A Tcl/Tk-Based Video Annotation Engine[1]

M. Carrer, L. Ligresti, 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

Abstract–The population of a video database requires tools for manipulation and annotation of raw video data. Characteristic of this requirement is the need to satisfy many disparate video-based application domains. In this paper we describe the design and development of a video annotation engine called Vane, intended to address the issue of domain-independent video annotation. Rather than relying on a single, general data model and application interface, we developed a dynamic interface and data model using the Tcl/Tk environment and SGML document type definitions (DTDs). This approach allowed us to implement an intuitive graphical user interface application that is easily portable to different systems. The tool is currently in use to annotate a video archive comprised of educational and news video content at Boston University.

Keywords: Tcl/Tk, GUI, video annotation, database, metadata, SGML.

[1]*Proc. 5th Annual Tcl/Tk Workshop '97*, Boston, MA, July 1997.

1 Introduction

A *Video Database* is a system which provides solutions for storing and then retrieving digital video. The primary distinction of a video database is the ability to rapidly locate and retrieve video content. 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 [5].

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 for 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 seek to "front-load" a multimedia database rich in video content. 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 an annotation tool, we make use of SGML (Standard Generalized Markup Language) whose context rules are highly customizable through definitions in a DTD (document type definition) tailored for video content. Therefore, we can 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, describing information content (annotations), 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.

On top of this structure, one can develop a user interface and system I/O for the annotation tool. Such an application interface must not only present common characteristics such as usability and consistency among different domains but must also be aware of the context rules used in the particular domain of the current annotation. Moreover, it must behave according to what is specified in the domain DTD and therefore must present a different layout for each specific domain. It must be very flexible offering powerful graphical means for editing and modifying existing annotation documents.

To this end, we have used Tcl/Tk as a scripting language and toolkit for implementation of our annotation tool. The outcome of our work is Vane, a multipurpose, domain-independent video annotation application [4, 3]. It has been developed taking advantage of the most advanced Tcl/Tk features for an easy building and reconfiguring of GUI widgets at execution time. Thereby offering a novel application model for Tcl/Tk.

The remainder of this paper is organized as follows. In Section 3 we consider the fundamentals of the video annotation domain. In Section 3 we describe the details of the annotation tool. Section 4 concludes the paper.

2 Background

Before proceeding with a detailed explanation of Vane and its novel use of Tcl/Tk, we introduce our context with an introduction to video databases and SGML.

2.1 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 [1].

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. 1 illustrates the overall process of video annotation that we describe here.

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. 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.

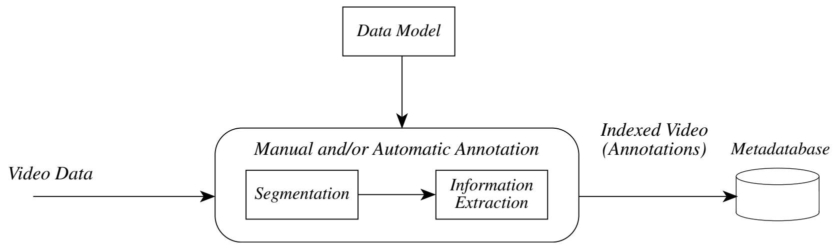


Figure 1: Block Diagram of the Functions and Data Flow in the Video Annotation Process.

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.

2.2 SGML

In the early 1980s, the International Standards Organization (ISO) has proposed the Standard Generalized Markup Language, (SGML) (ISO 8879), as a mean of managing and organizing information [2]. It was aimed to increase the portability of a wide range of documents between different computer-based systems.

The main idea behind SGML is the separation of the content of a document from the way it should appear in the output: information is managed as data objects instead of characters on a page. This division is obtained using *markup*: a series of instructions, mixed or embedded in the text of the document, which gives the system all the necessary processing rules. Traditionally, *procedural* markup has been used in most electronic publishing system, giving the typesetter the necessary directions about how to fit the document text on the final page. SGML makes use of *generic* markup, also known as *descriptive* markup: rather than focusing on how the text should graphically appear on the page, generic markup defines the purpose of the text in a document. Data is broken into discrete objects of information, called *elements*, that carry intelligence about their meaning within the overall system. Elements are organized in a strict logical structure which defines a rigorous hierarchical model for the document. To understand how SGML works, the relationship between the content and the structure of a document can be analyzed in two different layers:

- **Structure**: a file called the Document Type Definition, (DTD) sets up the whole document structure. It provides a framework for the types of element that constitute a document; it also defines the whole hierarchical 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 the method used by SGML to isolate the content within the document: by using *start* and *end tags*, logical elements are identified. With SGML, it is possible to associate *attributes* of arbitrary type to any element: user-defined or enumeration, numbers and strings. Elements can also be nested within other elements determining the organization of the document. As tags should strictly respect the set of context rules defined in the DTD,

special software, called SGML Parsers, have been designed to check the consistency of a document with its DTD.

In brief, this technology enables the efficient storage and reuse of the information, sharing it with many users and maintaining it in a database. We apply SGML as a format to capture and characterize video information as metadata in the Vane tool. Additional details of SGML can be found elsewhere [2].

3 Vane: the Video Annotation Engine

In Section 2.1 we described the fundamental role of annotation in the process of organizing and building a video database. 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.

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. Therefore, the Vane tool 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. Before considering in details of the latter, and thus the main relevance of this work to Tcl/Tk, some considerations about the video data model design are necessary. As you will see, these considerations will lead to precise requirements that have to be satisfied by the user interface, directly addressing Tcl/Tk features.

3.1 Video Data Modeling and DTD

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.

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

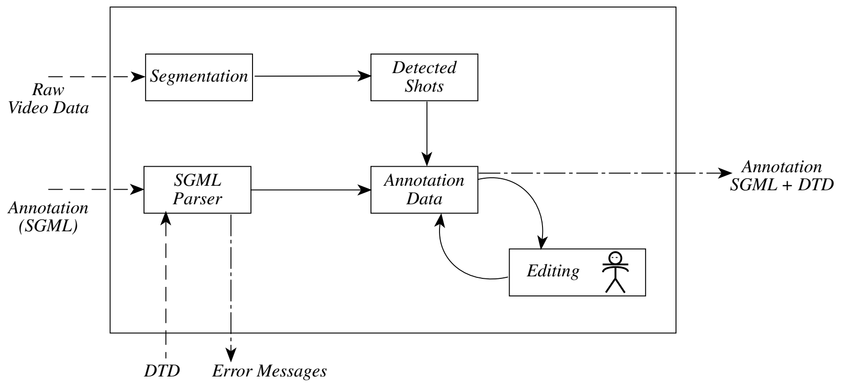


Figure 2: Block Diagram of the Vane Tool.

notification of mismatches between the opened annotations and the corresponding DTD. Fig. 2 is a block diagram of the Vane tool.

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.

We took all these concepts and ported them into a document type definition which takes 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. All the objects and their roles are here defined. An object is identified by an `ELEMENT` and its content model is enclosed in following parentheses. A set of attributes of default attributes is also associated to the four main elements, `FULLDOC`, `SEQUENCE`, `SCENE` and `SHOT`.

For every `ELEMENT` an entry field has to be defined in the Vane tool. While the annotator is working, Vane should configure itself accordingly to the `ELEMENT`s defined in the current DTD. In the next section we present the set of rules that governs the mapping between the DTD and the annotation window interfaces. Referring to Tcl/Tk terminology, this mapping defines the type of widget that it is used to represent a field of the DTD.

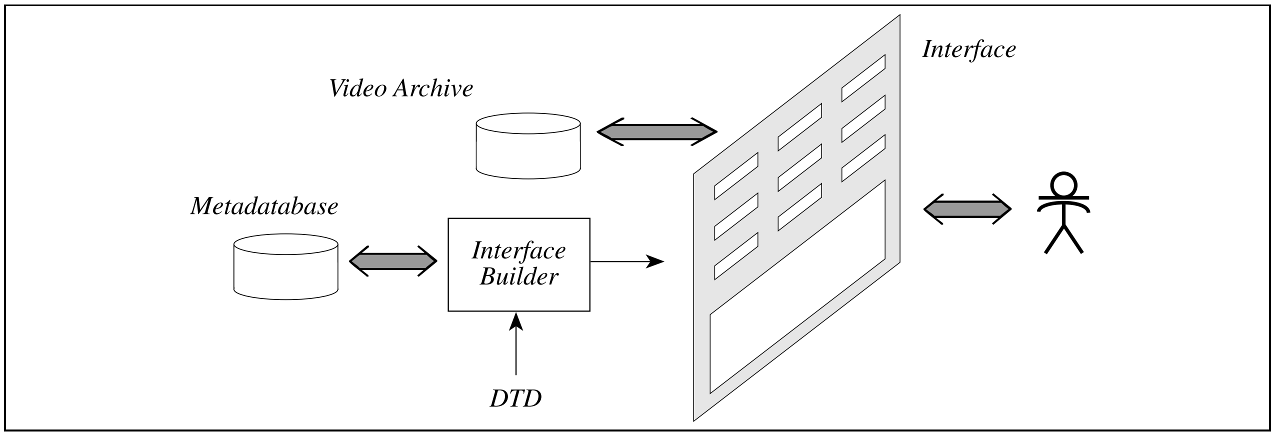


Figure 3: Dynamic Interface Definition in Vane.

3.2 Tcl/Tk and the Dynamic Definition

The 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 all the relevant information. On the other hand, it must also be consistent with the metadata format and the document definition specified for the current annotation.

We built Vane so that the DTD can be modified in many of its parts to make it suit the annotator needs and to better describe the current domain under analysis. Opening a new annotation in Vane means to identify the DTD associated to it. That particular DTD is then parsed by a Tcl procedure loading its syntax rules in an array in memory. When the annotation of one of the SGML elements defined is requested by the annotator, a new top-level window is built. Attributes belonging to those elements are mapped to a Tcl/Tk widget accordingly to their type; it could be a pop-up menu, an entry, a listbox, or a text area for attributes such as transcript. The sequence of steps followed to build an annotation window is depicted in Fig. 3. Defining a new DTD for a new annotation or changing an existing DTD in some of its parts will not affect the Vane implementation as all annotation windows are built on-the-fly. Here is a list of the possible changes that are currently recognized on an existing DTD:

- **Category Definition**. As the `CATEGORY` attribute can assume only one value among a discrete set of possibilities, it was thought to keep the list of possibilities open to further additions. The same operation can be performed on the each subcategory expanding, therefore, their sets of values. `CATEGORY`, `SUBCATEGORY` and any other elements for which a set of possible values has been predefined, are shown by a Tcl/Tk pop-up menus.
- **Attributes 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 annotator's needs.
- **Attributes 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 specified in the DTD. Some options are offered: if the new attribute's value can be chosen only among a defined set, the interface shows it up as a pop-up menu. Otherwise, a normal entry box is displayed.

The features offered by Tcl/Tk for string and widget handling have been essential in order to accomplish these characteristics of Vane. The powerful regular expression commands let us build the DTD parser in a snap. Moreover the unique feature of using strings as array indices lets us store the parsed syntax in a easily accessible Tcl array. There was no need for complex data structure and that reduced our implementation time by a considerable factor. Once we had that, mapping elements' attributes to widget and building the corresponding annotation window has been straightforward.

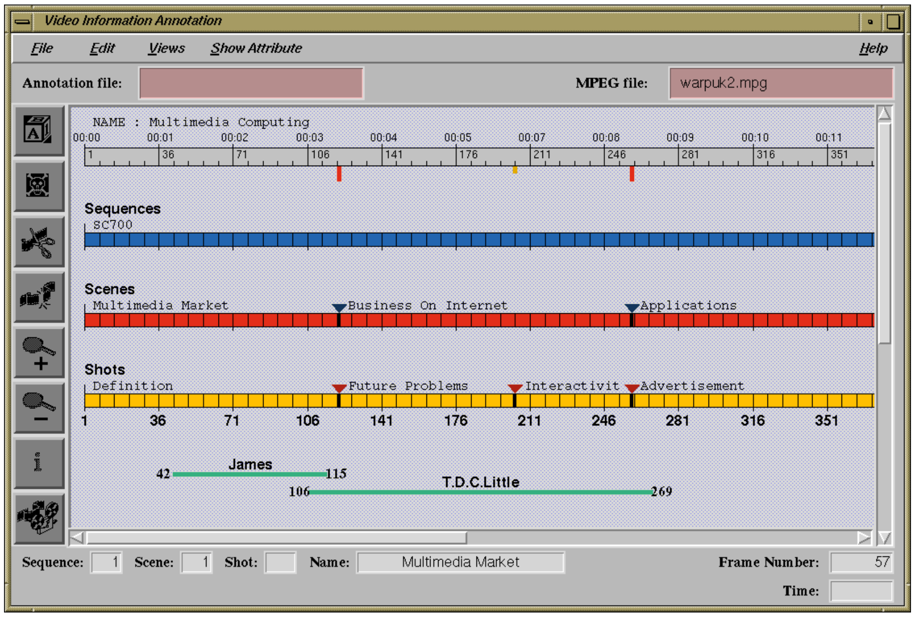
We did not use arrays only to store DTD syntax. From the DTD array we build another empty array which holds the annotation itself. As shown in the previous block diagram, loading an annotation means running a SGML parser through it in order to check for syntax mismatches. If no errors are found, the SGML parser reformats the original SGML document into an internal format. Another Tcl procedure parses this file and loads the field values in the annotation array. Being the array indices based on the DTD array (and therefore the DTD syntax), an extra level of checking for mismatches has being added.

The reverse process is applied to save annotations in SGML format. In this case we browse through the whole annotation array and generate SGML code following the syntax rules stored in the DTD array. The result is a "generated" SGML document which is certainly consistent with its own DTD.

3.3 Details of the Implementation

We designed the graphical user interface taking in consideration all the aspects previously discussed. We decided to depict the three level hierarchical structure of Section 3.1 with three horizontal colored bars, each representing a level with a time line on the horizontal axis (Fig. 4). The units on the time axes can be either number of frames or hours, minutes, seconds. A fourth bar represents the entire video, also called "full document"; its purpose is to summarize the main information of the whole annotation. When only a summary of the annotation is needed, the option to shrink the main window can be chosen at any time.

Graphically, the segmentation results in a division of the shot bar. The same idea of "graphical split" can be applied to the other bars and the set of frames between two contiguous breaks will be generally called a segment. An intuitive way to visualize these segments has been 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 (at the bottom) arrows show the starting and ending frame of each shot; on the scene bar, instead, they are set where shots are grouped together in scenes and similarly it works on the sequence bar. We used a colored coding scheme in the interface so that each level is associated to its own color. 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. We also



Annotation file:

MPEG file: warpuk2.mpg

Sequence: 1 Scene: 1 Shot: Name: Multimedia Market Frame Number: 57

Time:

Figure 4: Screenshot of the Video Annotation Tool Main Window.

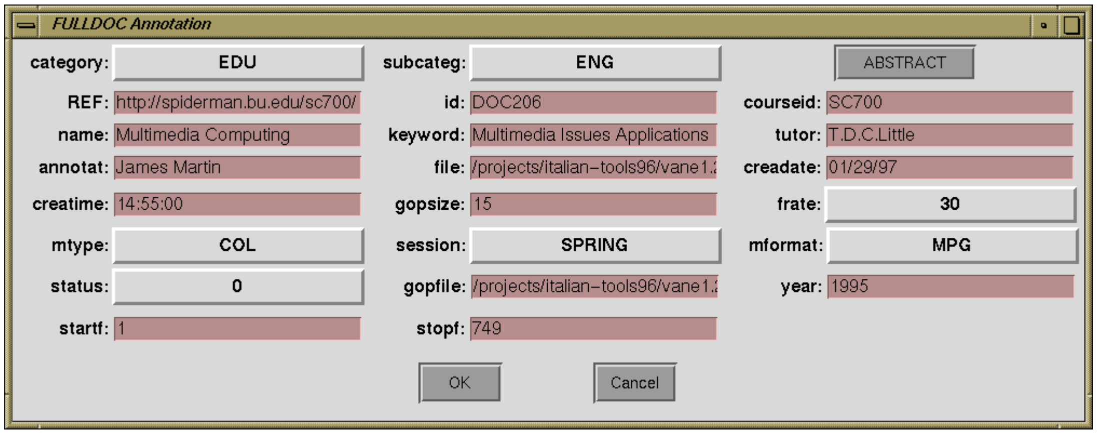
thought it useful to visualize the attributes of each element according to the choices taken. For example the category of the video can be seen on the top bar, the name of the sequences on the second, the `ID` on the third and the keywords on the fourth.

In Fig. 4, a column of buttons on the left side of the main window can be seen. This series of buttons has been inserted in the frame for quick access to the main functions. The commands that can be used from this area are: break, join, zoom in, zoom out, edit properties, and play. In this way users do not have to access the menu bar for any kind of operation. The zoom in and out options allow different degree of precision for the views; when the zoom does not allow a global view, the working area can be shifted thanks to a horizontal scroll bar.

Each single segment identified in the annotation can be selected by clicking on it. It will be highlighted by changing its color to black. In this way those actions, which are performed on one or more elements, can be easily accomplished after the selection of the element itself. For example, to annotate or play only a particular shot, the segment under analysis has to be selected and then the operation required can be chosen from the menu bar. If there is no current selection, the entire video (`FULLDOC`) is played or annotated.

Separate dialogues boxes, dynamically built using the scheme in Fig. 3, have been designed to let the annotator fill in all the fields defined in the current DTD (Fig. 5).

To change the structure of the annotation, new arrows, which represent breaks, can be inserted and existing arrows can be moved or deleted. To change the start or stop frame of an element (s equence, scene or shot), the correspondent arrow can be moved simply selecting and dragging it along the bar. When this action is performed, a vertical line appears and is moved together with the arrow so that the precise position of the cursor can be read on the ruler. The interface allows to "split" a segment, inserting a new arrow and, for example, creating two shots from one already present. This action, which is often used, is called *Break Segment*. The opposite operation, joining two contiguous elements (*Join Segments*) is also allowed. A fast and practical access to these functionalities has been provided through the toolbar on the left side on the annotation window.



category:	EDU	subcateg:	ENG		
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		

Figure 5: Annotation Window for the Whole Video.

To access the most important information of a particular element in every moment, the implemented status bar is very comprehensive. Keeping track of mouse pointer movements around the main window, the frame number, the name field and the identification number of the currently pointed element are updated in the bar in real time.

To make the main window even more information-rich the tool also provides a display the attributes of each level in the information hierarchy. The choice of the attribute is made through menus, and the results obtained are shown in Fig. 4 where, for example, the start frame attribute is visualized in the sequence and shot bars while the ID is reported in the scene bar.

4 Conclusion

The aim of the Vane tool was the design and implementation of an application for the annotation of video information. From considerations related to the complexity of dealing with video data belonging to different application domains, requirements for the Vane tool have been recognized and implemented. To this end, the Vane annotation tool was produced using Tcl/Tk to provide a domain independent solution to metadata collection using SGML as a basis for the metadata model. This approach lead us to the novel use of Tcl/Tk as a means for building dynamically configured graphical user interfaces starting from the DTD of an SGML "video" document.

The application of the similar concepts on different SGML-derived dialects, such as HTML, might lead to interesting results. This is, however; outside the scope of the development of Vane.

References

[1] G. Ahanger, D. Benson, and T. D. C. Little. Video query formulation. *IS&T/SPIE Symposium on Electronic Imaging Science and Technology (Storage and Retrieval for Image and Video Databases III)*, 2420:280–291, February 1995.

[2] M. Bryan. *SGML: an author's guide to the Standard Generalized Markup Language.* Addison-Wesley Publishing Company, 1988.

[3] M. Carrer. Environment for the annotation of video information via metadata collection and management. Thesis, Department of Electronics and Informatics, University of Padova, Italy, March 1996.

[4] L. Ligresti. Environment for capture, analysis, and annotation of video information. Thesis, Department of Electronics and Informatics, University of Padova, Italy, March 1996.

[5] R. W. Picard. Light-years from lena: Video and image libraries of the future. *M.I.T. Media Laboratory Perceptual Computing Section Technical Report*, 1(339), 1994.