Technical Report TR02-007

Department of Computer Science
Univ of North Carolina at Chapel Hill

Automated Hyperlink Markup for Archived Video

David Stotts, Jason Smith

Dept. of Computer Science
Univ. of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175 USA
1-919-962-1833

stotts@cs.unc.edu

February 15, 2002

Automated Hyperlink Markup for Archived Video

David Stotts
Dept. of Computer Science
Univ. of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175 USA
1-919-962-1833
stotts@cs.unc.edu

Jason Smith
Dept. of Computer Science
Univ. of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175 USA
1-919-962-1799
smithja@cs.unc.edu

ABSTRACT

One of the problems keeping video from being a fully first-class data component of hypermedia documents is the difficulty of treating the objects depicted in video as identifiable, linkable content. Rather, video tends to be manipulated as frames of pixels with no further subdivisions. When link markup is done on video streams, it is done manually frame-by-frame. We have been working with OvalTine, a system for tracking objects in video streams so that hypermedia link anchors can be associated with the objects in the video frames. In this work we show how the OvalTine tracking system can be used to do automated link markup of video streams. While our previous work with OvalTine presented object tracking in real-time streams, we concentrate here on markup of stored (archived) data. Our results will allow hypermedia structure to be generated and added to large digital libraries of video data.

General Terms

Algorithms, Documentation, Experimentation, Human Factors

Keywords

Video, image analysis, markup, hypermedia, editing, digital library, tracking

1. MOTIVATION AND BACKGROUND

We previously reported on the OvalTine system, and discussed the various distributed system architecture issues involved in storing and serving video hyperlinks in a client/server implementation [1]. This paper presents our continuing work with the system, demonstrating the use of the basic real-time image tracking algorithms for use in automated markup of stored video data. In the taxonomy we outlined previously [1], this is a *Server/Archived* scenario, and we have chosen to implement a *Manual* object selection scheme.

In our initial OvalTine implementation, we demonstrated real-time tracking of faces in live video streams, such as you might have in a real-time videoconferencing system. This tracking allowed hypermedia link anchors to be associated with objects in the video window, creating a first-class hypermedia capability for video data. The face in the video frame became a live link, a selectable target for the user to click on to trigger some action. As the face moved around the video screen, the live target area moved with it, providing the illusion that the face itself was the hyperlink anchor. The target was optionally made visible by means of a simple highlight oval that moved with the face. Clicks within the oval counted as a click on the face, and the enveloping hypermedia layer followed the link associated with the face. The links in that demonstration were URLs entered when the object was first selected as a link anchor (at the initiation of tracking); selecting one triggered the display of a web browser window with the proper URL loaded.

In this paper we present a second, equally important use for the OvalTine technology – automating the addition of link markup to stored (non-real-time) video data. There is increasing interest in video data being incorporated in hypermedia structures (which we will hereafter refer to as *hypervideo data*). Digitial libraries are growing in popularity and scope, and video is an important component of such archives. All major news services have vast video archives, valuable "footage" that would be of use in education, historical research, even entertainment.

The current best practices for link markup in video require completely, or considerably, manual markup of the video frames with the active, or hot, areas that serve as link anchors in hypervideo data [9]. Broad access to the vast stored video "footage" just mentioned will only be possible with automated link markup methods.

We do not specify to what a link anchor may refer, nor do we limit the type of objects which may be designated as a link anchor. The goal of our work is to explore the creation and maintenance of hyperlinks in video streams, and to automate these procedures as much as possible.

Before any technical discussions, we look at previous work in hypervideo.

1.1 Related work in hypervideo

We use the term hypervideo to refer to a displayed video stream that contains embedded user-clickable anchors. These anchors are logically attached to objects within the video environment, independent of location within the field of view [7]. For instance, a person's face in a video conferencing system may be designated as a hyperlink. As the person moves within the image, the clickable area that activates that hyperlink will move with the image of the face, so that the face itself defines the active region.

Multimedia has been trending from text to static images to video. On a parallel path, hyperlinking systems, most notably the World Wide Web, have evolved from the use of pure text to the incorporation of static images with embedded anchors (image maps), and now to video streams with active regions [9, 17].

A good overview of the issues and technologies in current hypervideo systems can be found in class notes at Texas A&M [8]. In the terms defined in this taxonomy, we are working on a system for automatically specifying mostly *spatio-temporal* links in hypervideo. Hypercafe is often cited [7] in hypervideo discussions; however, it is a presentation system mostly and does not support the dynamic and automatic link anchor creation we are exploring in OvalTine. The Multimedia Systems Lab at IISc India is doing work on object tracking in MPEG streams [22]. This project seems similar in scope and goals to OvalTine. They are tracking object in an MPEG stream, where as OvalTine is architected to be modular and extensible to define tracking and linking concepts at an abstract level, and to be applicable to different image and video formats with minimal extensions. Most other systems, however, that apply to hypervideo involve manual anchor creation when authoring hypervideos.

Every current popular method for adding active regions to video requires manual selection of video objects, on a frame-by-frame basis. No research efforts in automation have yet made it into common practice in a widely used system such as those from Apple (QuickTime). By contrast, an automatable object tracking system is much more desirable, both for real-time applications, and for the automated addition of hyperlinks to the vast amount of archived video currently in existence.

Automation of content in hypertext is a well-researched area [10, 11, 13]. While much work has been done in the realm of context-assisted anchor creation in video, particularly in news coverage videos, these rely on a blend of modal data, including much that is manually entered by human operators [12, 15, 20, 21].

We are more interested in identifying issues related to nearly context-free object tracking within image streams. The image analysis engine selected for use on the video, or the user manually creating links, supplies the context. A facial recognition system designed to attribute a link to a database record consists of an entirely different inherent context than an engine which identifies and tracks types of automobiles on a roadway, but they both can operate equally well on the same raw video stream. This is a complementary concept to contentoriented navigation [14], where the context is provided during modal analysis. Since the context of the link data has been removed from the dimensions we define, the resultant contextual link generation engines are also close analogues to Sprocs as defined by Nürnberg, et al [18].

Text is almost exclusively an archived data source, authored once and stored, and then presented as pre-established data to the user. Research into dynamic text systems is established [11, 16, 19], but it is our opinion that text will not reach the temporally and spatially dynamic properties of video in the near future. Video allows for, and generally requires, different approaches to anchor generation than does text.

2. SPRITE LINK LAYER OVER VIDEO

The previous OvalTine system did tracking of object sin video streams in real-time. While this has utility in creating hypermedia capabilities within previously inaccessible data domains like video-conferencing, it did not have enough infrastructure to capture and save all the calculated link anchors. They were computed frame-by-frame and then lost as the next frame came along.

To markup archived video for playback, the basic tracking capabilities of OvalTine had to be augmented with a capture and storage layer for links.

Our original plan was to enhance our existing OvalTine application on the SGI IRIX platform, utilizing the existing support for QuickTime saved movies. QuickTime is a container file format and access specification, not a movie *codec*, and allows for any QuickTime capable application to handle any media format for which a *codec* exists in the rather expansive QuickTime library. Multiple types of data can be added to a QuickTime file in *tracks*, such as video of various compression schemes, sound data, text, or any other time-variant information. QuickTime's primary function is to ensure that these tracks are kept synchronized. To reduce

confusion with the tracking properties of OvalTine, we will use the term *layer* in this paper to refer to QuickTime *tracks*.

The implementation of QuickTime on IRIX, however, does not offer support for *sprites*. In the QuickTime nomenclature, sprites are object-based animations in a layer that is separate from, but synchronized with, the video layer. Sprites are usually used to produce simple animations, moving a series of static images around a video field much like traditional cartooning. Sprites exist in a QuickTime layer of their own, and can be thought of as an overlay over the video layer. The animations are separate from the video, so can be included in the user presentation, or not, by easily toggling a visibility flag. Each animation can be individually visible or not, or the entire layer can be toggled at once.

Wired sprites are sprites that react to user interaction, registering events such as mouse-over (the cursor has been moved over the sprite by the user), mouse-down (the user has pressed the mouse button while the cursor is over the sprite), and mouse-up (the user has released the mouse button while the cursor is over the sprite). The wired sprites know their own boundaries and automatically track when the user is potentially interacting with them. They also keep track of which actions to perform for which user events, and these actions can include simple changes to the presented image, to moving within the movie to another timecode, opening another movie and jumping to any timecode within the new movie, and requesting actions of the system of other applications.

Since the object tracking library has gone through a redesign to be cross-platform implementable, we have taken the opportunity to move OvalTine to a more commercial OS with stronger QuickTime support, including wired sprites. The rapid application development environment of MacOS X is the current choice for development. The user interface is designed to be minimal at this stage as we investigate options for redesign of the original real-time tracker interface as well.

Wired sprites offered exactly what we needed to make a video markup layer – frame-by-frame synchronization, an overlay layer to contain the tracked object regions, with visibility toggling, built-in hit (click) detection, and 'link' traversal to trigger actions on a successful click by the user viewing the movie file. The file format is openly published, and QuickTime is fully supported on several platforms, including the various versions of Windows, and the two MacOS variants.

The OvalTine tracker system makes calls to the QuickTime library as it tracks objects in a frame. As an object's new location is discovered by the tracker, its coordinates are transmitted to the sprite layer and recorded. When tracking is completed, the sprite layer information is incorporated into the QuickTime playback so the user sees the original video frame as well as the overlaid sprite annotations. In the next section we discuss the association of link targets (URLs) with the wired sprites.

2.1 Example markup session

A sample session might involve a user loading a video stream, perhaps an existing news archive footage file. The goal is to add hyperlinks to the video pointing to biographical information for selected subjects in the video. The user begins by selecting a QuickTime file to work on through use of the *Open... menu* item. The movie appears in the video view, at the first frame. The user uses the standard movie controller interface to move to the segment of the video they wish to begin tracking in. The *editing mode* is chosen from radio button palette, and a rectilinear box is drawn around the face to be tracked. An oval will appear around the face to indicate that the tracker has acquired it If the registration of the oval to the face is not pleasing to the user, they can simply choose the *Undo* menu item, and select again.

Once the object has been selected appropriately, the user can add the necessary tag information to be used when the object is later clicked on during playback. The *Links* window contains bookmarks to items the user may wish to add to an anchor, including http URLs for browser viewing, file URLs for local file viewing, or movie URLs that trigger movement to another time-code in the current movie; they might also open another movie and jump to a particular time-code in it. A link can be dragged and dropped onto the oval desired to be its trigger.

The user can now initiate a frame by frame tracking by selecting *Track* from the *File* menu or toggle the *Track* button in the main window. As the tracker acquires the face in each frame, its reported position is displayed as an overlay over the movie. At the same time, the sprite, consisting of the oval, is saved to the movie data in memory. If the user at any time is unhappy with the tracking performance, they can stop the process, scan back to the first frame they feel is incorrect (again, using the standard movie controls), and re-select and re-initiate tracking. When the user is satisfied with the results of the tracking, the movie can be saved back to disk. Any QuickTime capable application will now be able to show the tracked face in the movie, and register hits on the face if clicked upon. The action on link traversal will be dependent on the capabilities of the particular viewer application. A web browser will be able to traverse URLs, for instance.

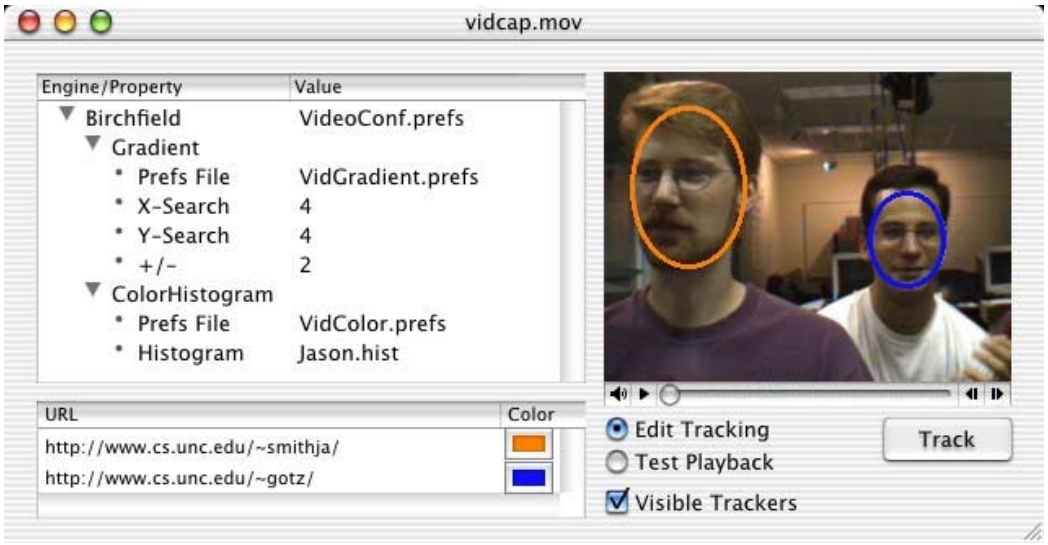

Figure 1. OvalTine user interface showing two tracked objects

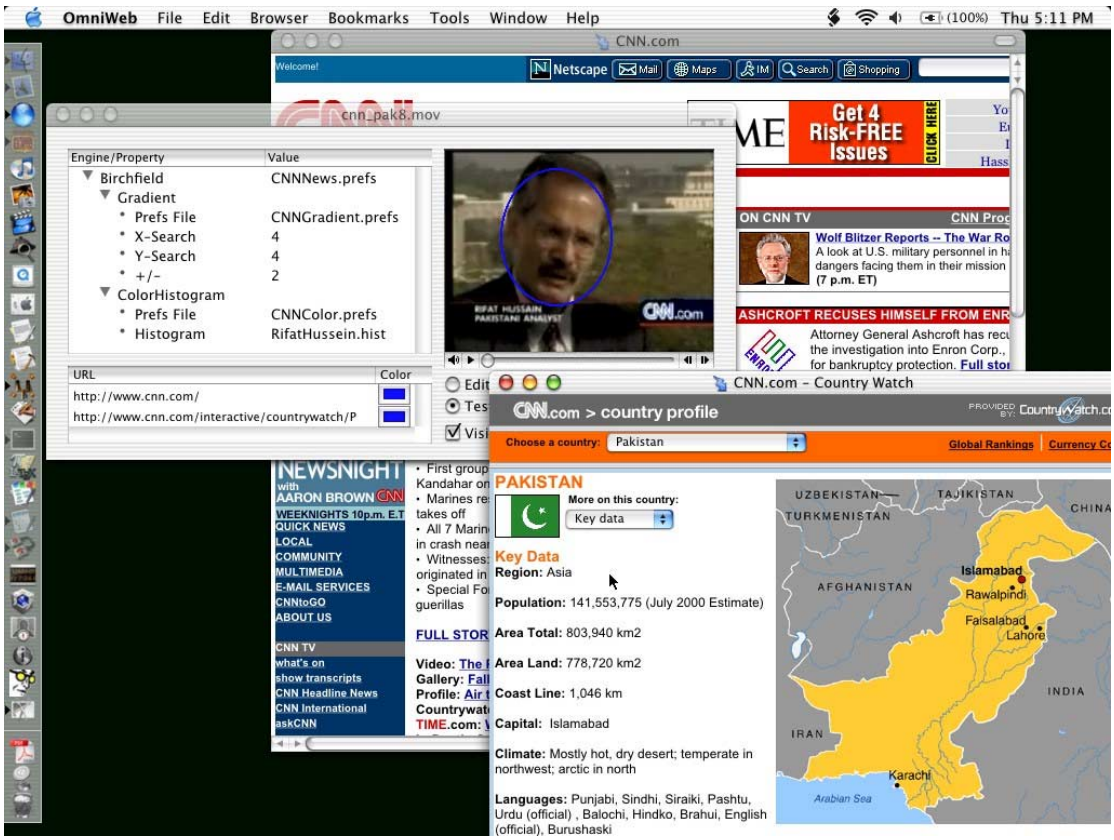

Figure 2. Two links on one tracked object, showing targets displayed

3. USER INTERFACE EXAMPLES

OvalTine is set up to allow link markup of stored video as an editing task. Figure 1 shows the video display window, the properties window, and the URLs window. The video display window presents the video data and shows the tracked objects as (optionally) outlined ovals. For real-time applications, such as video conferencing discussed previously [1], the playback controls would be inoperative. For markup of stored video, the user can do the standard *start*, *stop*, *pause*, and *slider* frame selection operations on the video stream.

The tracking properties window allows the user to select the type(s) of tracking algorithms to apply to the video frames. OvalTine's architecture has been structured with well-known design patterns [2], allowing multiple tracking algorithms to be chained together and applied in sequence. A user can even apply a different chain of trackers to each different oval if desired; OvalTine spawns a separate tracking thread for each oval. Some trackers work better than others in varying images; the selection of specific trackers to use depends on image properties such as color variability, background complexity, object motion, texture, etc.

Once a link anchor (oval) has been established in the video window, the user can associate one or more URLs with that anchor to be targets of the link(s). These URLs show up in the URL window in the lower left. Figure 1 shows one URL for each oval, and the association is made by the color of the tag.

Editing progresses as described in the previous section. The user selects a starting frame with the slider controls, and then designates one or more objects to be tracked. The video is started with the "track" button, and the tracker chain for each oval causes the link anchors to follow the objects as they move in the video frame. The sprite infrastructure captures the layout information needed to maintain the link anchors in association with the video data. At any point the user may pause the video, add or delete ovals, and continue with tracking.

URLs can be added to the tracked objects at any point, either during tracking, or during playback editing of the marked-up video data. Though not shown, ovals can be linked to any first-class Web data, including another OvalTine video clip. There is also a "lost" link palette (not shown) that collects the URLs associated with objects that are being tracked, but move out of the video window. Any URLs associated with such an object are taken out of the URL window and saved in the "lost" list. This is a convenience that makes it easier for the user to re-associate these URLs if the tracked object should reappear in the video window and need to be tracked again.

Figure 2 shows OvalTine in use while viewing a marked-up video stream. Here we see a CNN clip where the face of the reported has been annotated with 2 different links. One link is to the CNN home page (seen displayed in the background). The other link is to an article and map on Pakistan, which is the topic of the video; this page is shown in the foreground.

There are several ways to manage multiple links on each oval. One is to simply left-click the mouse on the oval in the video window. This will cause the top link in its URL list (shown in the URL window, lower left) to be followed and displayed. Another method is to right-click on the oval and a pop-up menu will present all the URLs for that oval for selection of one to follow. A third method is to directly click on the desired URL in the URL window; these URLs are only displayed in the window while their associated ovals are in the video window. In the editing mode (as opposed to playback) the order of URLs in the URL window for each oval can be rearranged via click-and-drag operations so that new defaults can be established.

Finally, there is a mode the user can select during editing that causes *all* the associated links on an object to be followed concurrently when an oval is left-clicked. Each link target will be displayed in a separate browser window. This mode is related to the concurrent and synchronized browsing possible with the multi-head/multi-tail links found in MMM [5,6] and first studied in Trellis [3, 4].

4. OVALTINE SYSTEM ARCHITECTURE

After the initial prototype was operating, we spent time re-architecting and re-implementing OvalTine to make the tool more adaptable to new tracking approaches and to adhere to good design patterns [2]. The resulting second prototype is robust and extensible.

We initiated a redesign of the code and functionality in January of 2001, and finished in late March of the same year, converting approximately 15,000 lines of C++ code of varying quality in some rather extreme ways. Design patterns were critical in the redesign and re-implementation phase, and used throughout the system.

OvalTine started as a conglomeration of two very different sets of code. One, called *MediaConf,* is a demonstration video conferencing app from Silicon Graphics that runs on their line of O2 workstations. The other was a half-completed (and quite broken) set of code

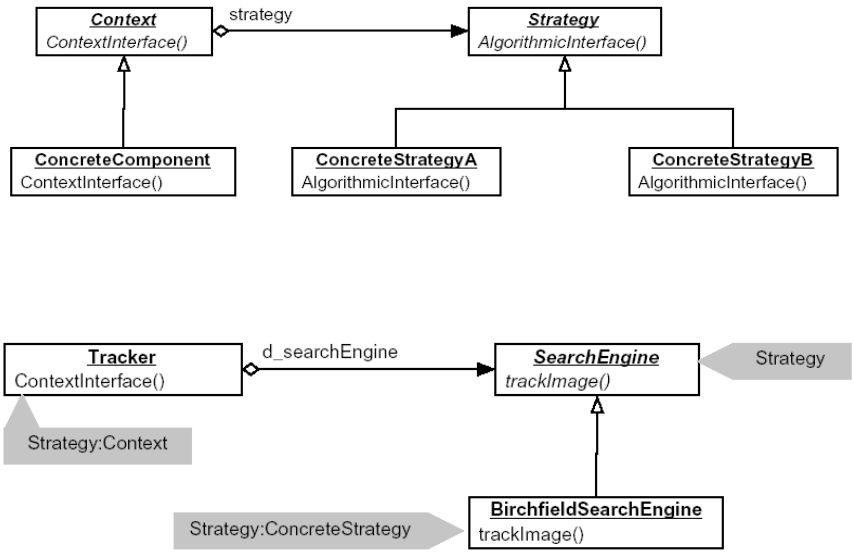

Figure 3: Strategy pattern for extensible trackers

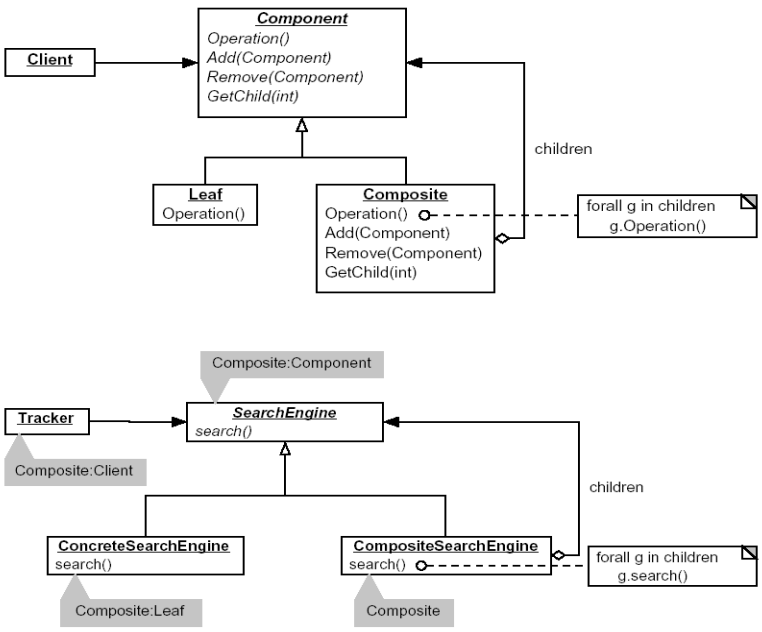

Figure 4: Composite pattern for chainable tracker

from Stanford University illustrating a way of tracking heads and faces in real-time video on a specific proprietary video card on the Windows platform. The work involved in taking these two utterly different bodies of code and making them work together was tremendous, not the least of which was involved in just making them work at all independently.

Once we had the code working well enough for demos, we decided to re-factor and rewrite the system, since it utterly failed to meet our original vision. What we wanted was a flexible research platform that would run on multiple OSs, would handle video feeds from live cameras, stored files, or from a network feed, would allow different tracking algorithms to be experimented with in a freeform manner, and would be extremely simple to use and develop with. Instead what we had was a system that was incredibly fragile, had one hard-coded algorithm for tracking, ran only on the SGI O2 platform (and then only on certain OS versions), and was limited to a live local video feed.

The reworking effort was done from the viewpoint of using design patterns as a springboard for system architecture design, as well as guiding implementation. These patterns gave the architecture the following properties:

- Extensible algorithm system for object tracking in the video stream... algorithms should be easily added to the pool for selection, and should work interchangeably
- Algorithms should be chainable for efficient pre and post processing
- Dynamic selection of algorithms, preferably on the fly during execution, but a simple selection process for configuration at launch is acceptable
- The video tracking library should have a simple external API
- The video tracking library should be completely separated from any GUI, front end app, or IO, except as through the above API
- The video tracking library should be cross-platform

While each requirement created specific patterns in our new architecture, the first two (extensible trackers, chainable trackers) are of particular interest to a hypervideo audience. We therefore examine each property in more detail in the following sections.

4.1 Extensible tracking algorithms

One of the most critical components needed in the redesign was to allow new tracking algorithms to be added to the system as imaging researchers develop them. This exactly corresponds to the *Strategy* pattern, whose intent is given in the patter literature as:

> "Define a family of algorithms, encapsulate each one, and make them interchangeable."

As new concepts in identifying object in an image are developed, OvalTine can be extended without being abandoned or re-written. Extensions can be done by hypervideo designers with OO programming skills; it involves writing new classes with the same basic form as the existing trackers, and adding their compiled for to the class libraries.

Figure 3 shows the full design pattern diagrams for the *Strategy* pattern that give OvalTine extensibility in tracker algorithms. By looking at the structure of the pattern in these diagrams, we can quickly start identifying the basic concept classes to fulfill the roles: *Context* is our Tracker system, *Strategy* is an abstract base class corresponding to our idea of a SearchEngine, and the *ConcreteStrategy* roles will be filled by the actual algorithms. In our initial situation, we have one algorithm, which we called *BirchfieldSearchEngine*, after the author of the describing paper.

4.2 Chainable trackers

The *BirchfieldSearchEngine* is actually a hybrid tracking algorithm that uses two separate approaches: a gradient based edge detection algorithm to find an oval shape corresponding to a head; and a color histogram approach that detects the face itself through a visual field analysis. We wanted to compare the performance of the *BirchfieldSearchEngine* against the individual algorithms comprising it, but unfortunately the conglomerated algorithm was originally written as a monolithic piece of code. We considered writing two completely new engines, to recreate the two algorithms, but decided that this was a poor way of reusing the concepts, since we would be reusing no code.

What was needed was a way to have *BirchfieldSearchEngine* reuse the code from the other engines, instead of attempting to reuse the code from the hybrid. This would necessitate a *SearchEngine* class calling two other *SearchEngines*, and using the results to create a third set of tracking feedback. This meets the criteria for applying the *Composite* pattern:

> "Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly."

Figure 4 shows the design pattern diagrams for the *Composite* we use to give OvalTine chainable trackers. The structure of the pattern is straightforward, where the *Client* role is naturally again that of *Tracker*, the *Component* is a *SearchEngine*, the class fulfilling the

Composite role we will call blandly enough *CompositeSearchEngine*, and the *Leaf* role will be our various *SearchEngines* that do the actual tracking.

This feature allows OvalTine to track objects by passing image information from a frame though several different tracking algorithms in series. Each has the opportunity to decide if an object appears in the image or not, either using the data on its own or by using the conclusions of a prior tracker.

We also spawn a new thread for each object being tracked. This allows a different chain of trackers to be applied to each different object, if such is desired. It might, for example, be easier to track faces with one combination of algorithms, and easier to track rectilinear objects (books, signs, etc.) with a different set of algorithms. OvalTine will support this variance.

5. CONCLUSIONS

OvalTine was previously a system for tracking objects in real-time video streams (video conferences) and allowing hyperlink anchors to be associated with these tracked objects. We have described here the further use of OvalTine for automatically tracking objects in stored video streams, thereby allowing automated markup of archived video data with hyperlinks. Extending OvalTine from real-time to archived video required a link layer to capture and maintain the link anchors are they are tracked from frame to frame; in the previous work, this information was available at each instant, but lost as one frame progressed to the next.

We created such a link layer on top of the basic OvalTine tracker using the Sprites of Apple's QuickTime standard. We also re-built OvalTine using good OO design patterns to allow hypervideo creators to extend the basic object tracking algorithms with their own work.

Our work has concentrated on producing the software needed to automate link productions and capturing this information for playback. We have not yet concentrated on new tracking algorithms for tracking objects other than ovoids with color histograms similar to faces. Work remains to be done to extend our library of trackers to full generality.

OvalTine is a demonstratable system and can be shown in operation at the conference.

6. REFERENCES

[1] Smith, J., D. Stotts, and S.-U. Kum, "An Orthogonal Taxonomy for Hyperlink Anchor Generation in Video Streams using OvalTine," *Proc. of Hypertext 2000 (ACM)*, May, 2000, San Antonio, Texas, pp. 11-18.

[2] Gamma, Helm, Johnson, and Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, 1995.

[3] Stotts, P.D., and R. Furuta, "Petri Net Based Hypertext: Document Structure with Browsing Semantics," *ACM Trans. on Information Systems* (ACM), vol. 7, no. 1, January 1989, pp. 3-29.

[4] Furuta, R., and P. D. Stotts, "Programmable Browsing Semantics in Trellis," *Proc. of Hypertext '89 (ACM)*, Pittsburgh, November 1989, pp. 27-42.

[5] Capps, M., B. Ladd, D. Stotts, "Enhanced Graph Models in the Web: Multi-client, Multi-head, Multi-tail Browsing," Computer Networks and ISDN Systems, vol. 28 (Proc. of the 5th WWW Conf., May 6-10, 1996, Paris), pp. 1105-1112.

[6] Ladd, B., M. Capps, D. Stotts, and R. Furuta, "Multi-head/Multi-tail Mosaic: Adding Parallel Automata Semantics to the Web," *World Wide Web Journal*, O'Reilly and Associates Inc., vol. 1 (*Proc. of the 4th International WWW Conference*, Boston, December 11-14, 1995), pp. 433-440.

[7] Sawhney, N., D. Balcom, and I. Smith, "HyperCafe: Narrative and Aesthetic Properties of Hypervideo", Hypertext '96 Proceedings, ACM, Washington, D.C., 1996, pp. 1-10.

[8] Francisco-Revilla, L., "A Picture of Hypervideo Today", http://www.csdl.tamu.edu/~l0f0954/academic/cpsc610/p-1.htm, 1998.

[9] Apple Computer, "Introduction to Wired Movies, Sprites, and the Sprite Toolbox", http://developer.apple.com/techpubs/quicktime/qtdevdocs/REF/refWiredIntro.htm

[10] Bernstein, M., "An apprentice that discovers hypertext links", Hypertext: Concepts, systems and applications: Proceedings of the European conference on Hypertext, INRIA, France, 1990, pp. 212-223.

[11] Bernstein, M., J. D. Bolter, M. Joyce, and E. Mylonas, "Architectures for Volatile Hypertext", Hypertext '91 Proceedings, ACM, San Antonio, TX, 1991, pp. 243-260.

[12] Boissière, G., "Automatic creation of hypervideo news libraries for the World Wide Web", Hypertext '98 Proceedings, ACM, Pittsburgh. PA, 1998.

[13] Glushko, R.J., "Design issues for multi-document hypertexts", Hypertext '89 Proceedings, ACM, Pittsburgh, PA, 1989, pp. 51-60.

[14] Hirata, K., Y. Hara, H. Takano, and S. Kawasaki, "Contentoriented Integration in Hypermedia Systems", Hypertext '96 Proceedings, ACM, Bethesda, MD, 1996, pp. 11-21.

[15] Ip, H. H., and S. Chan, "Hypertext-Assisted Video Indexing and Content-based Retrieval", Hypertext '97 Proceedings, ACM, Southhampton, UK, 1997, pp. 232-233.

[16] Kendall, R., "Hypertext dynamics in *A Life Set for Two"*, Hypertext '96 Proceedings, ACM, Bethesda, MD, 1996, pp. 74-84.

[17] Liestøl, Gunnar, "Aesthetic and Rhetorical Aspects of Linking Video in Hypermedia", Hypertext '94 Proceedings, ACM, 1994, pp. 217-223.

[18]. Nürnberg, P. J., J. J. Leggett, E. R. Schneider, "As We Should Have Thought", Hypertext '97 Proceedings, ACM, Southhampton, UK, 1997.

[19] Stotts, P. D., and R. Furuta, "Dynamic adaptation of hypertext structure", Hypertext '91 Proceedings, ACM, San Antonio TX, 1991, pp. 219-232.

[20] Zhang, H.J., et al., "Automatic Parsing and Indexing of News Video", *Multimedia Systems*, 2 (6), pp. 256-266, 1995.

[21] Zhang, H.J., C.Y. Low, S.W. Smoliar, and J.H. Wu, "Video Parsing, Retrival and Browsing: An Integrated and Content-Based Solution", Multimedia '95 Proceedings, ACM, 1995, pp. 15-24.

[22] Multimedia Systems Lab at IISc, "Object tracking and hypervideo", http://serc204a.serc.iisc.ernet.in/research/track.htm