An Annotation Engine for Supporting Video Database Population[1]

M. Carrer, L. Ligresti, G. Ahanger, and T.D.C. Little

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

MCL Technical Report No. 08-15-1996

Abstract–Segmentation, video data modeling, and annotation are indispensable operations necessary for creating and populating a video database. To support such video databases, annotation data can be collected as metadata for the database and subsequently used for indexing and query evaluation. In this paper we describe the design and development of a video annotation engine, called **Vane**, intended to solve this problem as a domain-independent video annotation application.

Using the Vane tool, the annotation of raw video data is achieved through metadata collection. This process, which is performed semi-automatically, produces tailored SGML documents whose purpose is to describe information about the video content. These documents constitute the metadatabase component of the video database. The video data model which has been developed for the metadata, is as open as possible for multiple domain-specific applications. The tool is currently in use to annotate a video archive comprised of educational and news video content.

Keywords: Video annotation, video segmentation, video databases, metadata, SGML.

[1] *Multimedia Tools and Applications*, Vol. 5, No. 3, November 1997, pp. 233-258. This work is supported in part by the National Science Foundation under Grant No. IRI-9502702.

1 Introduction

In recent years interest in digital communications has substantially increased due to advances in end-user applications and the publicity of the Internet. The use of digital video is not far behind, although the infrastructure, particularly bandwidth, is not yet sufficient to compete with existing television technology. The availability of video as an accessible media leads to a new set of applications including video conferencing, video-telephony, movies-on-demand, and distance learning.

Most of these applications, however, assume the viability of constructing substantial databases of video content. A *video database system*, as defined here, is an entity which provides fast and efficient storage and retrieval of digital video across multiple application domains. For example, the common functional requirements for digital news video or movie storage and delivery are provided by such a system. In this context we expect domain specific information models to be associated with each application; however, significant common functionality will exist and be supported by a video database system.

The primary distinction of a video database by our definition is the ability to rapidly locate and retrieve video content. In contrast, an archive of video tapes is fraught with latencies that render most applications not viable. This “fast access” enables new applications of video information such as personalized news-on-demand and educational reuse of classroom instruction that would otherwise be content or feature-deficient. (For example, our experience found many students asking for a synopsis of a lecture rather than to review a two-hour video tape.) The goal of a video database application is to provide this fast access to the *information* in the video data. To accomplish this objective; however, an approach to characterizing this information must be developed. The video content (i.e., the information embedded in the video data) must be extracted from video data, stored, and managed. Techniques for this process can be static or dynamic, and manual or automated. Due to our desire to construct working systems, we focus on pragmatic alternatives. This steers us towards semi-automated techniques that apply shot boundary detection coupled with human annotation. Moreover, our motivation has been to create a tool for collecting a reusable universe of annotated, interconnected multimedia documents that can subsequently be translated, indexed, and stored in any database in any format. That is, we sought to “front-load” a multimedia database rich in video content.

There are existing solutions for providing video data collection and annotation (e.g., references [4, 10]). However, because of the complexity of the problem, most of the proposed solutions are domain-specific, being optimized to work only on specific video content such as news, instruction, or specific categories of movies. To incorporate cross-domain annotation functionality in our annotation tool, we make use of SGML (Standard Generalized Markup Language). One important property of SGML for use in representing video data is its ability to define nested structures as required for hierarchical models of video data. Moreover, SGML context rules, even if very strict, are highly customizable through definitions in the DTD (document type definition)for video. Therefore, we establish a common nested structure across domains and simultaneously consider the properties of different domains. This is achieved by associating a DTD with each domain. All of the DTDs have common elements or objects, but the attributes associated with these elements differ with each domain. The metadata, therefore, exist in an SGML-compliant format, making it easy to make post-annotation changes or enhancements to the annotated data without requiring a redefinition of the end-application data model. The collected data can also be readily translated to alternative representations (e.g., relational, object-oriented, or deductive data models), thus making it straightforward to populate a video database.

With this context, and the desire to construct a usable system, we set out to develop the Vane annotation tool. An additional requirement was to render an annotation solution that would not be bound to a particular application domain, yet could deal with, or could be tailored to, the nuances of each. The result is a system built on these concepts that can be used/tailored for any annotation and subsequent data model. The novelty in the work is the generic, flexible model and representation for capturing video content in a reusable way, and the subsequent implementation.

The remainder of the paper describes our technical solution to this problem. In Section 2 we introduce basic technologies required as a foundation for the annotation solution. In Section 3 technologies related to our work are discussed. In Section 4 we discuss the annotation tool. Section 5 describes the translation of collected annotations to a relational database format. Section 6 concludes the paper.

2 The Fundamentals of Video Annotation

When video and audio signals are brought into a digital system, they yield a format that is very different from alphanumeric data. Textual information supports concepts of alphabetical ordering, indexing, and searching on words or phrases. In contrast, video and audio data yield a format that does not provide straightforward access to its content. Ordering of the basic units for audio and video (sample and frame, respectively) is temporal. The analogous indexing or searching based on pattern matching is a much more difficult task, one that has yet to prove as complete and successful as vector-space text indexing.

Because of the lack of satisfactory techniques for extracting information from raw video or audio content, we seek alternative schemes for achieving the same goal of supporting indexing and query of content. Fig. 2 illustrates the overall process of video annotation that we describe here [1].

For simplicity, we define the following terminology used throughout the paper. Raw digital video and audio data are aggregated and called *video data*. The content of the video and audio are referred to as *video information*, but are captured as *metadata*. Thus, video information describes the content of video data stored in the video database.

Video data, at a finest grain, are represented as individual frames (a finer decomposition is not significant in the context of video content as a whole). As such, any segmentation and feature extraction applicable to still images is relevant identification of the information contained in each still. We therefore, focus our attention on the information that is obtained by the juxtaposition of stills (i.e., shots), and the associated sound track, in a video composition.

The process of isolating small units of the video (segments) is also called segmentation. This process achieves a partition of the entire video stream into collections of frames whose characteristics are common. The criterion for automated segmentation is usually consistency in compressed-domain data size. Unfortunately, this does not guarantee a perfect mapping of frame collections to video information, nor does it guarantee the finest grain of decomposition, nor overlapping sets as is associated with stratification [20].

The next step in the process of video database construction is identification of the contents of the video or *information extraction* from the raw video data. Even if the logical partition of a video stream can help in dealing with raw video data, it is still very difficult to extract information from the raw data on-the-fly. All approaches reviewed from recent literature support the idea that in order to achieve fast data retrieval through queries, video information must be extracted from the raw video data and then stored in a different, more readily usable format which constitutes the input for the query engines. Information can be extracted automatically or manually. The new format must contain references to points in the physical video data so that a particular video segment (e.g., a news item, a movie), or a part of it, can be easily retrieved from the video database.

Of course, a suitable model must exist to bin the extracted information or to otherwise represent it in a manner that will lead to fast and efficient retrieval. Subsequently, the video information can be stored in this representation in the act of video database population.

2.1 Video Segmentation

Before starting the annotation process, raw data are partitioned into elemental units called *shots*. These shots are essentially sequences of contiguous frames formed by a continuous recording process. However, because this definition yields some difficulties for edited material (shot boundaries are vague) and surveillance footage (shots correspond to interesting events), it can be modified to describe a contiguous sequence of frames whose content is common. Starting from this segmentation, shots can be used to build up a logical units called *scenes*, which are collections of contiguous shots (Fig. 1), and *sequences*, which are groups of scenes.

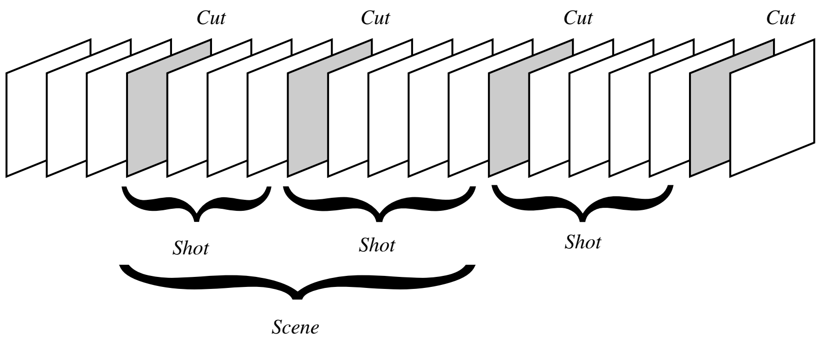


Figure 1: Segmentation of a Video Stream.

Shot detection within the raw data can be performed automatically using video segmentation techniques (e.g., [2, 9, 14, 15, 16, 24]). As a shot consists of one or more frames recorded contiguously and representing a continuous action in time and space [6], it can be completely defined by the timing of its beginning and ending points. This reduces the partitioning task to detecting shot boundaries (cuts) and identifying them by start and stop frame numbers. A variety of segmentation techniques can achieve this goal.

2.2 Information Extraction

The operation of a video database implies the management of a large quantity of raw video data. The presence of this raw data does not significantly assist in indexing and search. In contrast, video *information* assists this process. Data that characterize the information contained in video data can be called *metadata* [5, 8, 11, 12]. Although any suitable representation can be used to represent metadata, text is commonly used. In the Vane implementation we make use of SGML markup to represent video metadata.

The problem becomes one of identifying information contained in the video data and associating them with tokens (metadata). Not surprisingly, humans are quite good at extracting information from video data, whereas it is difficult to get the same performance from an automaton. In the annotation process, a viewer takes notes, albeit biased, of the content of the video stream. During this process, the annotator can be assisted by a computer to provide a more regular representation to capture domain-specific information. For example, a football announcer might use a football-specific metadata schema to capture information about goals scored. In this role, the computer, and the annotation process, provides a consistent and repeatable process for collecting metadata for the application domain.

As metadata are also intended to be applied to query formulation as well as to information capture, they should facilitate indexing and searching techniques on the video content. That is, the metadata component of the video database system can be treated as a conventional text (token)-based database system. This in turn will provide random and direct access to the raw video content. Thus it is possible to search the annotation/metadata as well as locate specific video data elements.

Three types of indexing identified by Rowe [19] are bibliographic, structural, and content-based. The first one, which reflects the conventional method to catalog movies and papers, covers the annotation of objective fields such as title, keywords, and author. Structural indexing, in contrast, is based on a logical segmentation of the video data. For each of these three elements, a data model identifies a unique set of attributes whose values must be annotated according to the elemental video content. Thanks to a hierarchical organization, structural indexing is useful for the visualization of essential information and for fast searches. Content-based indexing is used when the annotation must be focused on objects that appear in the video, irregardless if they are people or things. Supporting one or more of these indexing schemes means building different structures for metadata, and leads to a different implementation of the annotation process.

At the end of the annotation process, raw video data should be indexed to support retrieval by the multimedia system/application for which they are intended. Requested information can be identified within the metadata which will also provide references and links to the archive where the video data are stored.

2.3 Related Work

We believe that with current technologies video information must be converted to metadata to support effective database queries. Different approaches can be found in the literature for this process. The following is a summary of are related techniques.

Tonomura et al. [22] propose *Structured* video computing. Here the video stream is first analyzed and then automatically decomposed into shots. Each shot is indexed using features called *video indices* which are extracted from the video data. These indices include attributes such as camera information and representative colors. Subsequently, two different structures are built: a link structure that maintains relations between the shots and a content structure consisting of the textual descriptions related to the corresponding components of the video data.

Smith and Davenport [6, 21] propose partitioning video information into overlapping segments rather than disjoint shots. This technique facilitates multiple interpretation of segment content due to the overlap. The segments are called *strata*. With this scheme the concept of an object become more significant, as the objects, represented as strata, are independent of a strict hierarchical organization offered by a simple shot decomposition.

Tools have also been developed that consider image texture in the annotation and feature extraction process. Most rely on image processing and pattern recognition techniques. One example is *vision texture* [18] which extends the text-based concepts of sorting and similarity of alphanumeric data to the domain of images, video, and audio. With this scheme a user labels a part of an image, or of a video, and a texture model is used to propagate the label to other "visually similar" regions.

Metadata are commonly expressed using text-based tokens because of their ease of manipulation. An unusual approach is proposed by Davis [7] using icons. This approach, called *media streams* uses system-provided animated icons to describe elemental actions. Here the annotation step is performed by matching shot contents with one of the icons. Composition of icons can also achieve complex representations of shots.

Carreira et al. [4] developed a system to organize video for stored video delivery applications. The system, called the Video Broadcast Authoring Tool (VBAT) uses a simple hierarchical organization and graphic user interface that facilitates the collection of metadata. A post-processing step allows the VBAT to generate a video "table of contents" in HTML for subsequent viewing and video launch from a Web browser. The VBAT does not make use of content-based indexing techniques.

3 Related Technologies

A number of related technologies are used in the construction of our annotation engine. In this section we introduce these technologies as essential for understanding how the Vane tool is constructed. These related technologies are the SGML standard used in metadata collection, Tcl/Tk used in interface design, and the MPEG video compression standard.

3.1 SGML

In the early 1980s, the International Standards Organization (ISO) proposed the Standard Generalized Markup Language, (SGML–ISO 8879), as a means for managing and organizing information. It was designed to increase the portability documents among computers and text-processing systems.

One of the tenets of SGML is the separation of document content from its rendering. This division is achieved with text *markup*: a series of instructions, mixed and embedded in the text of the document to provide the system with necessary processing rules for interpretation during presentation. *Procedural* markup is used in most electronic publishing systems, giving a typesetter the necessary directions to fit document text on the rendered page. SGML makes use of *generic* markup, also known as *descriptive* markup, rather than focusing on how the text should appear on the page. Generic markup defines the purpose of the text in a document. Data are broken into *elements*, that represent object semantics within the overall system. Elements are organized in a strict logical structure defining a hierarchical model for the document. To understand how SGML works, the relationship between the content and the structure of a document is viewed as two layers:

- **Structure**: The DTD, or Document Type Definition, establishes a document structure. It provides a framework for the types of elements that constitute a document. It also defines the hierarchy of relationships within the elements and sets the context rules that must be followed to ensure a consistent and logical structure in the documents.
- **Content**: Tagging is used to isolate the content within the document: by using *start* and *end* tags, logical elements can be delineated. With SGML, it is possible to associate

attributes of arbitrary types to any element: user-defined or enumeration, numbers and strings. Elements can also be nested within other elements to define the organization of the document. As tags should strictly respect the set of context rules defined in the DTD, SGML parsers are required to ensure consistency and correctness of a document with respect to its DTD.

In summary, SGML enables the efficient storage and reuse of information, information sharing amongst users and applications, and information maintenance in storage. We apply SGML as a format to capture and characterize video information as metadata in the Vane tool.

3.2 Tcl/Tk

Tcl/Tk is a language developed by Ousterhout [17, 23] for rapid construction of user interfaces. It is comprised of the *Tcl* (Tool Command Language), a string-based command language and interpreter for the language, and the *Tk* (Tool Kit) which associates the X windows toolkit to Tcl. The latter defines Tcl commands that support the creation and manipulation of user interface widgets, extending the core of Tcl itself.

Tcl and Tk together also provide all functionalities of shell programs plus the ability to create graphical user interfaces (GUIs). They provide a high-level interface to a GUI toolkit, giving the possibility of implementing new interface design in a short time and hiding all the details faced by languages as C. As it is an interpreted language it does not need compilation, allowing programs to be tested and debugged quickly. For performance enhancement, libraries supporting new functions can be developed in C and pre-compiled for use with the developed application. In addition, there now exist Tcl/Tk interpreters that function within the context of Web browser software, making it possible to execute Tcl/Tk scripts in a platform-independent manner.

3.3 MPEG

A set of compression techniques developed by the Moving Pictures Experts Group (MPEG) have become a standard for the delivery of digital video for applications such as Video-on-Demand. At the moment they encompass: MPEG-1 (ISO/IEC 11172: Coding of moving pictures and associated audio - for digital storage media at up to about 1.5 Mb/s), MPEG-2

(ISO/IEC 13818) and MPEG-4 (an evolving standard). Their video and audio specifications give the semantics and syntax of encoded video and audio streams. The system specifications address the combination of audio and video into a single stream and their synchronization.

The video coding algorithms used are lossy compression schemes. The basic MPEG encoding scheme uses the prediction of motion from picture to picture, the use of Discrete Cosine Transforms, quantization, and Huffman coding to organize the redundancy in spatial directions. Pictures are encoded to yield three different frame types: *I*-frames which are encoded independently from other pictures, thus offering moderate compression but providing random access points into the compressed video data; *P*-frames whose encoding is based on the motion-compensated prediction from a past *I* or *P*-frame; and *B*-frames which are encoded by using of both past and future picture compensation [3, 13].

Each of these technologies is applied in the Vane tool, which we describe next.

4 The Video Annotation Engine (Vane)

In Section 2 we described the fundamental role of annotation in the process of organizing and building a video databases. We also highlighted the difficulties that can be encountered due to encompassing multiple application or video content domains. One of the key objectives for the Vane tool is to accommodate different application domains from a common basis, i.e., via extensibility and customization rather than by a comprehensive but inflexible solution.

To this end, we have designed the Vane tool to be a domain-independent application with the ability to support different domain-specific data models without rewriting the tool itself. This has been accomplished through the use of SGML and an exploitation of its unique characteristic of separating the context rules from the information content. This will be illustrated in Section 4.1.

In the design of Vane we incorporated a high-level semi-automatic annotation process which is intended to extract the content and semantic value of a raw video stream. Fully-automatic techniques, which are still in their early stage of experimentation and which are implemented with image pattern/image recognition algorithms were not considered reliable enough to be used in a practical manner for large video data sets (e.g., hundreds of hours/gigabytes of video data). Current technology in this domain facilitates the retrieval of pictures from image archives using color and texture analysis. Efforts also seek to identify objects and track their movements within a video stream. However, these are not appropri-

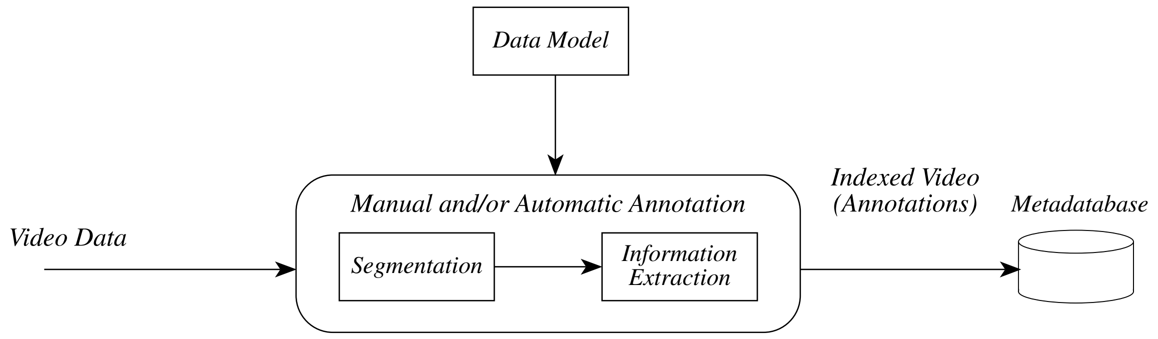


Figure 2: Data Flow in the Video Annotation Process.

ate for our objectives. None are able to make the computer sensitive to the semantic value of a video. More sophisticated techniques are required to recognize the information expressed within raw video data. As illustrated in Fig. 3, we integrated a segmentation tool into Vane so that shot detection is automatically performed when the annotation of new video data is initiated.

In our implementation, the lack of knowledge by the system is compensated by the presence of a human (the *annotator*), whose role is fundamental in the collection of video information. The Vane tool, however, has been developed to support and assist the annotator as much as possible so that annotation is expedited and leads to canonical representation of information as metadata within the application domain. It is therefore semi-automated. Finally, the Vane tool supports the playback of digital video in MPEG format with the conventional video shuttling functions.

The design of the Vane tool is decomposed into two parts dealing with metadata issues and user interface issues. The metadata issues include the definition of the format used for metadata, the specifications for its syntax and the tools necessary to deal with them. The user interface issues focus on assisting the annotator in the collection of video information as metadata. We describe these next.

4.1 Video Data Modeling

Before constructing the annotation tool, a suitable video data model was required. Our objective was to provide a flexible metadatabase able to be accessed by search engine and able to retrieve all interesting information sought (e.g., to support operations like information personalization). During the annotation process relevant content of the video must be extracted and annotated without loss of information. The optimization of the search engine to browse the whole metadatabase, or only a subset of it, leads to a system which can perform with different granularities and be able to satisfy different application needs. We model video information such that we can maintain both structural and content information. Such an approach must comprehend all types of indexes outlined by Rowe [19] (i.e., bibliographical, structural and content based) and indexes must be strictly associated to the objects they are expected to describe. Therefore, we define the metadata as *structural metadata* and *content metadata* as explained below:

Structural Metadata: Video structure includes media-specific attributes such as recording rate, compression format, and resolution; and cinematographic structure such as frames, shots, sequences, and the spatio-temporal characterization of represented objects. These are further decomposed as:

- **Media-specific metadata:** Describing implementation-specific information (e.g., video compression format, playout rate, resolution).
- **Cinematographic structure metadata:** Describing creation-specific information (e.g., title, date recorded, video format, camera motion, lighting conditions, weather; shots, scenes, sequences; object spatio-temporal information).

Structural annotations organize linear video sequences as a hierarchy of frames, shots, and scenes [6]. This decomposition constitutes the simplest video structural model.

Content Metadata: Video content metadata are concerned with objects and meaning in the video stream that appear within or across structural elements. Content metadata are further decomposed as:

- **Tangible objects:** Describing objects that appear as physical entities in the media stream (e.g., a dog, a disc).
- **Conceptual entities:** Describing events, actions, abstract objects, context, and concepts appearing in or resulting from the media stream (e.g., running, catching, tired, master).

Therefore, our annotation tool supports two types of objects: *structural objects* and *content objects*. The structural objects represent shots, scenes, sequences, format, etc. and the content objects represent the unstructured information. However, the content objects can be restricted by the structural boundaries. We include the concept of timeline so that content objects can represent strata as defined earlier [20] (Fig. 6).

In the requirements for the video data model we include object-oriented support, portability, embedded support for links and cross-references, and support for database query.

Given these requirements, the choice of SGML as the means to express and maintain metadata is natural. It is a text-based representation that is both lightweight and robust. Dealing with text is more straightforward and computationally less expensive than working directly with the time-dependent video or audio data, and intuitive concepts such as "equal to" and "similar to" are easily implementable. Moreover, SGML context rules, even if very strict, are highly customizable through the settings within a DTD. Furthermore, through its `ELEMENT` structure, object-oriented representation of concepts can be pursued. The advantage of operating with objects is that the metadata text file annotating the movie content will be easier to access and browse during database searches and data export. The support for links and cross-references is embedded in the SGML syntax, thanks to the `ID` attribute and the reference element `REF`, that can be associated with any element. All references are resolved during the post-processing achieved during data export when populating a database tailored to the application domain. The hypermedia link mechanism is useful to provide the user with other material related to the content of the video stream.

In SGML, the content of a document is separated from its structural definition. For this reason it is also possible to build a system that facilitates a dynamic document definition according to the annotator's needs and to the domain of the content to be annotated (we consider video content, which is unusual for SGML). Thus, it is possible, and we have achieved, a dynamic GUI based on a dynamic DTD. Each DTD for an application domain will have a common basis. Subsequently, tailored DTDs have augmentations reflecting the domain dependencies. In this manner we reduce the dependency of the system on the application domain allowing modifications to have little or no impact on the data model, database, or annotation engine.

The ability of SGML to nest several elements inside one another allows us to easily define a structured view of the video content. For example, a shot description can be nested inside a scene description.

It is clear that in this scenario where several types of video can be annotated the Vane tool will have to deal with different documents constituting the annotations. Each will refer

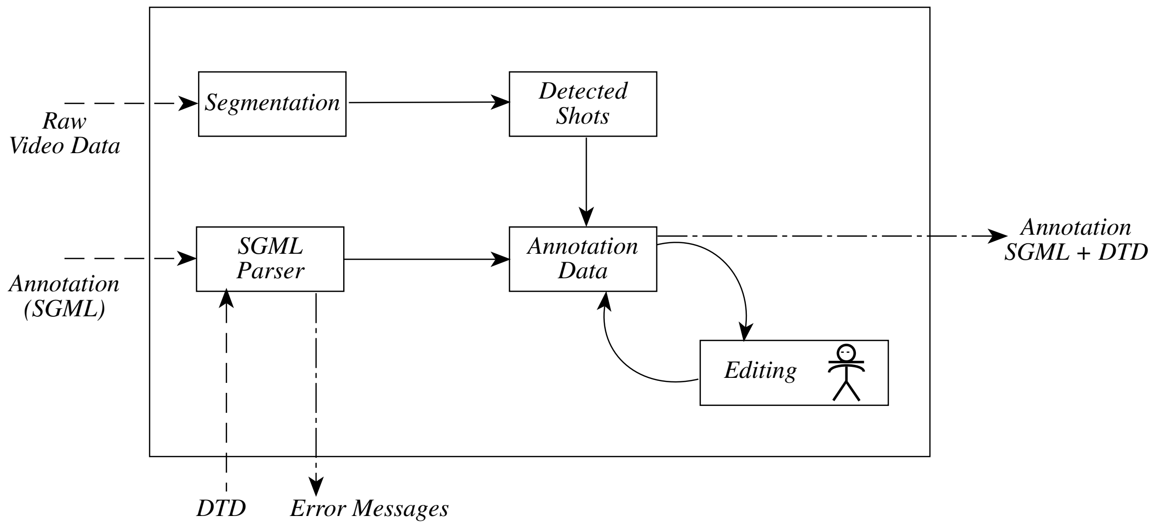


Figure 3: Data Flow in the Vane Tool.

to its own DTD. To ensure the consistency of SGML documents with the context rules specified by their own DTDs, we applied an SGML parser in the input stage of the tool. Among its functions is the notification of mismatches between the opened annotations and the corresponding DTD.

4.1.1 Model Design

Given SGML as the language to represent video content, the video content model itself must be designed following the specifications given above. The analysis of the video content follows a structural decomposition. The object representing the whole video stream can be considered the largest container in which all objects are encompassed. Its subsequent logical decomposition is performed using the basic structural decomposition as sequences, scenes, and shots. The choice of three levels of video content results in the most straightforward generic decomposition. Additional levels, in our experience, yield excessive fragmentation and do not provide significant additional information. Subsequent query operations on a finer-grain decomposition would operate at a level which is not obviously useful. Moreover, at a finer grain, there can be an excessive burden on the metadata management system with the number of components represented. All three structural layers need not be annotated. For example, it is reasonable to only annotate shots and scenes.

The step following the identification of the three levels of the video information model is the organization of identified objects. Our approach proposes a hierarchical structure in which the shots, commonly representing the smallest unit of information comprise the leaves while scenes and sequences form intermediate nodes of the hierarchy. Therefore, the decomposition of the content of the video reflects the logical organization of the objects. Scenes are groups of shots whose content is considered common and, consequently, sequences are collection of scenes (Fig. 1).

The data model also includes support for objects that encompass unstructured content data. This is achieved by annotating start and stop frames in which an object appears. A content object can belong to audio, video, graphics types or any composition thereof. For example, in the newscast domain an anchor person might reference a celebrity without showing the individual. We can annotate the individual associated with the audio medium. If desired all the annotations belonging to a particular object can be grouped together to form a stratum. Each object can be conceptually associated with any shot, scene, or segment by analyzing the time-line boundaries.

Consider the domain of educational video for this scheme. As instructional video typically follow an hierarchical organization (e.g., a lesson plan), they are organized in a hierarchical structure of topics and sub-topics and map well to the proposed model. A possible logical decomposition of educational video content can follow the following scheme: sequences represent daily class recordings, scenes represent topics within a class or lecture, and shots represent individual topics of a lesson. This logical decomposition simplifies the work of the annotation in its regular structure and facilitates the task of creating the query engine supporting the end retrieval application.

The basic purpose of Vane in this context is to provide infrastructure for collecting, annotating, and identifying structure and content in video data that will facilitate authoring and indexing at a later stage.

4.1.2 DTD Design and Syntax

We applied these aforementioned concepts in the creation of a baseline DTD with the following syntax:

```
<!ELEMENT FULLDOC   - -  (ABSTRACT? , CATEGORY? , REF* , SEQUENCE*, OBJECT*)>
<!ELEMENT SEQUENCE  - -  (ABSTRACT? , REF* , SCENE*)>
<!ELEMENT SCENE     - -  (ABSTRACT? , REF* , SHOT*)>
<!ELEMENT SHOT      - -  (ABSTRACT? , REF* , TRANSCR?)>
<!ELEMENT OBJECT    - -  (REF* , OBJECT*)>
<!ELEMENT ABSTRACT  - -  (#PCDATA & REF*)*>
<!ELEMENT TRANSCR   - -  (#PCDATA)>
<!ELEMENT REF       - O  EMPTY>
<!ELEMENT CATEGORY  - -  (EDU | NEWS | MOVIE | DOC | SPORT)>
```

The previous lines constitute the basis of the definition of an SGML document. Each object and its role is defined as follows. An object is identified by an `ELEMENT` definition which is delimited by angle brackets. The element `FULLDOC`, which represents the whole video data stream, is defined in the first line. `SEQUENCE`, `SCENE` and `SHOT` assume the obvious interpretation. All elements except `REF` have both start and stop tags. `REF`, instead, has only the start tag. The content model for the element is enclosed in parentheses. For each of the possible contained elements, an occurrence indicator is also expressed. `FULLDOC` can have at most one or possibly no `ABSTRACT - ?` occurrence indicator. As expected, a `FULLDOC` can also have one or more `SEQUENCE`s as represented by "`*`". To support stratification, content `OBJECTS`s are considered part of `FULLDOC`s and each `OBJECT` can be composed of sub-objects. In the same manner, we specify that a `SCENE` can have one or more nested `SHOT` elements.

With this set of rules expressed above, we ensure that if, by error, a shot contains a scene then an error message results when the SGML document is checked by the SGML parser. Other definitions of the content model include the reserved word `#PCDATA`, which means that the element being defined may contain any valid character data. We choose this option for the elements `ABSTRACT` and `TRANSCR` which are to be expressed by plain text. The `CATEGORY` element, in contrast, can assume only one of the values expressed in the defined list. The same syntax is used to define the possible sub-categories.

Once the object definition, or element in the SGML terminology, is completed a set of attributes linked to each of these objects must be defined. The objects `ABSTRACT`, `CATEGORY`, `REF`, `TRANSCR`, even if defined as SGML elements, are meant to be attributes of the elements which contain them. They are not meant to be instantiated as stand-alone elements but are to be linked to any of the three elements that comprise the hierarchical structure. For the four main structural elements, `FULLDOC`, `SEQUENCE`, `SCENE`, and `SHOT`, we define a list of pre-defined attributes following the classification scheme of Section 4.1. For example, `startf` and `stopf` are attributes of the cinematographic structure indicating starting and ending frame numbers for a particular sequence. Combined with the frame rate attribute (`frate`) of the FULLDOC, they are used to recover the time reference from the raw video data file. The file attribute keeps the information describing where the raw video data are stored. The current implementation assumes that the file is stored locally but can be extended to support network scenarios. Finally, in the SHOT element, we introduce the transcr attribute to report the dialogue (e.g., closed captioning) present in the corresponding raw video data.

Another important attribute common to each element is the ID. This attribute constitutes the element identification and provides a mechanism for expressing cross-references. Its value is computed with a specific automatic coding. For example, the ID of the fourth shot of the second scene belonging to the first sequence will be: DOC1.SEQ1.SEC2.SHT4. Therefore, the shot position inside the hierarchy is comprehensible from the ID itself. If, instead, a reference to that shot is required, we use the REF element.

An example of a DTD used for news video annotation is shown in Appendix A. News consists of sub-categories (e.g., politics, sport, foreign, local) that are further subdivided. Upon loading a new video data file to be annotated, shots and shot breaks are located automatically by a cut detection algorithm. Because news delivery follows a single timeline, all shots belonging to a news item are contiguous. Therefore, we can associate a news item with the "scene" component. References to similar news items in other documents are stored as REFs. Additional content information about the news video is annotated as OBJECTs, whose attributes include name, type, metatype, time and date of creation, source from where obtained, and the popularity of a particular object (popularity is associated with news items for use in personalization).

4.1.3 Dynamic Interface and Metadata Definition

In the previous section we presented the baseline DTD used by Vane. This DTD is extended for different application domains. As long as the syntax is correct, Vane is designed to adapt to different DTDs without reconstruction or recompilation. The following are a few examples of modifications to the DTD that the Vane tool will recognize:

- **Category Definition**. As the CATEGORY attribute can assume only one value among a discrete set of possibilities (an enumeration), it was designed to keep the list of possibilities open to further additions. The same operation can be performed on each sub-category expansion (pull-down). CATEGORY, SUBCATEGORY and any other elements for which a set of possible values has been predefined, are shown by the interface as pop-up menus.

- **Attribute Order**. The order of the attributes expressed in the attribute list of each element reflects the order of the entries presented by the interface. For a better customization of the annotation tool, the order can be changed and replaced with the one that best suits the annotator's needs.
- **Attribute Definition**. Any other attribute considered relevant for the description of the video content in the domain under consideration, can be added to the main four elements: `FULLDOC`, `SEQUENCE`, `SCENE` and `SHOT`. The interface of the tool will adapt itself accordingly to what is specified in the resultant DTD.

4.1.4 The Role of the Annotator

The delineation of shots within an entire video data stream (e.g., a movie) is performed automatically by the segmentation algorithm provided with the tool. Here we assume that each camera break corresponds to change in the content of the video. (If not, the annotator can make corrections.) At this point the human annotator is required in the process and the delineated shots can be grouped on a logical basis. For example, in the educational domain the shots related to a lesson topic can be aggregated into the same "scene." Similarly we group related scenes into sequences. This activity is performed by the human expert with steering by the Vane tool. Forming such aggregates is possible when the related data are present as a contiguous set of frames. Subsequent reorganization (repurposing) of shots and scenes from the complete of pre-annotated content can be achieved at a later authoring stage. In this educational domain, shots that have the same title (e.g., in a lesson) can be automatically grouped together into a scene. The annotator also has opportunity to use the `OBJECT` representation of the content to encompass unstructured annotation data beyond enumerated fields.

The human annotator is clearly important in our annotation system. This leads to an apparent drawback of the subjective nature of human observation. To relieve the system from the dependence on this subjective factor, multiple annotations of the same video data can be obtained by different annotators. Afterwards, they can either be combined into a single annotated document or stored in the metadatabase and processed during the database search, merging the results obtained. Therefore, multiple interpretation can lead to additional detail available to support information retrieval.

Note however, if the validity of the information collected by the annotator is in question, it only bears on the retrieval process. That is, it in no way jeopardizes the validity of the raw video data. Once accessed, the raw data can be interpreted by the user.

4.2 The Vane Interface

The Vane user interface is the key component which links the video archive and the metadatabase with the human annotator. It must provide meaningful and complete dialog boxes to the annotator to allow the insertion of relevant information. Moreover, it must be consistent with the metadata format and the document definition specified for the current annotation. The same is true for any query or browsing interface provided to the end-user, but with perhaps a more limited video of the database contents. Among the specifications of the video data model, we require the capability of dynamic definition and extension of the document. If new indices and new fields are entered in the document definition, the interface must be able to "build itself" accordingly, so that the annotator is steered to input information considered important to the application domain (Fig. 4). However, the interface presented is also consistent across domains. This characteristic increases the usability of the tool as a front-end to a multimedia database system and potentially decreases the learning time for an annotator.

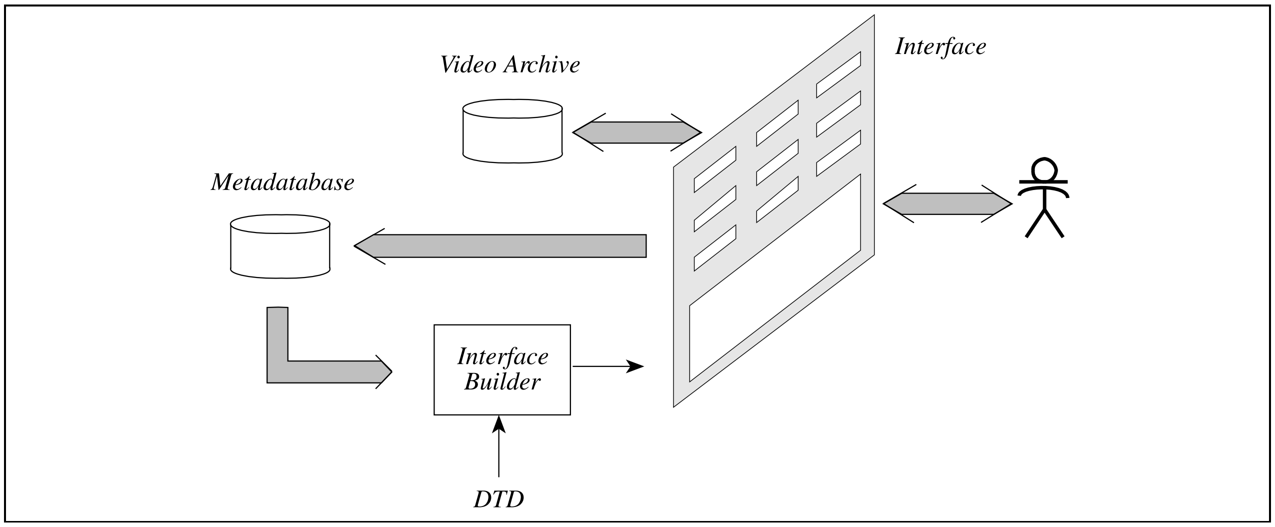


Figure 4: Role of the Interface of the Vane Tool.

The interface handles both raw video to be annotated and SGML documents already stored in the metadatabase. Outputs, on the other hand, are SGML documents, as the tool leaves the video archive untouched and does not perform any physical segmentation on the raw video data (video files are maintained as single entities). In the current implementation of Vane the video formats accepted as input to the tool are MPEG-1 video and MPEG-1 system streams. However, there are no specific dependencies on the MPEG-1 format. To be able to access video data at random points, we maintain offsets to groups of pictures (GOPs) in an associated file. Whenever an annotator wants to play a video segment, the start and end frames and the GOP size of the segment are passed to the video delivery engine to initiate playout.

We designed the graphical user interface taking in consideration all the aspects investigated in the previous section. We decided to depict the aforementioned hierarchical structure by three colored horizontal bars in which each represents a level in the hierarchy (Fig. 5). For complete correspondence between the levels and for "at a glance" visualization of relations between the levels, each is represented with the same time scale. The longer the duration of a video, the longer the bars will be. This provides the ability to expand and shrink the time scale to fit the entire time-line onto the working area. The units on the time axes can either be number of frames, or time in hours, minutes, and seconds. A fourth bar has also been designed to represent the entire video stream is associated with the "full document." Its purpose is to summarize the main metadata for the current stream annotation and to provide an overview of the current tags.

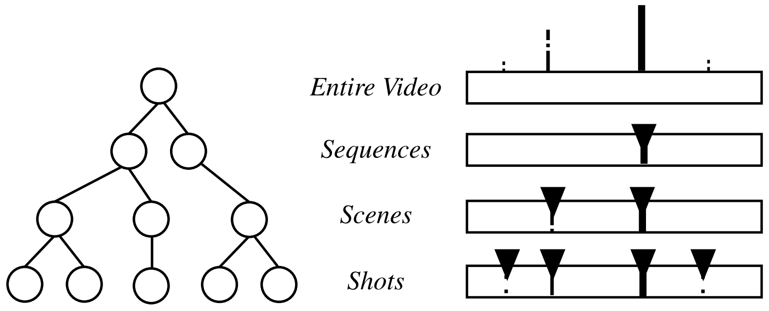
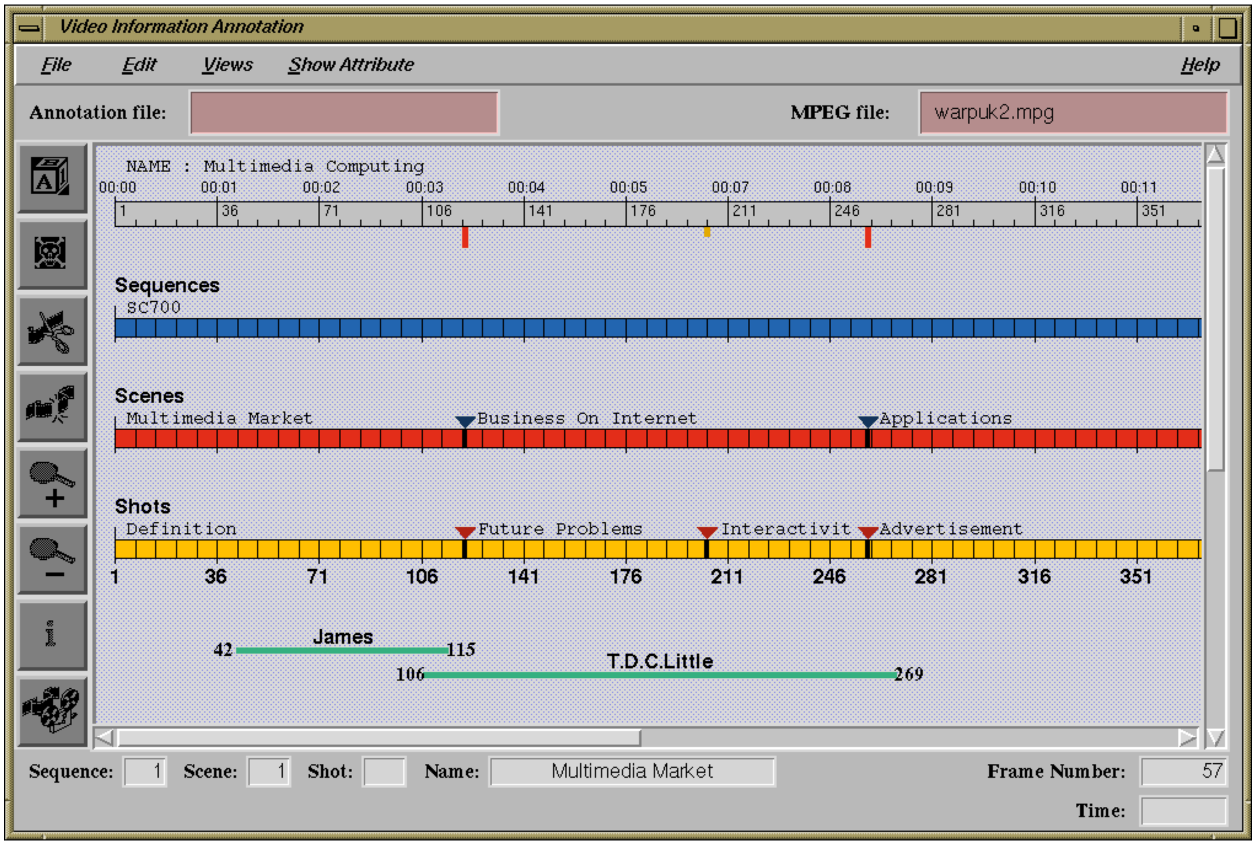


Figure 5: Design of the Interface.

A bottom-up approach is typically applied when a video stream is annotated from scratch. The annotation begins following automatic segmentation. As annotation proceeds, shots are aggregated to scenes and ultimately sequences. Object annotation is graphically represented by additional bars in the sequence, scene, and shot hierarchy.

The shot boundary segmentation algorithm we developed for Vane is based on the analysis of the sizes of the frames. As *I*, *B* and *P*-frames use different compression schemes, they cannot be compared. Therefore, only *I*-frames are compared by the algorithm. The consequence is an error as large as one half of a GOP size which is tolerable for most information retrieval applications.

Graphically, segmentation results in a division of the shot bar. This concept is applied to the other bars as well, and the set of frames between two contiguous breaks is called a segment. An intuitive method for visualizing these segments was designed so that the start and stop points (frames) can be seen very easily. On each bar, we implemented colored arrows to identify breaks. On the shot bar arrows show the starting and ending frame of each shot; on the scene bar arrows indicate where shots are grouped together into scenes; on the sequence bar arrows show the aggregation of scenes. The top bar acts as a time ruler where references for each cuts are reported to have a complete view over the current logical decomposition (Fig. 5). We also found it useful to offer the ability to visualize any element attribute on the bars based on user preference. For example the "category" of the video can be seen on the top bar, the name of the sequences on the second, the `ID` on the third and the keywords on the fourth.



Annotation file:
MPEG file: warpuk2.mpg
Sequence: 1 Scene: 1 Shot: Name: Multimedia Market Frame Number: 57
Time:

Figure 6: Screen-shot of the Vane Tool.

Separate dialog boxes, dynamically built using the scheme illustrated in Fig. 4, have been designed to let the annotator fill in all the fields defined in the current DTD. According to the domain, the different DTD will indicate a different interface for the collection of metadata. In addition, the top level, representing the video itself, can be annotated. The type of information that can be associated with it are bibliographical indices. All media-specific data on the video such as title and video category, or technical information as compression format or frame rate are required at this level.

The interface facilitates splitting shots to create multiple shots from an existing one. This action, which is often used during annotation, is called *break segment*. The inverse operation, joining two contiguous elements (“*join segments*”) is also supported. Arrows, which represent breaks, are draggable in different position along the bars providing an simple method to change the start and stop frame of the elements (e.g., to tweak automatic segmentation results).

During the annotation, a simple means to create hypermedia links between different elements is provided. Link references can be placed inside transcripts and abstracts using SGML. These are facilitated by interface tools that do not require the annotator to memorize SGML commands and syntax. A similar reference scheme is used for objects.

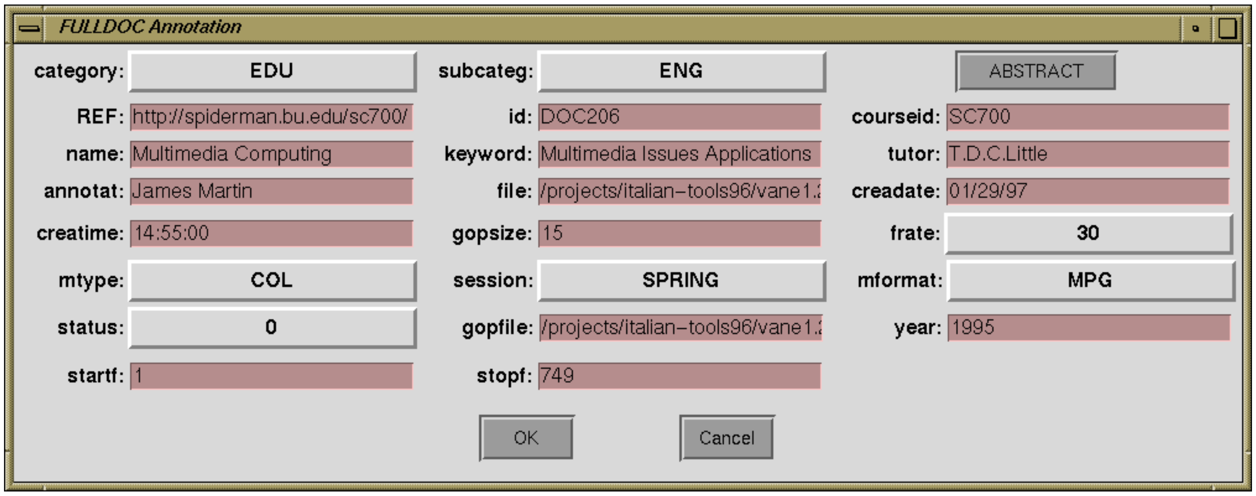
4.3 Vane Implementation

The Vane code is written using the Tcl/Tk scripting language described earlier. This choice was due to the large number of platforms which support this development environment. We used an SGI Indy running IRIX 5.3 for development and video playback. This platform allowed us to capture video in other formats as well, via built-in video capture hardware.

Fig. 6 shows a screen shot of the interface in the main window after an annotation file, previously constructed, has been opened. In this particular case, the associated DTD applies to the educational domain.

Color coding is used again to help the user in identification of the relationships among the elements. Thus, the color of a bar is the same used for the arrow in the lower level and for the vertical mark on the first bar, the one related to the entire video. In this way, yellow vertical marks on the first bar indicate the presence of a shot (yellow bar) in that position; red marks indicate scenes while blue stands for sequences. An overview of the entire annotation can be obtained by the analysis of the first bar. A time ruler is drawn here with two different scales (frames per second and hours, minutes and seconds). Note that the object view is separate from the hierarchy. The content objects are represented along the timeline in the color green.

To change the structure of the tree representing the annotation, new arrows, correspond-



FULLDOC Annotation

category:	EDU	subcateg:	ENG		ABSTRACT
REF:	http://spiderman.bu.edu/sc700/	id:	DOC206	courseid:	SC700
name:	Multimedia Computing	keyword:	Multimedia Issues Applications	tutor:	T.D.C.Little
annotat:	James Martin	file:	/projects/italian–tools96/vane1.	creadate:	01/29/97
creatime:	14:55:00	gopsize:	15	frate:	30
mtype:	COL	session:	SPRING	mformat:	MPG
status:	0	gopfile:	/projects/italian–tools96/vane1.	year:	1995
startf:	1	stopf:	749		

OK Cancel

Figure 7: The Annotation Window.

ing to breaks, can be inserted and existing arrows can be moved or deleted. These actions can be performed using the toolbar on the left and acting subsequently on the bars. For example, after the “joining operator” has been picked up, the cursor changes and the user has only to click on the arrow that must be erased. The annotation data present on the left segment will then be merged with the one belonging to the next element. The result constitutes the data of the joined segment.

Zoom-in and zoom-out options allow different depth of magnification for the views. When the zoom does not allow a global view, the working area can be shifted via a horizontal scroll bar. To change the start or the stop frame of an element (sequence, scene or shot), the corresponding arrow can be moved simply selecting and dragging it along the bar. When this action is performed, a vertical line appears and follows the movements of the arrow so that the precise position of the cursor can be read on the ruler. At the bottom of the window, we also added a status bar which keeps track of the mouse position. This bar provides information such as the frame number, the field name and the identification number of the currently pointed element.

Each segment identified in the annotation can be selected by a mouse click. Subsequently it appears as a dedicated dialog box. Fig. 7 shows the dialog box for the annotation of the whole video. In this example the fields that can be edited are related to the aforementioned educational DTD. A different DTD would result in a different automatic construction of the windows. The “Abstract” button will pop-up an additional window where the abstract of the segment can be entered. For the shot element we have also included a transcript window.

Objects belonging to unstructured data or defined as content objects are annotated by identifying the segments to which they belong. Each object in the interface is represented by a line and by clicking on a line a dedicated dialog box appears in which any data (description, type, origin, medium) concerning an object can be annotated.

5 Mapping of SGML Content to a Database Schema

After video data have been collected and annotated, the resultant annotations are stored as metadata using SGML. This is a convenient format for encapsulating collected metadata and supporting reuse. However, it is a bulky format that is not tailored to application needs. For example, one might seek to create search indices on the objects represented in a video database as well as keywords found in the audio transcripts. Such a scenario is appropriate for accessing on-line news video. In this case, an appropriate database schema can be defined that is far more efficient in space and access time than the raw SGML format. An additional feature of this approach is the ability to support changes in the end-application requirements. For example, because the the raw content and format is comprehesive, it can be translated to different output formats such as HTML 2 or HTML 3 by changes in the translator, not the content.

Based on an end-application requirement, the translation process can be defined. This will include mapping of tags to fields in a database schema, populating the data fields, and resolving hypertext references. In the following we describe one translation process that has been constructed to support SQL queries. The translator is called `sgml2sql`.

`Sgml2sql` is a conversion tool written to parse the SGML output of the Vane tool and to populate an SQL database. The `sgml2sql` implementation is modular in nature, built with the premise of supporting enhancements at the production side of the conversion. For example, a change of the database manager affects only the module which interfaces with the database.

`Sgml2sql` is written in `Perl 5` and uses the DBD/DBI (database interface) to communicate with the database. Currently we are using the `mSQL-DBD` package and the mini SQL database. However, the choice of DBMS is not significant for our functionality. `Sgml2sql` first runs an SGML parser on the SGML file under conversion. The parser checks the SGML file and its associated DTD file for any inconsistencies. If no errors are found at this stage then the tool reads the DTD-to-database-map file, consisting of a mapping between various

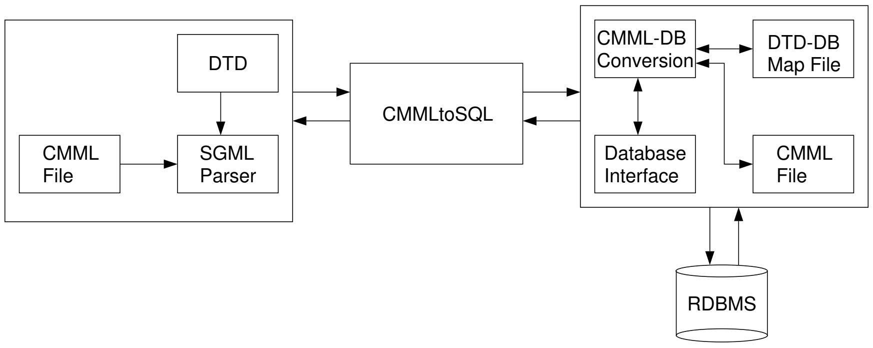


Figure 8: `Sgml2sql` Conversion Tool Architecture

table attributes to the fields in the database. An example of the mapping between an DTD and database schema in Fig. 9 for an instructional database application is shown in the Table 1.

As seem in the table, metadata about the "fulldoc" (complete video document) is mapped to "course," scenes are mapped to "topics," and objects are mapped to "object" in the database schema. Some of the attributes of the video document such as format, medium, frame rate, video file name, GOP metadata file name are mapped to the physical layer schema (Fig. 10) for providing information to the video playout engine on how the data should be handled. We populate the "course-topic" field by assuming that all the topics in the video document belong to same course. This assumption is based on the fact that all video data which we are annotating are already authored, but if a topic from any other course is to be added or existing topic is to be deleted then this can be achieved in a separate authoring process. The "scene" start and stop frames are utilized to populate the "topic-object" table in the database. By comparing the start and stop frame boundaries of an object and the topics we can find which object belongs to what topic. In the future, if any changes to the database schema are made, only the map file needs to be changed. After loading in the relevant contents from the SGML annotation file, the database interface module writes the appropriate fields to the database.

6 Summary

The goal of this work is to facilitate the construction of large and useful video databases supporting fast access to raw video content. To this end, the Vane annotation tool was

Table 1: Map Between SGML and DB

SGML Attribute	DB field
fulldoc.id	course.course_id
fulldoc.courseid	course.title_id
fulldoc.name	course.title
fulldoc.tutor	course.instructor
fulldoc.gopsize	save for physical_map.gop_size
fulldoc.frate	save for physical_map.frame_rate
fulldoc.mtype	save for physical_map.mtype
fulldoc.session	course.session
fulldoc.mformat	save for physical_map.mformat
fulldoc.gopfile	save for physical_map.mfilename
fulldoc.year	course.year
(fulldoc).category	course.subcategory
(fulldoc).sequence	IGNORE
(fulldoc).object	new object table entry
sequence.(except SCENE)	IGNORE
scene	new topic entry, new course_topic entry
scene.id	topic.topic_id
scene.name	topic.title
scene.keyword	topic.keywords
scene.imgfile	topic.image_file
scene.frame	topic.frame_num
scene.time	topic.time
scene.date	topic.date
scene.populaty	topic.popularity
scene.startf	SAVE FOR POST PROCESS
scene.stopf	SAVE FOR POST PROCESS
(scene).ref	new object and topic_object entry
(scene).abstract	topic.abstract
(scene).transcr	uniquely-named file
(scene).shot	IGNORE
shot.*	IGNORE
object	new object
object.id	object.object_id
object.name	object.name
object.type	object.type
object.metatype	object.meta_type
object.time	object.time
object.date	object.date
object.medium	object.medium
object.origin	object.origin
object.populaty	object.popularity
object.file	new physical_map
object.startf	physical_map.start_frm
object.stopf	physical_map.stop_frm

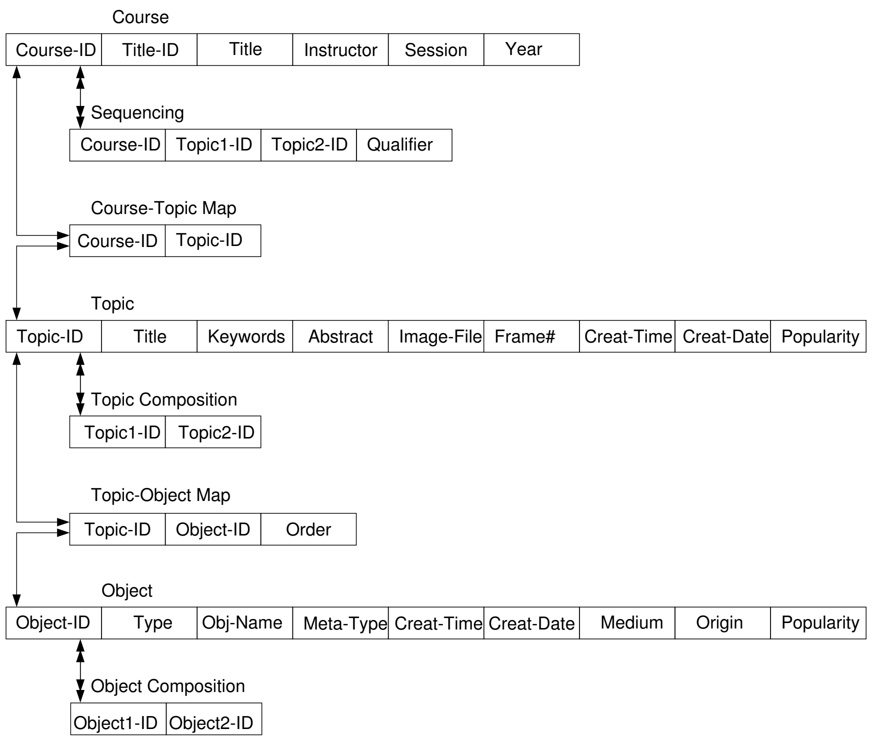


Figure 9: Instructional Domain Application-Specific Schema

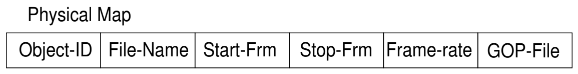


Figure 10: Physical Layer Schema

designed and constructed to provide a domain independent solution to metadata collection using SGML as a basis for the metadata model. In operation the tool interprets input DTDs to generate a view on a content model for a video domain and presents an interface tailored to that domain. In this manner, a single instance of the tool is appropriate for databases of news, educational materials, entertainment, or other video.

Vane is currently in use in the capture and annotation of a data set of over 500 hours of instructional video existing on VHS tapes at Boston University.

Acknowledgments: This work is supported in part by the National Science Foundation under Grant No. IRI-9502702. We extend our thanks to John Casebolt for implementing the SGML to SQL conversion utility and to William Klippgen for his support in the text-indexing area.

References

[1] G. Ahanger and T.D.C. Little. A survey of technologies for parsing and indexing digital video. *Journal of Visual Communication and Image Representation*, 7(1):28–43, March 1996.

[2] F. Arman, A. Hsu, and M-.Y. Chiu. Image processing on compressed data for large video databases. In *1st ACM Intl. Conf. on Multimedia*, pages 267–272, August 1993.

[3] V. Bhaskaran and K. Konstantinides. *Image and Video Compression Standards: Algorithms and Architectures*. Kluwer Academic Publishers, 1995.

[4] M. Carreira, J. Casebolt, G. Desrosiers, and T.D.C. Little. Capture-time indexing paradigm, authoring tool, and browsing environment for digital broadcast video. *SPIE*, 2417:380–388, May 1995.

[5] F. Chen, M. Hearst, J. Kupiec, J. Pedersen, and L. Wilcox. Metadata for mixed-media access. *SIGMOD Record*, 23(4):64–71, December 1994.

[6] G. Davenport, T.A. Smith, and N. Pincever. Cinematic primitives for multimedia. *IEEE Computer Graphics and Applications*, pages 67–74, July 1991.

[7] M. Davis. Media streams: an iconic visual language for video annotation. *Proc. IEEE Symposium on Visual Languages*, pages 196–202, 1993.

[8] W.I. Grosky, F. Fotouhi, and I.K. Sethi. Using metadata for the intelligent browsing of structured media objects. *SIGMOD Record*, 23(4):49–56, December 1994.

[9] A. Hampapur, R. Jain, and T. Weymouth. Digital video segmentation. In *2nd ACM Intl. Conf. on Multimedia*, pages 357–364, 1994.

[10] R. Hjelsvold, S. Langorgen, R. Midtstraum, and O. Sandsta. Integrated video archive tools. In 3*rd ACM Intl. Multimedia Conf.*, pages 5–9, November 1995.

[11] K.Böhm and T.C. Rakow. Metadata for multimedia documents. *SIGMOD Record*, 23(4):21–26, December 1994.

[12] W. Klaus and A. Sheth. Metadata for digital media: Introduction to the special issue. *SIGMOD Record*, 23(4):19–20, December 1994.

[13] W. Kou. *Digital Image Compression: Algorithms and Standards.* Kluwer Academic Publishers, 1995.

[14] J. Lee and B.W. Dickinson. Multiresolution video indexing for subband coded video databases. In *IS&T/SPIE, Conference on Storage and Retrieval for Image and Video Databases*, February 1994.

[15] A. Nagasaka and Y. Tanaka. Automatic video indexing and full-video search for object appearances. In E. Knuth and L.M. Wegner, editors, *Visual Database Systems, II*, pages 113–127. IFIP, Elsevier Science Publishers B.V., 1992.

[16] K. Otsuji and Y. Tonomura. Projection detecting filter for video cut detection. In *1st ACM Intl. Conf. on Multimedia*, pages 251–257, August 1993.

[17] J.K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley Publishing Company, 1994.

[18] R.W. Picard and T.P. Minka. Vision texture for annotation. *M.I.T. Media Laboratory Perceptual Computing Section Technical Report*, 1(302):3–14, 1995.

[19] L.A. Rowe, J.S. Boreczky, and C.A. Eads. Indexes for user access to large video databases. *SPIE*, 2185:150–161, February 1994.

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

[21] T.G. Aguierre Smith and N.C. Pincever. Parsing movies in context. *Proc. USENIX*, pages 157–168, Summer 1991.

[22] Y. Tonomura, A. Akutsu, Y. Taniguchi, and G. Suzuki. Structured video computing. *IEEE Multimedia*, pages 34–43, Fall 1994.

[23] B.B. Welch. *Practical Programming in Tcl and Tk*. Addison-Wesley Publishing Company, Upper Saddle River, New Jersey, 1995.

[24] H.J. Zhang, A. Kankanhalli, and S.W. Smoliar. Automatic partitioning of full-motion video. *ACM/Springer Multimedia Systems*, 1(1):10–28, 1993.

A News Video DTD

```
<!--Document Type Definition for Generalized WYSIWYG Example (FULLDOC)-->

 <!ELEMENT FULLDOC   --  (ABSTRACT?,CATEGORY?,REF*,SEQUENCE*,OBJECT*)>
 <!ELEMENT SEQUENCE  --  (ABSTRACT?,REF*,SCENE*)>
 <!ELEMENT SCENE     --  (ABSTRACT?,REF*,SHOT*,TRANSCR?)>
 <!ELEMENT SHOT      --  (ABSTRACT?,REF*,TRANSCR?)>
 <!ELEMENT OBJECT    --  (REF*,OBJECT*)>
 <!ELEMENT ABSTRACT  --  (#PCDATA & REF*)*>
 <!ELEMENT TRANSCR   --  (#PCDATA)>
 <!ELEMENT REF       -O  EMPTY>
 <!ELEMENT CATEGORY  --  (NEWS)>
 <!ELEMENT SCCATOGR  --  (POLITICS | SPORT | FOREIGN | LOCAL)>
 <!ELEMENT NEWS      -O  EMPTY>
 <!ELEMENT SPORT     -O  EMPTY>
 <!ELEMENT POLITICS  -O  EMPTY>
 <!ELEMENT FOREIGN   -O  EMPTY>
 <!ELEMENT LOCAL     -O  EMPTY>

<!ATTLIST NEWS subcat (morning | mid-day | evening) morning>
<!ATTLIST SPORT subcat (basket | soccer | football | ski | baseball) basket>


<!ATTLIST FULLDOC
 id          CDATA                              #IMPLIED
 keyword     CDATA                              #IMPLIED
 anchor      CDATA                              #CURRENT
 producer    CDATA                              #IMPLIED
 location    CDATA                              #IMPLIED
 language    CDATA                              #IMPLIED
 annotat     CDATA                              #CURRENT
 country     CDATA                              #IMPLIED
 videofile   CDATA                              #REQUIRED
 creadate    CDATA                              #IMPLIED
 creatime    CDATA                              #IMPLIED
 frate       (30 | 24 | 15)                     30
 mtype       (col |BW)                          col
 mformat     (mpg | cosmo | qt | par | avi)     mpg
 startf      NUMBER                             1
 stopf       NUMBER                             #REQUIRED>
```

```
<!ATTLIST SEQUENCE
 id       CDATA   #IMPLIED
 name     CDATA   #REQUIRED
 keyword  CDATA   #CURRENT
 file     CDATA   #CURRENT
 startf   NUMBER  #REQUIRED
 stopf    NUMBER  #REQUIRED>


<!ATTLIST SCENE
 id        CDATA   #IMPLIED
 name      CDATA   #REQUIRED
 keyword   CDATA   #CURRENT
 populaty  CDATA   #IMPLIED
 startf    NUMBER  #REQUIRED
 stopf     NUMBER  #REQUIRED>


<!ATTLIST SHOT
 id       CDATA   #IMPLIED
 name     CDATA   #REQUIRED
 keyword  CDATA   #CURRENT
 startf   NUMBER  #REQUIRED
 stopf    NUMBER  #REQUIRED>


<!ATTLIST REF
 target  CDATA  #IMPLIED>

<!ATTLIST OBJECT
 id        CDATA   #REQUIRED
 name      CDATA   #REQUIRED
 type      CDATA   #IMPLIED
 metatype  CDATA   #IMPLIED
 time      CDATA   #IMPLIED
 date      CDATA   #IMPLIED
 medium    CDATA   #IMPLIED
 origin    CDATA   #IMPLIED
 populaty  CDATA   #IMPLIED
 startf    NUMBER  #REQUIRED
 stopf     NUMBER  #REQUIRED>
```