics of this pathname query protocol.

4.2 Playback

The *AV scheduler* and *AV player* playback facilities of the algebraic video system rely on the VuSystem for the display of digital video on the workstation. The *player* module is an extension of the VuSystem that enables multi-window video presentations and synchronizes playback between the independent windows. It extends the VuSystem with an algebraic video TCL procedure library and with an **`AvFileSource`** C++ module that controls playback of ranges within raw video files. For each algebraic video node, the *scheduler* compiles a schedule file that is used by the *player* module for playback of the video presentation as defined by the node. The scheduler compiles the files in two phases. First, the scheduler expands the node's video expression so that it does not include any node names. It performs this task by recursively parsing the hierarchy of algebraic video nodes rooted at the selected node. Then, the scheduler converts the expanded video expression into a schedule file that is used independently to control the playback of the video node. Users invoke the video player module to playback a given algebraic video file. If the time stamp of the schedule file is earlier than the time stamp of the algebraic video file, the player module executes the scheduler module to produce an up-to-date schedule file.

```
#av_schedule

createWindow win1 -horizDistance 0 -vertDistance 0 -scale 1
createWindow win2 -horizDistance 0 -vertDistance 0 -scale 1
createWindow win3 -horizDistance 0 -vertDistance 0 -scale 1
createWindow win4 -horizDistance 0 -vertDistance 0 -scale 1

createVidSegment s1 "win1 Hoffa.uv 4945241 7140730"
createVidSegment s2 "win2 ScentOfAWoman.uv 81288 1181096"
createVidSegment s3 "win3 ScoobyDoo.uv 80252 2119652"
createVidSegment s4 "win4 Alladin.uv 8394752 16788274"

audioPriorities s4 s3 s2 s1
videoPriorities s4 s3 s2 s1
initAlgebraicVideo

eval \
{ avNode conditional.av \
{ avConcat { avPlayRaw s1 } \
{ avConditional { \
puts stdout "Please select movie ending" \
puts stdout " (1 = sad, 2 = happy, 3 = Scooby-Doo):" \
set c [read stdin 2] \
return "$c" \
} { avPlayRaw s2 } \
{ avPlayRaw s3 } \
{ avPlayRaw s4 } \
} \
} \
} \
```

Figure 4-3: A TCL schedule file

The schedule file is a TCL script that consists of window and audio output declarations and a segment activation script. Figure 4-3 shows the schedule file that corresponds to the video node of Figure 3-4. The `#av_schedule` identifies this TCL script as one compiled from an algebraic video expression. The scheduler creates a display window for every video segment produced with the `create` operation. These window declarations consist of spatial position and dimension parameters. The scheduler also associates all video segments with their corresponding display windows. A video segment declaration consists of identifying the raw video file and the byte-offsets for the begin-segment and end-segment markers. These bytes offsets are calculated using the time values provided with the `create` operation. Currently, the format of these time values is $sec : \mu sec$ from the start of the raw video file. The scheduler consults an index file associated with every raw video file to determine the mapping from time values to file byte-offsets. The `audioPriorities` and `videoPriorities` are determined globally for all the video segments within a presentation using the algorithm described in Section 3.1.2. Finally, the scheduler converts the algebraic video expression into TCL syntax that includes the proper calls to routines that handle the various video algebra operations. Note that the schedule file is not intended to be human-readable.

The player module implements *playback* by dynamically interpreting the schedule files to produce streams of digital video. The streams are transmitted to the VuSystem, which then displays the digital video on the client workstation. For each video segment, the algebraic video system creates an instance of a special module (`avFileSource`) to manage the playback of the video segment into a display window. The module accepts parameters that name the raw video file and specify the begin and end points within that file. Normally, a video expression defines the scheduling constraints between the video segments included in the expression. The TCL library routines, such as `avConcat` and `avParallel`, manage the `avFileSource` modules by invoking the playback of the video segments at the appropriate times as defined by the scheduling constraints. The library routines also communicate to the modules a TCL expression to evaluate once the playback of the video segment is complete. For example, assuming S_1 and S_2 are simple video segments, for the video expression $S_1 \circ S_2$ the `avConcat` routine initiates the playback of the module responsible for S_1. In addition, `avConcat` notifies S_1's module that it must initiate the playback of S_2 once it has finished playing S_1.

Some of the playback information can be determined offline. For example, the *concatenation* and *parallel* operations on raw video segments will always result in the same video stream. However, other composition alternatives, such as *conditional* or a live video feed coupled with the *union* operation, require the system to modify the playback characteristics dynamically. The prototype implementation interprets all video expressions dynamically.

Parallel Playback

The *parallel* operation requires keeping track of termination events and invoking other operations once the playback of certain video segments has terminated. The video expression $(E_1 \parallel E_2) \circ E_3)$ defines a video presentation where E_3 is not played back until both E_1 and E_2 are finished. The algebraic video system uses simple counting semaphores to determine when all the termination events have occurred. Let s_1 be the semaphore associated with the termination of E_1 and E_2. When the playback of these two expression is initiated, the value of s_1 is initialized to 2. The algebraic video system decrements the value of s_1 upon termination of the playback of each of E_1 and E_2. The system initiates the playback of E_3 only when the value of s_1 is equal to zero. Since the VuSystem is single threaded and non-preemptive, critical sections are not necessary to access and modify the values of these semaphores.

4.2.1 Playback of Union, Intersection, Difference

The *union* and *difference* operations define video presentations where common footage is not repeated, and the *intersection* operation defines video presentations where only common footage is played back. The implementation of these operation in the algebraic video system prototype preserves the structural composition and the output characteristics of the component video expressions. For each segment in the original video arguments, the operations modify the video footage that is actually played back. Because the video expressions also have a temporal component, these operations are not associative or commutative. Note that this is only one possible interpretation of the *union*, *intersection* and *difference* operations. Other interpretations that modify window placements or that attempt to provide commutative or associative properties are also possible. This section offers a more precise definition of the above operations as implemented in the prototype of the algebraic video system. This section uses the following conventions:

Let E_a and E_b be fully expanded video expressions that do not include any node names, where E_a and E_b contain the sets of video segments $\{A_1, A_2, \ldots, A_n\}$ and $\{B_1, B_2, \ldots, B_m\}$ respectively. A video expression of the form (`create` X_i^{name} X_i^{start} X_i^{end}) is used to define each video segment X_i. The range of video footage that is associated with this video segment is denoted by $[X_i^{start}, X_i^{end}]$. A video expression E_1 is *syntactically similar* to E_2 if and only if E_2 can be transformed to E_1 by a series of substitutions. All these substitutions must replace a (`create` *seg-name* $range^{start}$ $range^{end}$) subexpression in E_2 with the subexpression ((`create` *seg-name* $range_1{}^{start}$ $range_1{}^{end}$) $\circ$ (`create` *seg-name* $range_2{}^{start}$ $range_2{}^{end}$) $\circ \ldots \circ$ (`create` *seg-name* $range_n{}^{start}$ $range_n{}^{end}$)) in E_1. Furthermore, the sequence of ranges substituted into E_1 must be a sequence of non-overlapping subranges of $[range^{start}, range^{end}]$, such that

$$\forall i, 1 \leq i \leq n \Rightarrow (range_i{}^{start} \leq range_i{}^{end}) \wedge (range_i{}^{end} < range_{i+1}{}^{start}),$$

excluding the boundary condition $range_{n+1}{}^{start}$.

The *minus* operation is used in the specification of the *union* and *difference* algebraic video operations. C *minus* A_x, where the video expression C is the concatenation of video segments ((`create` C^{name} $C_0{}^{start}$ $C_0{}^{end}$) $\circ \ldots \circ$ (`create` C^{name} $C_n{}^{start}$ $C_n{}^{end}$)), and the expression A_x is a specific video segment, is defined as:

$$C \ minus \ A_x = \begin{cases} C & \text{if } C^{name} \neq A_x{}^{name} \\ C' & \text{if } C^{name} = A_x{}^{name} \end{cases}$$

where

$$\begin{aligned} C' \ = \ & ((\texttt{create}\ C^{name}\ C_0{}^{start}\ C_0{}^{end}) \circ \ldots \circ (\texttt{create}\ C^{name}\ C_i{}^{start}\ C_i'^{end}) \circ \ldots \circ \\ & (\texttt{create}\ C^{name}\ C_j'^{start}\ C_j{}^{end}) \circ \ldots \circ (\texttt{create}\ C^{name}\ C_n{}^{start}\ C_n{}^{end})). \end{aligned}$$

In determining C_i', the following conditions must hold:

1. choose i such that $C_i{}^{start} < A_x{}^{start} \leq C_{i+1}{}^{start}$

2. $C_i'^{end} = max\{p : p < A_x{}^{start}\}$

Clearly, $C_i'^{end} \geq C_i{}^{start}$. Similarly, for C_j', the following conditions must hold:

1. choose j such that $C_{j-1}{}^{end} < A_x{}^{end} \leq C_j{}^{end}$

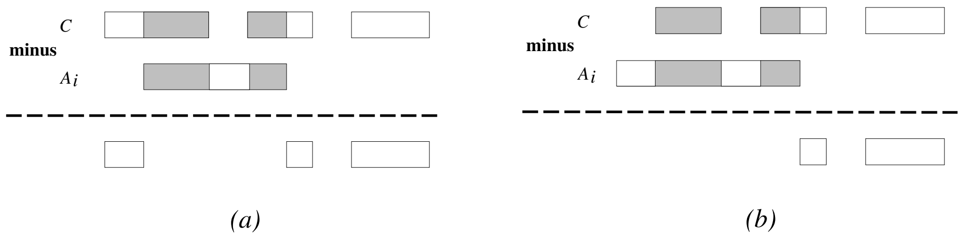

Figure 4-4: Graphical Illustration of *minus*

2. $C'_j{}^{start} = min\{p : p > A_x{}^{end}\}$

Clearly, $C_j{}^{end} \geq C'_j{}^{start}$. Note that the segments C_k, such that $i < k < j$, are not in C'.

Figure 4-4 illustrates two cases of the minus operation on video segments, assuming that the shown ranges represent time intervals within a single raw video file. The shaded areas denote the overlapping subranges. Only the non-shaded time intervals of C are played back in C *minus* A_i.

The *intersect* operation is used in the specification of the *intersection* algebraic video operation. C *intersect* $\{A_1, A_2, \ldots, A_n\}$, where the video expression C is a concatenation of video segments as above and each expression A_i is a specific video segment, is defined as:

$$C' = C \ intersect \ \{A_1, A_2, \ldots, A_n\} =$$

where

1. $C' = C'_0 \ \circ \ \ldots \ \circ \ C'_m$
2. $C'_i = (\texttt{create}\ C_i{}^{name}\ C'_{i_0}{}^{start}\ C'_{i_0}{}^{end}) \ \circ \ \ldots \ \circ \ (\texttt{create}\ C_i{}^{name}\ C'_{i_q}{}^{start}\ C'_{i_q}{}^{end})$
3. $\forall i, j,\ 0 \leq i < q, 0 \leq j < m,\ C'_{m_j}{}^{start} \leq C'_{m_j}{}^{end} < C'_{m_{j+1}}{}^{start} \leq C'_{m_{j+1}}{}^{end}$
4. $\forall i, j,\ 0 \leq i < q, 0 \leq j \leq m,\ [C'_{i_j}{}^{start}, C'_{i_j}{}^{end}] \in \ ranges \ of \ \{A_1, A_2, \ldots A_n\}$
5. let (p_0, p_1) denote the range $[p_0, p_1]$ excluding the end frames p_0 and p_1, then
   $\forall i, j,\ 0 \leq i < q, 0 \leq j \leq m-1,\ (C'_{i_j}{}^{end}, C'_{i_{j+1}}{}^{start}) \notin ranges \ of \ \{A_1, A_2, \ldots A_n\}$.

Note that any C_k can be null.

Figure 4-5 illustrates a case of the intersect operation on a set of ranges, that the shown ranges represent time intervals within a single raw video file. The shaded areas denote the

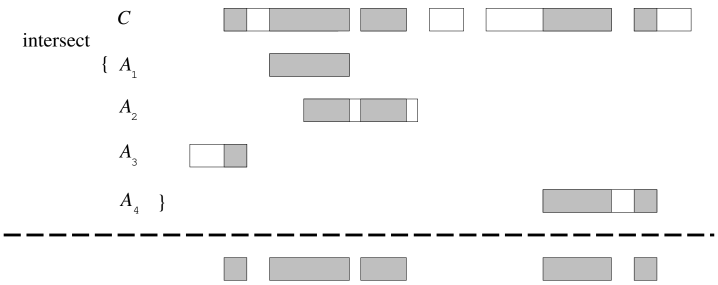

Figure 4-5: Graphical Illustration of *intersect*

overlapping subranges. Only the shaded time intervals of C are played back in C *intersect* $\{A_1, A_2, A_3, A_4\}$.

Difference

$E_a - E_b$ defines a video presentation that plays back E_c, where E_c is syntactically similar to E_a. Let the video segment C_i occur in E_c where the video segment A_i occurs in E_a. Then, C_i is defined as a concatenation of video subsegments of A_i according to the above substitution rule. Specifically, C_i is defined as A_i minus all video ranges in $\{B_1, B_2, \ldots, B_n\}$, i.e. $C_i = (\ \ldots\ ((\ C_i\ minus\ B_1)\ minus\ B_2)\ minus\ \ldots\ B_n)$.

Union

$E_a \cup E_b$ defines a video presentation where E_a is played, followed by the video presentation E_c, where E_c is syntactically similar to E_b. Let the video expression C_i occur in E_c where the video segment B_i occurs in E_b. Then, C_i is defined as a concatenation of video subsegments of B_i according to the above substitution rule. Specifically, C_i is defined as B_i minus all video segments in $\{A_1, A_2, \ldots, A_n\}$, i.e. $C_i = (\ldots((B_i\ minus\ A_1)\ minus\ A_2)\ minus\ \ldots\ A_n)$. Note that *union* can also be defined in terms of *difference*, i.e. $E_a \cup E_b = E_a \circ (E_b - E_a)$.

Intersection

$E_a \cap E_b$ defines a video presentation that plays back E_c, where E_c is syntactically similar to E_a. Let the video segment C_i occur in E_c where the video segment A_i occurs in E_a. Then, C_i is defined as a concatenation of video subsegments of A_i according to the above substitution rule. Specifically, C_i is defined as the intersection of the video segment A_i with any video segment from $\{B_1, B_2, \ldots, B_n\}$, i.e. $C_i = A_i\ intersect\ \{B_1, B_2, \ldots, B_n\}$.

The implementation of the above operations requires that the modules responsible for managing video segments be able to playback selected subranges of the original video range as defined by the segment creation operation. For example, the video segment S_1 = (`create` CNN.HeadlineNews.rv 10:00 17:00) managed by module M_1, and video segment S_2 = (`create` CNN.HeadlineNews.rv 15:00 20:00) managed by module M_2, contain some common footage. The video expression $S_1 \cup S_2$ defines a presentation that plays back the continuous range $[10:00, 20:00]$ from the CNN.HeadlineNews.rv raw video file. It therefore involves more than just scheduling the segments independently. In this case, module M_1 must notify module M_2 that it has already played back the time interval $[10:00, 17:00]$ of the CNN.HeadlineNews.rv video segment, so that M_2 does not repeat this footage.

To solve this problem, the TCL algebraic video library routines communicate with each other and with `avFileSource` modules using a message protocol. The communication is carried out using procedure invocation, where the invocation of a module or a library routine is identical for the caller. Recall that each module instance is responsible for the management of a specific video segment, whereas routines represent an algebraic video operation. For the implementation of the *union, intersection*, and *difference* operations, the communications protocol includes the following two messages from a library routine to another library routine or to an `avFileSource` module instance:

- **Message 1**: *return the video ranges you would play*
- **Message 2**: *initiate playback.*
  - *do not play video ranges* $\{range_1, range_2, \ldots, range_n\}$
  - *when finished playback, invoke the routine* p *with args* $\{a_1, a_2, \ldots, n\}$

The following simplified communications pattern illustrates the interactions that take place during the playback of the above example ($S_1 \cup S_2$):

- `avUnion` asks S_1 for result of message 1.
- `avUnion` sends message 2 to S_1. It notifies S_1 that S_1 must invoke S_2 and tell S_2 not to play $\ll$result of message 1 to $S_1 \gg$.
- After playback is complete, S_1 sends message 2 to S_2 and tells it not to play any video range in $\ll$result of message 1 to $S_1 \gg$.

Chapter 5

User Interface

Users interact with the Algebraic Video System by creating and editing algebraic video nodes, invoking queries, navigating through a node hierarchy, and playing back the algebraic video nodes. The system can be accessed directly through the Semantic File System interface, through a TCL based graphical query interface, through the WWW [8] and Mosaic [17] using the *HTML info* module, and by invoking the *AV player*. This chapter introduces the user interface components of the algebraic video system. Section 5.1 discusses the SFS interface for content-based access, Section 5.2 introduces the *AV Query* graphical interface for content-based access, Section 5.3 discusses the *HTML info* interface that supports search, navigation, editing and composing, and finally Section 5.4 introduces the algebraic video player.

5.1 SFS Interface for Content-Bases Access

A user can discover relevant nodes by invoking queries within an algebraic video node collection directly through the Semantic File System interface. A user selects a *virtual directory* pathname that describes desired attributes of the relevant video nodes. This query pathname consists of an SFS *server name* that identifies the video collection, concatenated with a list of attribute-value pairs. A virtual directory that is created dynamically in response to the user query contains the set of video nodes that match the conjunction of the query terms. For example, consider the following query and response:

```
% ls /sfs/av/text:/smith/title:/economy
CNN.HeadlineNews.5.17.av
EconomicLectureSeries.10.19.av
WorldNewsReport-SmithEconomicForecast.av
```

In this example, the user learns that there are three video nodes in the `/sfs/av` video collection that match the query "`text:smith & title:economy`".

The Semantic File System also supports a form of *query completion*. Through the *virtual directory* interaction, a user can determine which video node attribute values are present in a specific collection. For example:

```
% ls /sfs/av/title:
2            clinton      headline       news         world
alladin      cnn          health         reform       young
and          disney       healthcare     walt
care         from         hillary        wayne's

% ls /sfs/av/title:/world
waynes_world_2.av
```

In this example, the user requests a list of all the `title` attribute values that are present in this video collection. The user then queries the system to discover the algebraic video nodes that contain the attribute `title:world`.

5.2 AV Graphical Query Interface

The algebraic video system includes a graphical interface that enables users to formulate queries and perform query completion without directly interacting with the SFS pathname interface. Figure 5-1 illustrates a query based interface that allows searching, browsing, and playback of algebraic video nodes. The user creates a query by combining one or several attribute-value *query terms*. The **Search** button is used to invoke the specified query. The system responds by

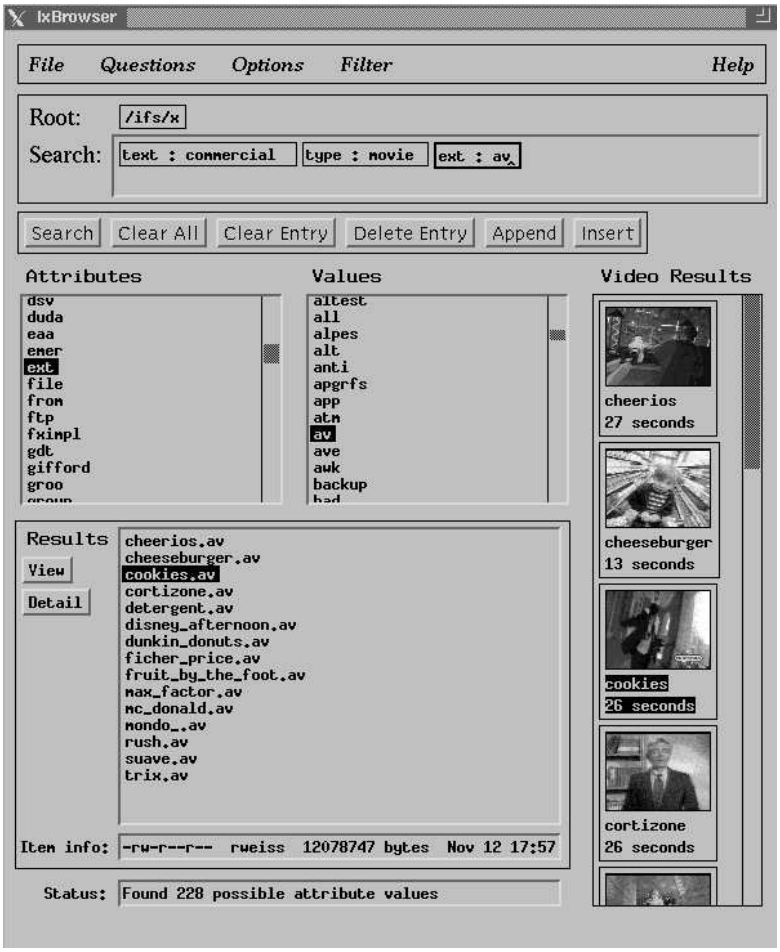

Figure 5-1: Algebraic Video Query Interface

enumerating all the algebraic video nodes that match the query in the **Results** list. The user can then select to `View` any of the nodes in the result set.

This graphical user interface supports several operations that facilitate the query formulation process. The attribute names found in the selected video collection are always enumerated in the **Attributes** scrollable list. Once a name is selected from the **Attributes** list, the query interface enumerates the possible attribute values that coincide with this name in the **Values** scrollable list. Other actions that are provided to aid in the manipulation of the query terms include: `Clear All`, `Clear Entry`, `Delete Entry`, `Append` and `Insert`. The **Video Results** list provides an iconic view of the first frames of video footage that matches the specified query. This interface currently only supports this iconic view for raw video segments.

5.3 HTML Info Interface for Editing, Navigation, and Query

The *HTML Info* module provide access facilities for creating and editing algebraic video nodes, for navigating and browsing a video collection hierarchy, and for performing searches with a video collection. The Mosaic interface to the algebraic video system consists of a set of related HTML [7] documents that include *forms* to support interactive access. Some of these documents, such as the ones that enumerate the result set for a particular query, are created dynamically to reflect the interaction with the user.

5.3.1 Create and Edit Algebraic Video Nodes

As described in chapter 4, algebraic video nodes are represented by semi-structured files in the algebraic video system. The user can create and edit algebraic video files using any text editor. To facilitate this process, the system also provides an HTML page accessible through Mosaic as shown in Figure 5-2. This forms-based HTML document enables users to create and edit algebraic video files. The user creates new files by typing in a new node name and then saving the node. The multi-line text entry field is used to create new video node expressions and edit existing ones. The *Undo* action reverts the text entry field and the node name to their original content. In addition, the following actions are available by selecting the corresponding menu option and pressing the *Perform* button:

- **Save Node**: save the current algebraic video node.
- **Preview This Node**: preview the video presentation defined by the current node, but do not save this node.
- **Clear**: erase the current video expression.
- **Edit AV Node**: edit the selected algebraic video file from the available video nodes list.

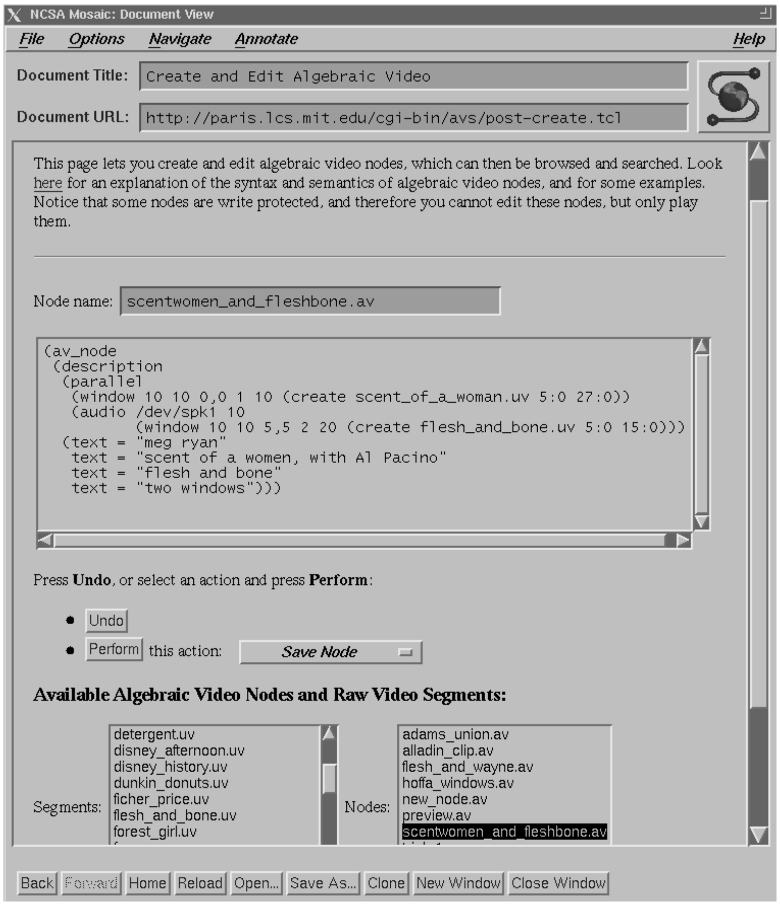

Figure 5-2: Create and Edit Algebraic Video Files

- **Play AV Node**: play the algebraic video file selected from the available video nodes list without affecting the current video expression. This option enables the user to browse and preview an algebraic video file that can then be included in the current video expression.

- **Play Video Segment**: play the raw video file selected from the available video segments list. This option enables the user to browse the existing raw video collection, and then create new algebraic video expressions from specified ranges within a raw video segment.

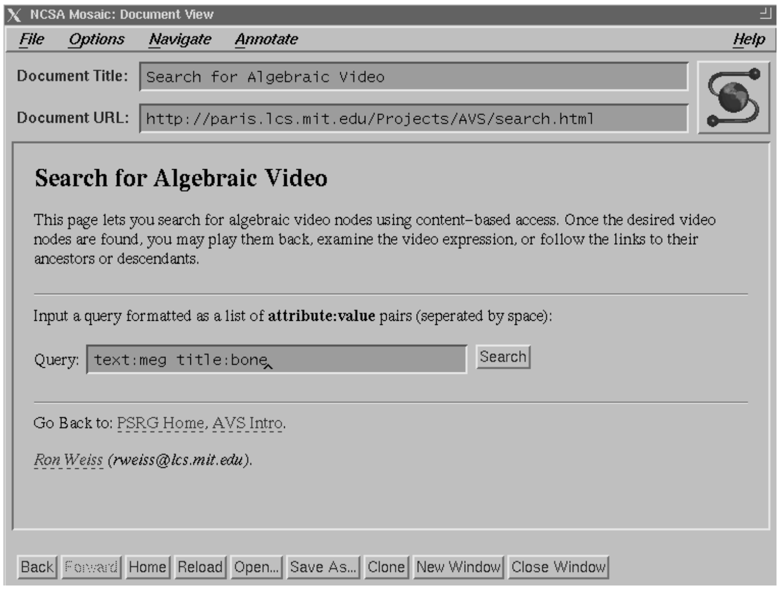

Figure 5-3: Mosaic Query Interface

5.3.2 Query Access using the HTML Info module

In addition to the direct SFS interface and the *AV Query* interface module, the algebraic video system provides a simpler query interface through Mosaic as shown in Figure 5-3. The user enters a query that consists of attribute-value pairs and submits that query to the system by pressing the `Search` button. The system responds by creating another HTML page that contains a list of all video nodes that match the specified query. The user may then playback any of these nodes, or return to the search page and formulate another query. This interface does not currently support query completion as provided by the *AV Query* interface described above. An advantage of this interface is that nodes whose ancestors are also in the result set are removed from the final result set.

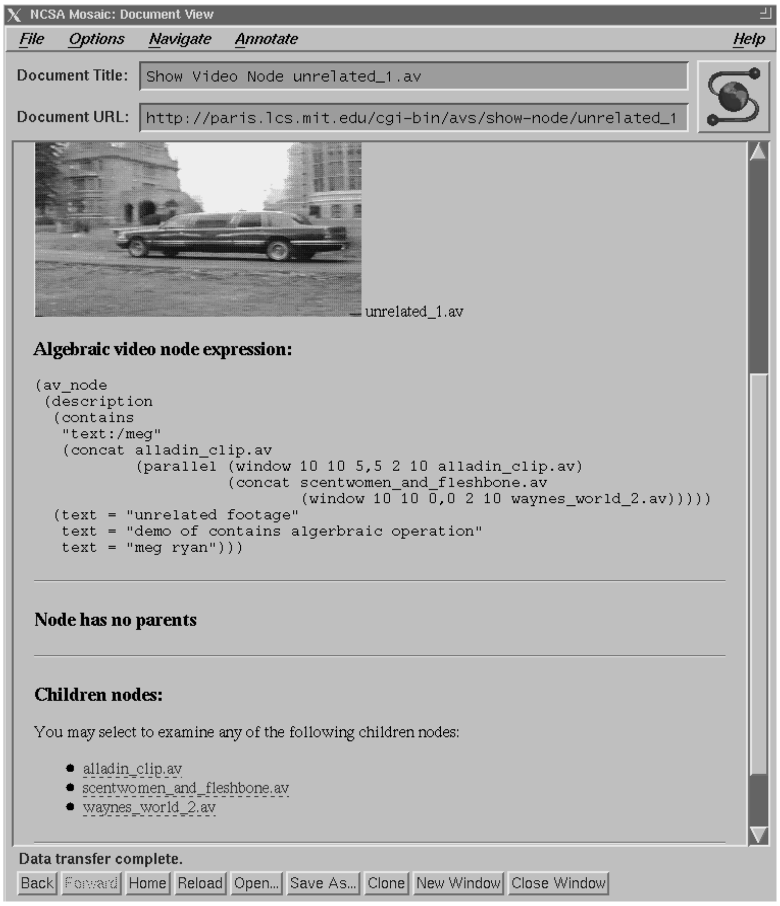

Figure 5-4: Navigation of the Video Collection using Mosaic

5.3.3 Navigation using Video Node Ancestral Relationships

Users can navigate the video collection hierarchy by following links between algebraic video nodes that have ancestral relationships with other video nodes in their collection. A node p is a *parent* of nodes $c_0, c_1, \ldots, c_n$ if the video expression of node p includes the names of nodes $c_0, c_1, \ldots, c_n$. Likewise, nodes $c_0, c_1, \ldots, c_n$ are *children* of node p. A user navigates through a video collection by following ancestral links from one video node to another. The operations *get-parents* and *get-children* are available through the file system interface. Given a node a, invoking "**`get-parents`** a" from the command line results in a list of the names of the parent nodes of a in the video collection. Similarly, invoking "`get-children` a" from the command line results in a list of the names of the children nodes of a in the video collection.

The system also supports navigation using the Mosaic interface of the *HTML info* module. Figure 5-4 illustrates the HTML page as seen by the user once a node has been selected. This selection is achieved either through query access or by browsing the list of nodes in the video collection. This page displays a snapshot of the playback of the algebraic video node, provides the video expression of the node, and lists other nodes with ancestral relationships. The user visits related algebraic video nodes by selecting either a parent or a child of the current node. Figure 5-4 shows that the node *unrelated.av* has no parents in the video collection, but does have three children.

5.4 Algebraic Video Player

The algebraic video system provides a player (shown in Figure 5-5) that allows users to view the video presentation defined by the selected video node. Users control the presentation's temporal flow by initiating the playback, stopping the playback, rewinding the video presentation to the beginning, and modifying the playback speed. In addition, users may resize the root display window, which causes the system to proportionally resize all the subwindows.

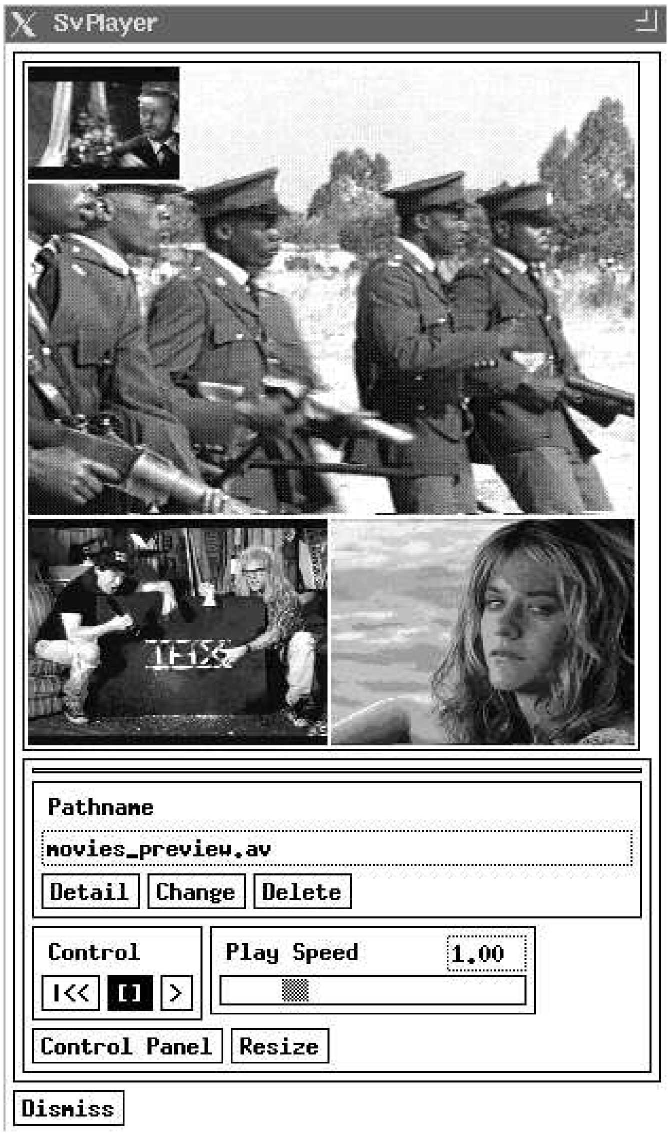

Figure 5-5: Algebraic Video Node Player

Chapter 6

Experience and Conclusions

This chapter reports experience with the content-based access components of the system and the performance of video playback (Section 6.1), evaluates the results of the thesis (Section 6.2), describes possible areas for further research (Section 6.3) and summarizes the contribution of this thesis (Section 6.4).

6.1 Experimental Results and Performance

The algebraic video system prototype provides rapid attribute-based access to the video collection, allows browsing video result sets, and supports playback of algebraic video presentations. This section includes representative experiments with the prototype system that help gain insight into the algebraic video data model, its support for content-based access, and the performance of the prototype system. For the experimentations, a collection of video segments that included TV broadcast news, commercials, and movie trailers were acquired and indexed. The experience reported in this chapter is from running the query client and the video player on a SparcStation 10, the query and file server on an SGI PowerSeries 4D/320S, and the HTTP server on a different SparcStation 10. The three machines use an ethernet local area network for communication.

6.1.1 Experience with Content Based Access

The prototype system delivers reasonable performance for query access to the video collection. Users can submit queries and browse video result sets with the TCL-based algebraic video query interface described in Chapter 5. For a result set of twenty-five video segments, the elapsed time between query invocation by the user and system response is less than five seconds. The system response includes enumerating algebraic video segments that match the query, as well as displaying the first frame of the matching raw video segments. Once the user selects a video node to play, typically three seconds elapse until the browser begins to play the video stream. Section 6.1.2 contains more detailed statistics for this elapsed time, which also depends on the number of independent windows in the video presentation.

The system also offers an HTTP interface that allows content based access to the video collection, browsing of video result sets, navigation through the video node hierarchy, and playback of relevant video presentations. The prototype configuration includes a Mosaic HTTP client and a video player on a SparcStation 10, and an HTTP server on a different SparcStation 10 that is also a client of the SGI query and file server. The response time for this configuration over an ethernet local area network is similar to the TCL based query interface. For a result set of twenty-five video segments, the elapsed time between query invocation by the user and system response is usually between two and five seconds. The system response includes dynamically creating an HTML document that contains links to the video nodes in the result set. As expected, network and server load are two important factors that affect the system response time.

6.1.2 Playback Performance

This section illustrates the bandwidth requirements for video playback, and reports performance measurements for the algebraic video player. The experiments reveal that the system delivers acceptable video playback performance for a small number of concurrent windows, but improvements in hardware are necessary before the player is able to deliver reasonable performance for a large number of windows.

Raw Bandwidth Requirements

The transfer of raw, uncompressed digital video requires substantial network and I/O bandwidth. The following table illustrates the bandwidth requirements for playing back raw video. It measures the throughput for several different frame rates (in terms of KBytes per second) that is required to support one video stream with frames of 320×240 pixels (VHS quality), at 8 bits and 24 bits per pixel,

depth \ rate	5 *fps*	10 *fps*	15 *fps*	30 *fps*
8 bits	384	768	1152	2304
24 bits	1152	2304	3456	6912

These numbers can help deduce the limitations on achievable frame rates, given available network and disk I/O bandwidth. Current ethernet local area networks operate at a peak of 1.25 Mbytes/sec. When taking into account the network overhead, distributed video application are likely to sustain playback of around only 10 *fps* of 8-bit raw video in lightly loaded ethernet local area networks. Such applications must therefore employ video compression techniques, such as JPEG [21] and MPEG [22] to overcome the bandwidth limitations of current networks.

Playback Frame Rates

For evaluating the playback performance of a multi-window concurrent algebraic video presentation, playback speed is measured as the frame rate achieved by each of the concurrently playing subwindows. Figure 6-1 illustrates the playback speed of several multi-window video presentations, measured in frames per second per window versus the number of windows. The video presentations in this experiment play back several concurrent subwindows of the same 30 second, 296 frame, uncompressed video footage. For each measurement, the windows are arranged in a square grid with a total video display area of approximately 640×480 pixels for all the windows. For example, the video presentation that includes 4 windows arranges them in a 2×2 grid, where each window has a display size of 320×240 pixels. Some configurations, such as 9, 25 and 49 windows, have a total display size of slightly less than 640×480 because the current prototype only allows certain scales for resizing windows. Note that the video footage

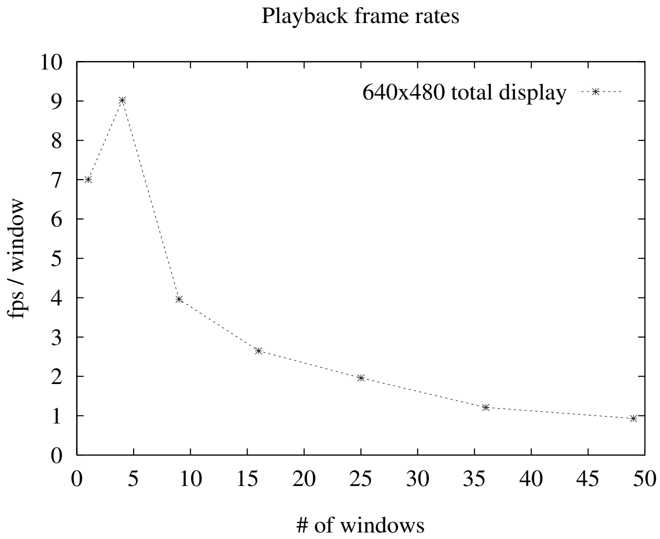

Figure 6-1: Playback speed versus number of windows

is stored in a 320×240 pixel resolution, with 8 bits per pixel. The video player on the client machine scales the video source resolution to fit within the video window.

Because current ethernet networks cannot transmit more than approximately 10 *fps* of raw uncompressed video, in this experiment each subwindow plays the same video segment. Users of the system will probably want to view several concurrent windows that contain different footage, but the current network configuration limits the total frame rate for all concurrent windows to 10 *fps*. Because the experiment considers a configuration where all the windows are playing the same video footage, as a benefit of client side caching the video player application needs to retrieve the same video frame from the file server only once for each subwindow. Therefore, when playing the same video footage, it is possible to sustain 50 windows at 1 *fps* each, for a total of 50 frames per second, while it is possible to sustain 4 windows at 9 *fps*, for a total of only 36 frames per second. Up to some reasonable limit, the window display is not a bottleneck in achieving higher frame rates. For example, a similar video presentation of 50 windows, where each window was 4 times the original area (i.e. a total display of 1280x720) achieved almost the exact frame rate as the 50 window video presentation with a total display of 640x480.

The author's subjective qualitative assessments of frame rates is that a rate of 10 frames per second is sufficient for viewing video, 5 frames per second is acceptable only for a short duration, and rates that are significantly lower are not pleasant to watch. On the existing hardware platform, the current prototype fails to achieve satisfactory viewing performance when more than 10 windows are playing concurrently. However, with the rapid advancements in computer hardware and video compression technology, acceptable frame rates are within reach.

Finally, note that the frame rate for one 640×480 window is slower than 4 windows each of 320×240 pixel resolution. Because the video source uses a resolution of 320×240, in order to display at 640×480 the system must extrapolate pixels from the original resolution. This results in a slower frame rate than a configuration where the system does not perform such pixel extrapolation.

Video Player Application Startup Time

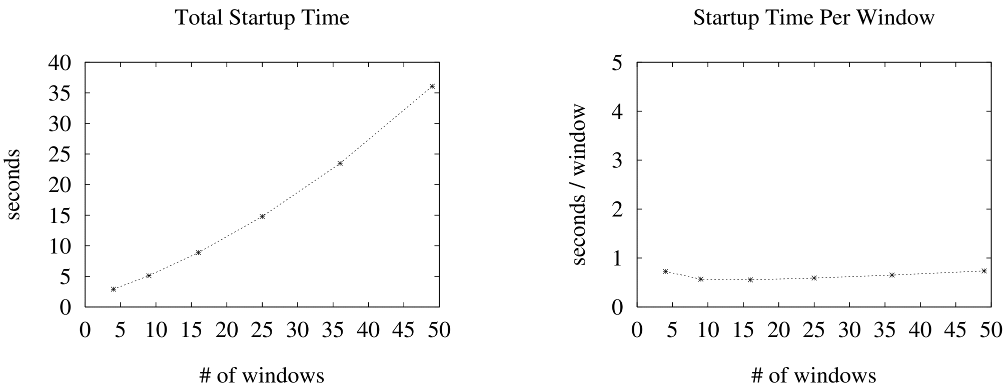

Figure 6-2: Video Player Startup Time

The startup time of the video player application increases linearly with the number of windows in the video presentation. Figure 6-2 shows the amount of time required to start up the algebraic video player application as a function of the number of windows in the presentation. The reported time measurements do not include an initial duration that varies between two and seven seconds of invoking the application and starting up the X window. The times quoted in the graph mostly include the initialization of all the VuSystem modules and the X Windows graphics context. The measurements usually vary by no more than 5-10% between each invocation of the video player. Note that the amount of time spent per window remains relatively constant as the total number of windows in the video presentation increases. Because the window creation process is time consuming, dynamically creating the windows during playback on an as-needed basis will result in noticeable delays during transitions between video streams in different windows. Therefore, the prototype implementation creates all the windows that are used in the video presentation by instantiating all the appropriate modules in the VuSystem before the user is allowed to interact with the player.

Compiling a Schedule File

For playing back an algebraic video node, the system compiles a TCL schedule file from the algebraic video file specification. The *scheduler* module, written using Lex and Yacc, is in charge of this task. The schedule compilation process is not very time consuming, even for reasonably large files. For example, the system compiles an algebraic video file of size 2.8 KBytes, that contains declarations for 50 windows as well as other algebraic operations, in approximately 1.3 seconds processing time (1.9 seconds real time). These numbers do not change noticeably for the same 2.8KByte video expression when stored in 50 different files, rather than one file. In this case, the processing time is approximately 1.5 seconds (2.1 seconds real time).

6.2 Evaluation

As stated in Chapter 1, the algebraic video model and a system that implements this model must meet the following criteria:

1. **Complete, Integrated Framework**: The model must support the fundamental access methods to video in an integrated fashion.

2. **Ease of use**: The data model and the system that implements the data model must be easy to use.

3. **Performance**: Query and video playback should provide satisfactory performance.

The rest of this section first considers some limitations of the algebraic video data model and the prototype implementation, and then evaluates how algebraic video meets the criteria for success and points out the advantages of this model.

6.2.1 Limitations

Prototype Limitations

The current system prototype suffers from two limitations. First, the user interface to the implementation consists of several components, namely the direct SFS interface, *AV Query* module, the *HTML info*, and the video player. While the combination of these components supports all the facilities offered by the system, no one such module is complete. The *HTML info* module is the most comprehensive, but it lacks the query completion capability and because of network access security considerations, the module does not allow users to store video nodes in arbitrary locations on the file system.

Second, to overcome the frame rate limitations imposed by current networks and I/O bandwidths, the prototype should incorporate video compression technology. At least in the near future, as network bandwidth increases, user demands on acceptable frame rates and the number of concurrent windows will also increase. However, since the human eye perceives 30 *fps* as continuous motion, at some point in the future the bandwidth requirements will mostly depend on the number of concurrent windows. Once network and workstation I/O bandwidth are sufficiently large, video compression will no longer be necessary for transmission purposes (but may still be required to reduce storage).

No Random Access

The current video algebra framework does not allow random access to a video presentation. It is inherently impossible to provide fully generalized random access because some of video operations, such as *conditional* and *contains*, introduce non-determinism in resolving which video footage needs to be played out. Therefore, a user cannot always "jump to" a specified location in the video presentation, such as 30 seconds into the presentation, without first resolving the conditionals in the video expression. In a situation where random access is considered more important than the conditionals, a practical solution would be to generate another video presentation, where conditionals have been resolved, and use this new presentation instead. This process can also generate an index file that contains a mapping between video presentation temporal indices and the window display configuration of the presentation. When a user wishes to access some particular time index, the system consults the index file, and generates the corresponding video display.

A digital video system should also provide random access based on the logical structure of the video presentation. For instance, based on the logical description of the video presentation, the user may select to view a video presentation starting from a particular scene. Again, because of the non-determinism inherent in algebraic video, any non-determinism must first be resolved. The system can employ a similar solution to the one proposed above. Note that the current navigation facilities allow the user to have a primitive form of this type of random access. A user can independently view a component of a video presentation, but cannot “jump to” the start of a particular component within the context of an encompassing presentation.

Dynamic Evaluation Requires Improved System Performance

The non-deterministic nature of algebraic video requires that the system dynamically evaluate the video expression for displaying the appropriate video footage. This requires more system resources than a configuration where the video stream is determined *a-priory*. For instance, the system can determine the cropping of obscured portions of video windows, and not transmit the corresponding video data. In the current prototype, all video data is transferred to the player application, and the player application crops the video frames accordingly. It is also possible to prefetch data if the video player client notifies the server of what data will be required in the near future. To alleviate some of these problems, the system should attempt to precompile portions of the video presentation, such that standard performance enhancement techniques can be employed.

Progamming Language Interface is Needed

As discussed in Section 3.4, extending an existing programming languages to include algebraic video will benefit the data model. Users that create video presentation will be able to employ means of abstractions, flow control, and utilize variables with state for more expressive power. Also, while algebraic video is well-suited for users, it does not currently address interaction with other applications. The model needs to be extended so that applications may interface to algebraic video and be able to integrate facilities such as search and playback into their own.

No Specified Framework for Automatically Associating Content with Video

The current model of algebraic video does not specify a framework for automatically associating content with video segments. However, algebraic video can incorporate systems that perform this task as such systems materialize. While this is still an active area of research, it is unclear whether associating content with video can ever be fully automated.

6.2.2 Key Results

The algebraic video data model successfully meets the criteria goals mentioned above. Section 6.1 illustrated that the system offers acceptable performance, both in query access and for video playback. The prototype system helps demonstrate that the goals set forth in this thesis have been met. The following discussion evaluates the key results of algebraic video.

Complete, Integrated Framework

The motivating examples in the introductory chapter reveal that users who access and manage digital video must be able to search, create, and playback video presentations. Algebraic video offers a complete, integrated framework for accessing and managing digital video through the *search*, *navigate*, *compose* and *playback* operations. Users find relevant video footage and presentations using the query and navigation mechanisms. Users can then browse and view the result sets using the video player. Finally, they can compose new video presentation using the results of the searches.

Video Algebra is a Powerful Metaphor for Composition

The video algebra presents a powerful metaphor for composing video presentation by allowing the user to temporally and spatially combine video segments. Users also describe the logical structure of the video by specifying the hierarchical relations between the video nodes. In addition, any video presentation that is expressed using the timeline metaphor can be expressed using the video algebra operations. Users employ *Video algebra* to compose and edit many diverse types of video presentations. The examples of algebraic video node in Chapter 3 illustrate a variety of video presentations that can be constructed using the algebraic operations.

Assigning Content Attributes is Orthogonal in the Video Algebra

The support of content-based access is an integral factor in the design of the video algebra. The *description* algebraic video operation allows content attributes to be associated with video presentations in an orthogonal manner. To fully benefit from content-based access, the association of attributes with portions of a video presentation must be incorporated as part of creating a presentation. The user is able to assign attributes to any video subexpression used in defining the presentation, and therefore to any logical entity in the presentation.

The Model Supports Multiple Coexisting Views that Preserve Hierarchical Relations

The ability to support multiple coexisting views of the same video footage is essential in describing video. Multiple coexisting views are associated with the same video footage using nested stratification. A video data model must accommodate the coexistence of many different, overlapping interpretations of the same video footage. Video cannot be sufficiently described by a simple, linear textual annotation, nor can it be appropriately described by multiple, unrelated and possibly overlapping linear annotations. The content annotation of video must preserve the logical structuring of the video presentation, and possibly supply additional logical structuring as new interpretations of the same video footage are added. The hierarchical relations between the nodes signify this logical structuring. The attribute association must be orthogonal to creating presentations, and therefore it is incorporated into the algebra. Moreover, multiple interpretations of the same footage can coexist by using the video algebra operations and the algebraic video data model.

Users can Efficiently Find Video Footage of Interest

Users can efficiently find video presentations of interest using the query and navigation facilities. The process of finding relevant video footage initially involves specifying the desired attributes using the search mechanism of algebraic video. This is accomplished either using the *AV Query* interface module or with the *HTML info* module. The *AV Query* module can further aid in formulating and selecting queries with the query completion mechanism. The performance measurements of Section 6.1 reveal that the system response time is adequate for an interactive query session. Once the user finds relevant footage, he or she can also explore the surrounding context with the navigation mechanism. The navigation mechanism can efficiently reveal related video footage that the user may not have come across using only content-based access.

Model is Easy to Use

Interaction and experimentation with the algebraic video system reveals that the model is easy to use. The creation and editing of video presentation is simple using the intuitive video algebra operations. Furthermore, the ability to modularize a video presentation by considering and modifying the component video subexpressions separately simplifies the task of creating complex presentations because it allows the user to concentrate on a single independent aspect of the presentation. This modularization also simplifies the construction of the logical structure of the video presentation. Algebraic video enables rapid prototyping through reuse of existing component presentations. The user can easily construct a video presentation from elements of video result sets by combining them with the video algebra operations. Video composition and editing may be simplified with a graphical editor, but this editor must incorporate all the ideas of algebraic video.

Because the model requires no progamming skills, the system can be used by non-programmers. This is essential to the wide spread use of systems based on the algebraic video data model, since many video presentation authors are film makers rather than programmers. Users can easily search for video footage by enumerating the desired attributes. The system also provides query completion to aid in the query formulation process. Finally, with the *HTML info* module, navigation simply involves traversing links in the World-Wide-Web.

6.3 Future Work

The work described in this thesis can be extended in several interesting ways:

- *Internet Video Server*: The current HTTP server support, bundled with the VuSystem and the Semantic File System, can form the basis for an internet video server with content-based access to video collections. Users will be able to set up video collections that may cover a specific topic or an array of topics, and then allow other users on the internet to access their algebraic video system through the World-Wide-Web.

- *Hypermedia links*: The model can be extended to include *hypermedia links* that are instantiated in video expressions. A user may traverse these links to related video nodes that exist in different collections.
- *Object-Oriented Databases*: The system may benefit from using an OO database for algebraic video storage. Possible benefits include support for transactions and modeling of object relationships that allow richer content based access.
- *Interactive Movies*: Results from this research can be applied to implement a system for creating full featured interactive movies.
- *Automatic Video Information Filter*: A practical application of the system is a video information filter that provides users with up-to-date daily video presentations that match the users' interests. For example, the system could regularly digitize close-captioned news (such as CNN), and automatically create a short multi-window video presentation using the close-captioned text to filter the relevant video segments. This application will extend the text based information filtering capabilities of currently available systems, such as the Boston Community Information System [16].
- *Graphical Video Editor*: Algebraic video would greatly enhance the flexibility and ease of use of a direct manipulation graphical video editor. The editor should combine direct manipulation with scripting capabilities.
- *Algebraic Multimedia Documents*: The algebraic model can be extended to multimedia documents that temporally and spatially combine text, pictures, audio and video. The multimedia document algebra must also model asynchronous user actions that can affect the playback of the user presentation.
- *Video Algebra into a Programming Language*: Finally, another area of future research is incorporating algebraic video into an existing interpreted programming language, such as TCL or Scheme. Authors of video presentations will then be able to utilize abstraction, control, and variables with state in composing video presentations.

6.4 Summary

The video algebra expresses unique compositions of temporal relationships between component video expressions, defines output characteristics of the video streams, and associates content descriptions with the video. With the algebra, semantic information about video can be structured and used for content-based access. The semantically rich model of algebraic video provides an efficient means of organizing and manipulating video data by assigning logical representations to the underlying video streams and their contents. It supports nested stratification for powerful descriptions of video footage.

The algebraic video system is a prototype implementation of the algebraic video data model. It has been used to retrieve video segments and to browse a video collection. Experience suggests that algebraic video enables efficient access and management of video collections in interesting and diverse ways. From the experience thus far, the algebraic video data model appears to be an adequate abstraction for representing digital video and supporting content-based access.

The algebraic video data model offers the following contributions for accessing and managing digital video:

- It provides the fundamental functions required to deal with digital video: composition, reuse, organization, searching, and browsing.
- It models complex, nested logical structure of video using video algebra. The video algebra is a useful metaphor for expressing temporal interdependencies between video segments, as well as associating descriptions and output characteristics with video segments.
- The model allows associative access based on the content of the video, its logical structure and temporal composition.

The algebraic video system prototype demonstrates that it is feasible to implement a real production system based on the ideas of algebraic video. This thesis offers a video data model that facilitates the access and management of digital video, and thus enhances its usability. The ideas in this thesis can help ensure that as digital video becomes ubiquitous, it is implemented on top of a solid foundation that will help users extract and benefit the most from this immensely expressive form of communication.

Appendix A

Algebraic Video File Syntax

```
av_file          :
                 | declarations
                   OPEN_PAREN AV_NODE video_expression CLOSE_PAREN   ;

declarations     :
                 |  declaration declarations                         ;

declaration      :  NAME ASSIGN video_expression                     ;

video_expression : OPEN_PAREN video_expression CLOSE_PAREN
                 | CREATE NAME NUMBER COLON NUMBER NUMBER COLON NUMBER
                 | DELAY NUMBER
                 | CONCAT video_expression video_expression
                 | INTERSECT video_expression video_expression
                 | DIFFERENCE video_expression video_expression
                 | PARALLEL video_expression video_expression
                 | UNION video_expression video_expression
```

```
                 | CONDITIONAL TCL_EXP video_expression_list
                 | LOOP NUMBER video_expression
                 | LOOP FOREVER video_expression
                 | STRETCH video_expression NUMBER
                 | TRANSITION video_expression video_expression NAME NUMBER
                 | CONTAINS TXT_VALUE video_expression
                 | WINDOW NUMBER NUMBER NUMBER COMMA NUMBER NUMBER NUMBER
                   video_expression
                 | AUDIO NAME NUMBER video_expression
                 | LIMIT NUMBER video_expression
                 | DESCRIPTION video_expression content
                 | HIDE_CONTENT        video_expression
                 | NAME                                             ;

video_expression_list :
                      | video_expression video_expression_list
                        ;

content          : OPEN_PAREN attribute_list CLOSE_PAREN
                   ;

attribute_list   :
                 | attribute_element attribute_list
                   ;

attribute_element : NAME ASSIGN TXT_VALUE
                   ;
```

Bibliography

[1] Harold Abelson, Gerald Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs.* The MIT Press, McGraw-Hill Book Company, 1985.

[2] Adobe Systems Incorporated, Mountain View, CA. *Adobe Premiere User Guide*, first edition, 1991.

[3] T.G. Aguierre Smith and G. Davenport. The stratification system: A design environment for random access video. In *Proc. 3rd International Workshop on Network and Operating System Support for Digital Audio and Video.*, La Jolla, CA, November 1992.

[4] T.G. Aguierre Smith and N.C. Pincever. Parsing movies in context. In *Proc Summer 1991 Usenix Conference.*, pages 157–168, Nashville, Tennessee, June 1991.

[5] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11), November 1983.

[6] Avid Media Composer. Avid technology, inc. Metropolitan Technology Park, One Park West, Tewksbury, MA 01867, 1994.

[7] Tim Berners-Lee. Hypertext mark-up language, internet draft: Rfc 1341. http://info.cern.ch/hypertext/WWW/MarkUp/MarkUp.html, 1993.

[8] Tim Berners-Lee et al. World-wide web: The information universe. *Electronic Networking: Research, Applications, and Policy*, 1(2), 1992.

[9] A. S. Bruckman. Electronic scrapbook: Towards an intelligent home-video editing system. Master's thesis, Massachusetts Institute of Technology, September 1991.

[10] M. Davis. Media Streams: An iconic visual language for video annotation. In *Proc. IEEE Symposium on Visual Languages*, pages 196–202, Bergen, Norway, 1993.

[11] DiVA Corporation, Cambridge, MA. *DiVA VideoShop User's Guide*, 1991.

[12] David L. Drucker and Michael D. Murie. *Quicktime Handbook*. Hayden, 1992.

[13] E. Fiume, D. Tsichritzis, and L. Dami. A temporal scripting language for object-oriented animation. In *Proc. Eurographics 1987*, pages 283–294, Amsterdam, Netherlands, August 1987.

[14] S. Gibbs, C. Breiteneder, and D. Tsichritzis. Audio/Video databases: An object-oriented approach. In *Proc. 9th IEEE Int. Data Engineering Conference*, pages 381–390, 1993.

[15] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole. Semantic file systems. In *Thirteenth ACM Symposium on Operating Systems Principles*. ACM, October 1991. Available as *Operating Systems Review* Volume 25, Number 5.

[16] David K. Gifford, John M. Lucassen, and Stephen T. Berlin. An architecture for large scale information systems. In *10th Symposium on Operating System Principles*, pages 161–170. ACM, December 1985.

[17] Software Development Group. Ncsa mosaic for the x window system. National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 605 E. Springfield, Champaign IL 61820, mosaic@ncsa.uiuc.edu.

[18] R. Hamakawa and J. Rekimoto. Object composition and playback models for handling multimedia data. In *Proc. First ACM International Conference on Multimedia.*, pages 273–281, Anaheim, CA, August 1993.

[19] Lynda Hardman, Guido van Rossum, and Dick C A Bulterman. Structured multimedia authoring. In *Proc. First ACM International Conference on Multimedia.*, pages 283–289, Anaheim, CA, August 1993.

[20] J.F. Koegel et al. Hyoctane: A hytime engine for an mmis. In *Proc. First ACM International Conference on Multimedia.*, pages 129–136, Anaheim, CA, August 1993.

[21] T. Lane. Jpeg software. Independent JPEG Group, December 1992.

[22] D. Legall. Mpeg - a video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, April 1991.

[23] T.D.C Little et al. A digital on-demand video service supporting content-based queries. In *Proc. First ACM International Conference on Multimedia.*, pages 427–436, Anaheim, California, August 1993.

[24] W. E. Mackay and G. Davenport. Virtual video editing in interactive multimedia applications. *Communications of the ACM*, 32(7), July 1989.

[25] MacroMind. *Director Version 2.0*, 1990.

[26] James Matthews, Peter Gloor, and Filiia Makedon. Video scheme: A programmable video editing system for automation and media recognition. In *Proc. First ACM International Conference on Multimedia.*, pages 419–426, Anaheim, CA, August 1993.

[27] Akio Nagasaka and Yuzuru Tanaka. Automatic video indexing and full-video search for object appearances. In *Visual Database Systems, II*, pages 113–127. Elsevier Science Publishers, 1992.

[28] J.K. Ousterhout. An X11 toolkit based on the Tcl language. In *USENIX Association 1991 Winter Conference Proceedings*, pages 105–115, Dallas, TX, January 1991.

[29] Roger Price. Mheg: An introduction to the future international standard for hypermedia object interchange. In *Proc. First ACM International Conference on Multimedia.*, pages 121–128, Anaheim, CA, August 1993.

[30] R. Snodgrass. The temporal query language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[31] International Standard. Information technology hypermedia/time-based structuring language (hytime). ISO/IEC 10743, November 1992.

[32] D. Swanberg, C.F. Chu, and R. Jain. Architecture of a multimedia information system for content-based retrieval. In *Proc. 3rd International Workshop on Network and Operating System Support for Digital Audio and Video.*, La Jolla, CA, November 1992.

[33] D. Swanberg, C.F. Chu, and R. Jain. Knowledge guided parsing in video datbases. In *IS&T/SPIE's Symposium on Electronic Imaging: Science & Technology*, San Jose, CA, January 1993.

[34] D. K. Tennenhouse et al. A software-oriented approach to the design of media processing environments. In *Proc. IEEE International Conference on Multimedia Computing and Systems.*, Boston, MA, May 1994.

[35] L. Teodosio and W. Bender. Salient video stills: Content and context preserved. In *Proc. First ACM International Conference on Multimedia.*, pages 39–46, Anaheim, CA, August 1993.

[36] G. van Rossum et al. CMIFed: A presentation environment for portable hypermedia documents. In *Proc. First ACM International Conference on Multimedia.*, pages 183–188, Anaheim, CA, August 1993.