Embedded Video in Hypermedia Documents: Supporting Integration and Adaptive Control

DICK C.A. BULTERMAN
CWI: Centrum voor Wiskunde en Informatica

As the availability of digital video becomes commonplace, a shift in application focus will occur from merely *accessing* video as an independent data stream to *embedding* video with other multimedia data types into coordinated hypermedia presentations. The migration to embedded video will present new demands on applications and support environments: processing of any one piece of video data will depend on how that data relates to other data streams active within the same presentation. This article describes presentation, synchronization and interaction control issues for manipulating embedded video. First, we describe the requirements for embedded video, contrasted against other forms of video use. Next we consider mechanisms for describing and implementing the behavior of embedded video segments relative to other data items in a document; these relationships form the basis of implementing cooperative control among the events in a presentation. Finally, we consider extending the possibilities for tailoring embedded video to the characteristics of the local runtime environment; this forms the basis for adaptive, application-level quality of service control of a presentation. In all cases, we describe a mechanism to externalize the behavior of hypermedia presentations containing resource intensive data requirements so that effective control can be implemented by low-level system facilities based on application-specific requirements. We present our results in terms of the CMIFed authoring/presentation system.

Categories and Subject Descriptors: I.7.2. Multi/Mixed Media, H.5.1. Video Processing

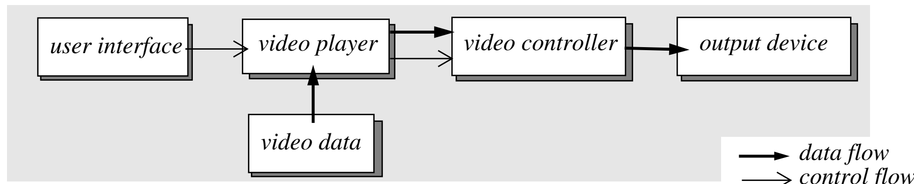
General Terms: Multimedia, Hypermedia Documents, Media Synchronization, Video Presentation

Additional Key Words and Phrases: Adaptive Control

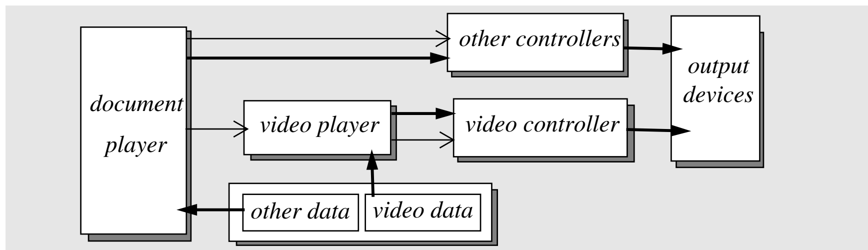
1. INTRODUCTION

During the past decade, the presentation of video data on desktop computers has grown from a curiosity to a common facility. Most current workstations can provide at least rudimentary video display support (with one even supplying a camera as a part of its standard configuration) and many PC third-party vendors offer interfaces that allow even low-end systems to display continuous video from analog inputs. While these developments may not be technologically startling—after all, a typical computer *is* a television set that has had its tuner and antenna replaced by a computer's I/O bus—they do provide perhaps the most dramatic example of the shift to new data types that has occurred under the name of *multimedia computing*.

In the current generation of video presentation systems, video is either processed as a stand-alone data type or it is processed as an independent stream that is attached to other related media streams in a *multimedia document* [12], [13]. Processing for stand-alone video is illustrated in Fig. 1(a): a special-purpose *video player* takes self-contained video data from a local disk, a CD-ROM, a network connection, etc., and displays it via a video device controller subsystem. The device controller may



(a) stand-alone video



(b) attached video

Figure 1: Generic multimedia document processing architecture.

simply map bits onto a display, or it may be a complex processor that allows video decompression and image transformations. In both cases, the functions/freedom provided by the device will determine the type of control—such as *start/stop* and sometimes *pause/rewind/fast-forward*—that the user can exert over the presentation through the video player. Fig. 1(b) illustrates the approach most commonly used for attached video [25]: the activation of the video is initiated by a *document player* (which also directly or indirectly controls other data streams) but the actual video control remains with an external video player. This approach has the advantage of localizing data-dependent control in the video player and overall presentation control in the document player; given the lack of standardization of video presentation hardware, this pragmatic approach makes video presentation possible if an appropriate video player exists on a presentation platform.

In both stand-alone and attached video processing, the emphasis of current-generation systems has been on simply accessing video data. The scarcity of existing video material and the relatively low delivery speed of networked video has limited the potential for more active video processing. This will change as processor speed and I/O and network bandwidth increase, and as storage devices provide higher density—all allowing a new generation of browsers and editors to make video data more readily available. As video becomes commonplace, one can expect a shift toward integrating video fragments with other information streams into hypermedia documents, with the coordinated presentation of each stream depending on content-based associations among the streams. We call this type of use *embedded video.*

The coincident use of embedded video with other data items will require significant document-specific control over activation and inter-stream data synchronization. This poses a fundamental control problem in current approaches to supporting video data: at present, any two streams can be started at document-defined relative

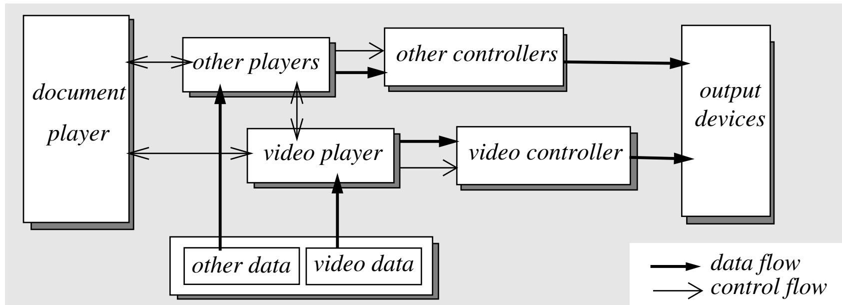


Figure 2: A cooperative-control approach to supportive embedded video.

times, but other control events cannot be fed back to the document player because processing in individual subsystems have no relationship to each other. As a consequence, new control models are required that allow the coordinated presentation of embedded data.

This article describes the approach used in the CWI CMIFed presentation environment for supporting embedded video in networked hypermedia documents. Our emphasis is not on "how do you get video to work" but rather: "what do you do with video once it does work." Our approach is to *externalize* the document-dependent interactions among media streams. That is, we allow the author to specify how components interact explicitly at the document level, rather than using implicit interactions defined in internal data definitions. In this way, light-weight, media-specific control components can be defined that coordinate the interaction among various media streams. Rather than migrating the data and format control for video (and other speciality data types) backward into the document player, we allow the implementation of inter-stream control decisions to be migrated forward to the media players. We are able to do this because our document structure makes presentation, synchronization and interaction information explicitly available for use by downstream processing components. In this way, each media player can coordinate its activities with other streams while retaining the localization of device dependencies within relatively small and isolated components of the presentation architecture. The structure of our approach is illustrated in Fig. 2.

Our discussion is structured as follows. Section 2 reviews background information on the characteristics and implications of using embedded video in hypermedia documents. Section 3 describes how we currently externalize the traditionally internalized aspects of a document's data interactions to isolate low-level control of video flow while retaining integration with other data streams in the document. Section 4 discusses extended approaches to providing adaptive support for embedded video for increased presentation and interaction control in heterogeneous environments. Sections 3 and 4 include examples from the CMIFed player; the CMIF document format is reviewed briefly in the Appendix.

2. EMBEDDED VIDEO: CHARACTERISTICS AND IMPLICATIONS

In order to introduce the problems of supporting embedded video, this section reviews general network-based video processing and then contrasts the characteristics of embedded video with those of other video applications. We begin with a set of examples of embedded video use in hypermedia documents.

2.1 Examples of Embedded Video Use

Embedded video distinguishes itself from stand-alone or attached video in the following ways: (1) the presentation of video data will not be the central purpose of the application or document using embedded video; (2) the presentation scope of an embedded video fragment may vary, depending on the runtime behavior of other streams in the presentation; (3) the video that *is* shown will be of short duration (on the order of seconds); and (4) the content-based integration of embedded video with other data streams will required significant document-specific influence over process scheduling and resource allocation.

In order to make our presentation in this paper clear, we define three examples that illustrate the use of embedded video in hypermedia documents. We return to these examples throughout the paper. The examples are:

- *electronic encyclopedia:* in a section describing bird life in the city, a series of videos are shown of birds in different places, with the duration of any one fragment dependent on the length of audio and text data that provide the basic descriptions of our feathered friends; this use of video supports and amplifies other information in the section rather than providing self-contained presentations
- *personal communication:* in the body of an electronic mail message, a fragment of embedded video is presented with an accompanying but independent audio block *if* the user selects the pair by following a hyperlink from the body of the text; this use of embedded video depends on the behavior of the user when the message is read as well as the length of the audio fragment
- *multimedia cookbook:* as part of a recipe for a hot-curry-and-chocolate cheesecake, a short video and (independent) captions text describing the batter preparation technique and required consistency is placed between a text block containing ingredients and a still image, showing the results; here the video is the main information carrier of information.

In each example, the current approach of relying only on low-level, application-independent management of video data is insufficient to meet the needs of embedded video presentations.

2.2 Delivering and Controlling Networked Video

In conventional processing of video data, a *video transaction* is initiated between a server and a client. Once the transaction starts, synchronization and resource control is usually implemented by low-level client and server services without intervention of application-level code [8], [19]. System resources are reserved in advance and

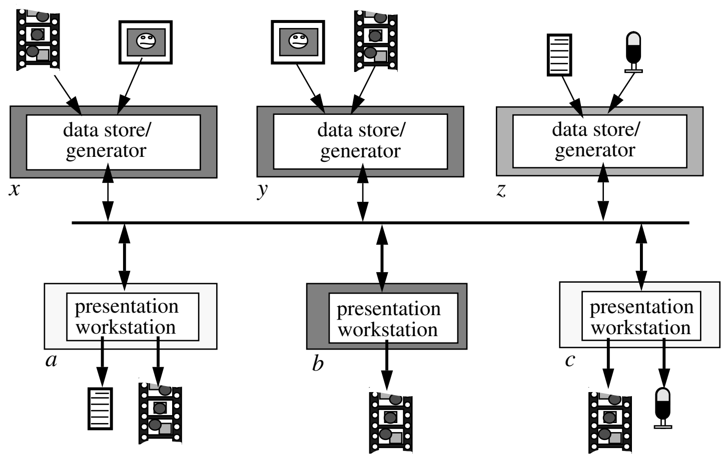


Figure 3: Networked embedded video.

low-level components within the operating systems of each host control the transfer of data to ensure correct presentation synchronization [32]. A problem arises when demand for resources exceeds resource availability. Consider the situation in Fig. 3. If more clients wanted to access server *y* than it could handle, or if the network were being saturated by competing requests for other video and non-video services, the environment might not be able to provide correctly synchronized video data. A common solution to this problem is the definition of an *admission control* protocol for network and server resources [10], [34], [35]. Admission control protocols are typically biased toward existing clients in the network. If a new client requests service, it is only honored if sufficient resources are available; if not, the new request is either (temporarily) denied or a negotiation process is initiated that results in a new request for less-intensive use of resources. (That is, a lower frame rate could be requested, yielding lower quality video.)

In the case of embedded video, the resource control problem is more complex because of the application-dependent nature of video use. Table 1 summarizes five properties that have an influence on resource control relative to embedded video and three traditional video-based applications: *video-on-demand* [19], [22], *video multicasting* [20], and *video-based CSCW* [11], [31]. We discuss each property briefly in the following paragraphs.

Role of video in the document. The role of embedded video is highly document dependent. In our encyclopedia example, video plays a supporting role in that a document author may define it to be optional. In the personal communication example, the video presentation is conditional: it is used only if the user selects it by following a hyperlink in the document. In the cookbook example, the video plays a more critical role. Rather than highlighting or supplementing other data, the video is used as the central information carrier.

This application dependency is typically not present in other uses of video. In video-on-demand and multicast video, the video component is central—without it, the application cannot function. In CSCW applications, video is important but not central—video is often a common backdrop against which other types of information (such as pictures or shared whiteboards) are presented, some of which may play a more important role in the exchange of information than the video [11].

Structure of the video transaction. In a document using multiple embedded video sequences, each segment will be a pre-defined object fetched from a single source and presented at a single destination. Since all of the segments will not, in the general case, come from the same server, aggregate behavior will approximate a multiple source and single destination structure.

Video-on-demand is typically single source to single destination (although some reuse of coincident requests may be possible [7]), video multicasts are typically single source to multiple destinations (with selection occurring at the receiver), while CSCW applications are by definition multiple sources to multiple destinations. Video-on-demand data is nearly always pre-defined; video multicasts may be either pre-defined or 'live'; while video CSCW applications will contain a mix, but the video component will nearly always be live.

Transaction duration. The length of each video segment will be determined by the document. In general, the incidental nature of embedded video sequences suggests that each fragment will be relatively short. (This maximizes reusability, decreases transfer overhead and minimizes production costs.) In our experience, the typical segment length is measured in seconds.

For video-on-demand, a single transaction will match the effective length of the video item being accessed; for feature-length films, this will be on the order of hours, although it can also be shorter in data/news browsing applications [19], [22]. Video multicasts may vary, but their length will typically also be on the order of (large fractions of) hours. CSCW video transmissions may also vary in length, but will typically be measured in blocks of minutes.

Transaction initiation. Each embedded fragment transaction is initiated by the

Application: / ***Properties:***	*embedded video*	*video on demand*	*video multicast*	*video-based CSCW*
role of video	**application dependent**	central	central	variable
transfer structure	***source:*** **multiple** ***destination:*** **single**	single single	single multiple	multiple multiple
duration	**O(seconds)**	O(hours)	O(hour)	O(minutes)
initiation	**at destination**	at destination	at source	consensus
video-based interaction	**substantial (video navigation)**	minimal (VCR)	minimal (duplex)	minimal (start/point)

Table 1: Characteristics of important applications using video data

destination, although the actual transfer may be denied if the infrastructure cannot fetch the data "on time," depending on the nature of the data's use in the presentation. The scheduling of the transaction may depend on other data streams active in the presentation.

Video-on-demand, by definition, is scheduled at the request of the user, although the actual transaction will start only when resources are made available across the infrastructure. In video multicasting, the start of the transaction is governed by the source of the multicast—the nature of this class is such that events are usually pre-scheduled based on an external time reference. For CSCW, the physical presence of each participant during the session implies the need for external scheduling.

Interaction requirements. Interaction with embedded video takes two forms: interaction within the video segment and interaction with the portion of the document that includes the segment. In the first case, hypertext-like structures such as links and anchors [12] must be integrated within the video and means must be found to activating these during playback [18], [33]. The second case involves defining the behavior of the video stream in relation to the behavior of the streams within which it is embedded. For example, if a video is shown together with a text fragment containing the source or destination of a hyperlink, the effect of following the link will have ramifications for the video sequence as well. (We return to these topics in section 3.)

Video-on-demand and video multiplexing require only limited interaction support, typically at the level of VCR-like controls [7]. Non-sequential access (such as hyperlinks within the video source) is typically unsupported. In CSCW applications, interaction will dominate, although the types of interactions resemble the social conventions of conversation rather than having each user selectively navigate through (video) data sets simultaneously.

2.3 Implications of Embedded Video Use.

The video-rate resource needs of embedded video, coupled with its variable importance, short duration, and multi-source nature presents the support environment at both the source and destination of a video transaction with a series of constraints that are impossible to manage outside the context of the document using the embedded data. The economy of scale and long-term connection characteristics of conventional video processing all but disappear for embedded video use.

While the resources required for delivery of an individual fragment of embedded video resemble a short-term video-on-demand session, the increased interaction needs of embedded video and the need to coordinate delivery with other (independent) data streams will make actual resource reservation difficult throughout the delivery pipeline. Consider the situation in Fig. 1 where workstation *a* is receiving video and text data from servers *x* and *z*, respectively, and workstation *c* is receiving video and audio data from servers *y* and *z*. In the case of workstation *a*, where text is being combined with video, exact synchronization of the data streams may not be required; for a given piece of text, the video data may even be optional—if the video cannot be delivered within document-specified constraints, it can be skipped without loss of overall document content. In contrast, if workstation *c* requires lip-synchronous data presentation, the needs will be more critical. In both cases, the manner is which the embedded video is used rather than the nature of (embedded) video itself will determine the desired system behavior.

3. DEFINING AND IMPLEMENTING EMBEDDED VIDEO BEHAVIOR

In this section, we describe how the scheduling and synchronization of video and other multimedia events can be coordinated by using structural and logical relationships among the data items that are saved with a document's description. We begin by describing the architecture of the CMIFed playback environment, developed at CWI as part of our research effort in authoring and document presentation support environments [30]. We then describe how we encode and partition individual data items and conclude with a description of how we control data object interactions. Several examples will be given in terms of the CMIFed authoring and playback environment; a summary of the CMIF document structure is given in the Appendix.

3.1 The CMIFed Document Presentation Architecture

Designing and defining hypermedia documents is a non-trivial task. Data items must be created or located in external databases and 'pleasing' presentations need to be defined that run on heterogeneous presentation platforms at various times of the day, under changing network loads. Of all of the tasks associated with building hypermedia documents, the most time consuming is the creation of the data items and the definition of the basic document structure. One of our primary goals has been to ease the long-term authoring burden by allowing documents to be defined as abstract document specifications in a relatively portable format. In this way, a single document specification can be authored once, and then transformed to meet the needs of the users and the constraints of the presentation environment at runtime.

In defining an architecture to support documents in a heterogeneous environment, we retain structure information in the document and used it to guide presentation playback. We externalize document behavior so that it can be used to influence network-wide presentation scheduling and resource control while still coupling the various media streams to allow presentation synchronization. This is in contrast to the approach used in HTML-formatted documents [15], where media types like audio and video (where supported) are processed as separate entities that cannot be synchronized with other events active at the same time.

Fig. 4 illustrates the general runtime architecture of the CMIFed environment. The CMIFed authoring system produces a CMIF hypermedia document. The document is a data structure that is interpreted by the CMIFed player. The player consists of four major components: the *interpreter* control program, the *scheduler*, the *link processor* and a set of *channel threads*. (The notion of a 'channel' is discussed shortly.) The interpreter, scheduler and link processor are Python [29] programs, while the media threads are machine-dependent routines written in Python or C.

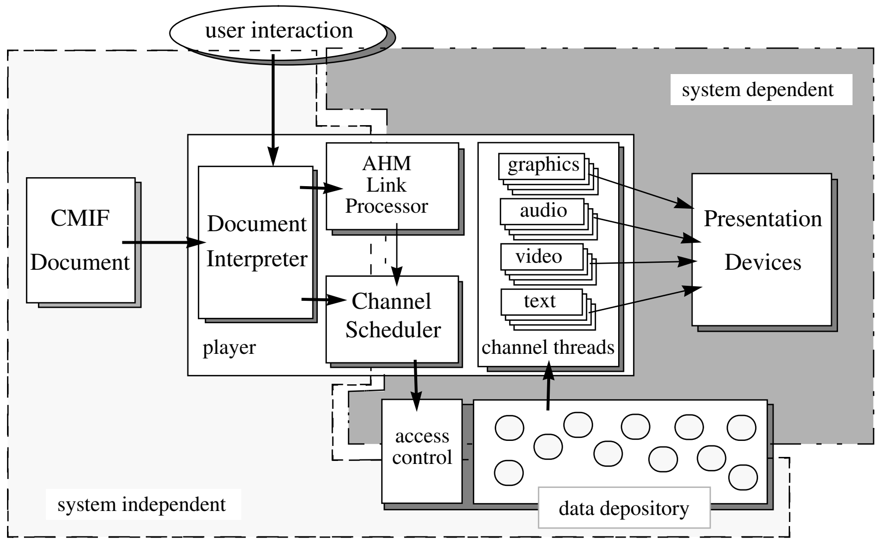


Figure 4: The CMIFed player architecture.

The player was designed to translate a system-independent description into an executable version of the document. The document itself is not a script, but a data structure that describes the components and interactions of an application. During execution, the interpreter dispatches portions of the document to the scheduler, under control of the user and under influence of the link processor. (The link processor implements the hypermedia control based on the Amsterdam Hypermedia Model (AHM) [14].) The scheduler resolves synchronization constraints, brings media items from the data depository to the various media threads, pre-fetching information and caching results, as appropriate to the document's content. The organization of the data depository and the activities of the scheduler are discussed in more detail below.

The document defines a time and resource description of the presentation and it contains directives that externalize the structure of the application. Externalizing application structure not only enables the possibility of improved document scheduling, it also facilitates the development of reusable applications. Authoring complex presentations, just as writing complex computer programs, is a task that should be designed to be portable—even across presentation platform facilities. This requires the inclusion of sufficient information to reproduce the presentation on different environments and the encoding of this representation in a platform-independent form.

The CMIF document—and the CMIFed player—use our own private document format. While formats such as MHEG [17],[21] or HyTime [16],[23] have been designed that address some portability issues, we explicitly chose not to use these for two reasons: first, the core of our project pre-dates any available versions of these formats, and second, our intention is not to produce distributable documents, but rather to study authoring and presentation support that could be integrated into future systems and future standards. In this last sense, it is useful to maintain a private document format, although our authoring interface is more general than our system alone [3]. We are currently integrating support for other formats in related work [6].

3.2 Describing data fragments

Intuitively, each embedded video fragment is an atomic unit that is fetched by the application as a single entity and then integrated with other data streams to form a composite presentation. We call such a fragment a *data block*.[1] Each data block may have a simplex or complex internal structure (that is, it may consist of only one data type or some combination of types "glued together" and accessed as a unit). Examples of video fragments are: the cookbook, encyclopedia and personal communication videos described in section 2; a news item (with or without soundtrack) from a video broadcast; or a few seconds of video material contained in personal or public archives. The unifying characteristic of all these segments is their potential for reuse in a wide variety of applications.

In terms of the player, we are not particularly interested how data are internally organized; we assume low-level activation routines will be available to (quickly) access and deliver data. What we are concerned with is de-coupling data from any application dependencies. For instance, in our cookbook example, the fact that the video of the cheesecake batter is a required part of a presentation cannot be encoded as part of the data or the database, since the sequence may be reused in many different applications where it may or may not be optional. It may also be used several times in the same application, each time with a different purpose and presentation priority.

In order to promote reusability, we keep the external properties of each data fragment separately from the data. These properties include information on the resource requirements, data format and semantic content of the embedded data. This allows searching for and analysis of video segments to occur without first having to access the data sets. Our partitioning of attributes is given in Fig. 5: the *data block*, the *data block descriptor*, and the *event descriptor* [4]. In the example, a 15.3 second video fragment of a hot-curry-and-chocolate cheesecake mixture is stored as an MPEG-1 data block. The data block descriptor contains resource requirements and a contents field, which is used by our authoring system to select this particular block. (It may also be useful in defining dynamic hyperlinks within the presentation.) The use of one instance of a particular block within a hypermedia document is described by the event descriptor—in this case, indicating that only a portion of the complete data block will be used in the presentation (the part active from t_a to t_b), that the presentation should be synchronized with other events and that the presentation of the video is a non-optional part of the document's presentation.

In the example, we see a single encoding of the data item. A more general approach is to control access to an *adaptive information object* (AIO), which can

1. Do not confuse this high-level data abstraction with low-level structures such as disk blocks.

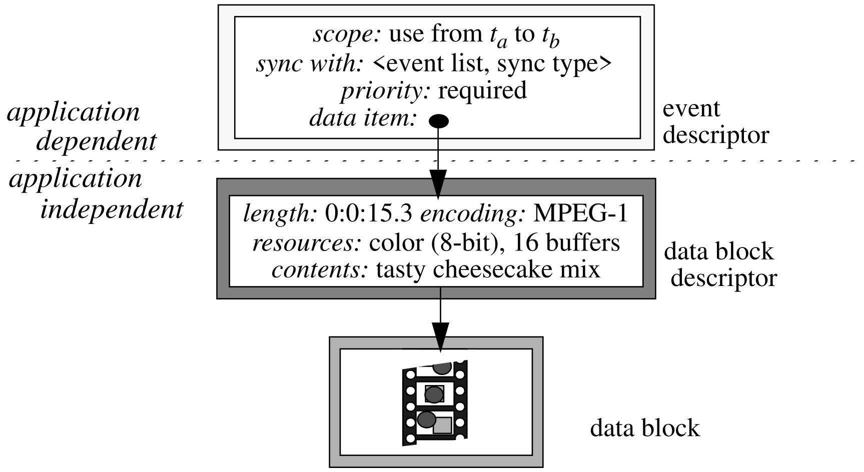


Figure 5: A {data block | data block descriptor | event descriptor} hierarchy.

select a representation based on the runtime resources available. We return to this organization in section 4.

3.3 Embedding events within a presentation

The CMIF document defines a collection of events that (conditionally) make up a presentation, as well as three types of constraints across those events. *Presentation constraints* define the placement of events on the output media, *synchronization constraints* "guarantee" that placement will occur at the desired time, and *interaction constraints* determine the collection of events that actually get presented.

Presentation constraints. The playback environment must present data at a location and a size defined by the author, under control of the user. Consider the multi-lingual electronic encyclopedia fragment shown in Fig. 6. Here we see a presentation that consists of two streams of headline text (of which only one is visible), a video stream, two formatted text streams containing captions (one in Dutch, one in English), a block of formatted text in English, three audio output

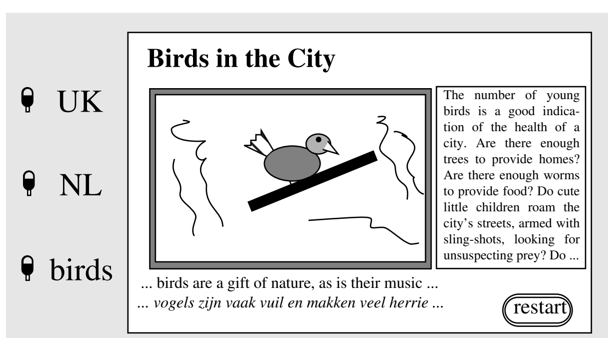


Figure 6: The presentation of a document using embedded video

streams (two with spoken text to match the captions and one sound-effects track), plus a general control interface. Each of the data streams has its own presentation requirements that depend on the characteristics of the data encoding and the semantics of the message. The presentation requirements for this document include the placement of information (involving the allocation of screen space and audio channels) and the relative ordering of information. In case of conflicts, decisions on screen resource use must be made by the presentation system. For example, if both headline streams are allocated the same space on the screen, then only one stream is visible; the stream is selected by system default or by explicit user action. Alternatively, the two formatted text streams containing English and Dutch captions could be defined so that both can be displayed together, meaning that the presentation system would need to determine exact runtime placement to avoid text overlap. (These options are specified by the author and implemented by the player.) Note that if the captions have a content-based relationship to the video and/or audio streams, the relative presentation time of each stream also becomes important.

In the CMIF environment, one class of presentation constraints is defined in the event descriptor (as illustrated in Fig. 5) and a second class is defined via an *abstract media channel* (or simply a *channel*). As we will see, the presentation constraints in the channel are used as a global mechanism for grouping attributes of several events that share similar properties, where the event presentation constraints can be used to over-ride those defined for the channel.

While the channel is typically of one media type, there may be several channels per media type in the document. The mapping of channels to physical system resources is done at runtime by channel threads optimized for use on a local system. Fig. 7 shows a channel partitioning of the illustration in Fig. 6. Ten channels are used to encode the logical portions of the document. Each channel definition con-

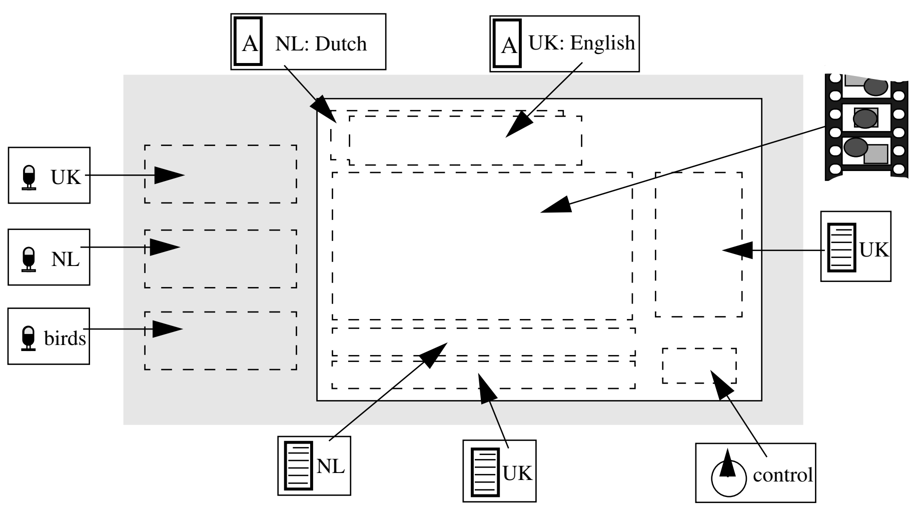


Figure 7: The use of channels in Fig. 6.

tains information on the location, sizing and priority of all of the events on that channel. During presentation, the scheduler uses channel information to determine how channel-related events are to be processed. The user is also able to exhibit runtime control over scheduling by explicitly activating/deactivating particular channels.

Processing for video channels is similar to that of other media types, except that special-purpose facilities are provided to manage network delay and local buffering of data. We discuss these activities in section 3.4.

Synchronization constraints. The temporal aspects of a document's presentation are expressed in terms of synchronization constraints. We can identify two very general types of synchronization: low-level intra-stream and high-level inter-stream synchronization. Intra-stream synchronization concerns the control of delay and jitter of the data transfer between the data server and the presentation client; for our purposes, we consider this a solved problem. Inter-stream synchronization is more difficult, since it involves semantic dependencies as well as encoding-based timing concerns. (Inter-stream synchronization is often called orchestration [24]; we avoid this terms because it implies a particular layered implementation technique rather than the definition of specification-level constraints.)

Perhaps the best-known video related synchronization constraint is 'lip synchronization.' Here, a piece of video is accompanied by a separate audio track, where both data items must be presented so that sound utterances are presented at more or less the same time as the corresponding video frame. While the development of algorithms to support lip-synchronous presentation under varying presentation circumstances remains an active research topic, we feel that is of only limited importance in presenting embedded video data: the video sequences are often not long enough to justify extensive processing to support utterance matches. Instead, if lip-synchronous (or other isochronous) presentation is important, a complex object will probably be defined that resolves the problem at the source.

A more useful class of synchronization constraints is the specification of interactions among a group of objects that can be conditionally presented together. In Fig. 6, we saw a continuous text block on the value of birds that was presented together with the various bird video sequences. It is useful to define two types of synchronization constraints for this type of presentation: encoding-relative and content-relative. Encoding relative synchronization allows two or more events to be scheduled relative to the activation of the encoding of a control event; this usually takes one of the following forms: start together, end together, offset starts, etc. Content-based synchronization allows events to be scheduled relative the sematics of the information contained inside the data associated with an event; for example, a particular portion of a text sequence can be set to be displayed when a particular portion of a video sequence is reached.

If a document is defined to run locally, most of the synchronization constraints can be analyzed before the presentation begins [1], [8], [19]. If, on the other hand, a networked presentation environment is used, the specification and analysis of the timing dependencies becomes more difficult—especially if runtime variations in the availability of resources are considered. In hypermedia documents, one can question the value of extensive pre-scheduling of synchronization points, since it is never certain when—if ever—particular portions of a document will be reached.

Interaction constraints. If documents containing embedded video data are activated in 'VCR' mode, the required user interaction facilities are relatively trivial to support. In this mode, fast-forward/rewind/slow-motion/pause functions can be implemented by increasing/decreasing the rate of a 'document clock' or searching the encoding for key frames. Once a hyper-information structure is imposed on the documents, however, interaction control becomes significantly more complex.

Having video be the target of a link is relatively straight-forward. Having video contain the sources of links is more of a challenge, since link anchors need to be associated with each video frame. Each frame will ultimately need to be tagged with a collection of link anchors and the playback environment will need to resolve user interaction (typically a mouse click) to a particular anchor; while automated techniques are under study to assist in this task [5], it remains a difficult implementation problem in part because of practical limits (size, copyright) on producing multiple versions of video source data.

In the standard case of following a link, the control focus of a document can shift to a new document portion much in the same way that a subroutine is used in a sequential programming language. In our personal communications example in section 1, a user can follow a link to an embedded video fragment, display it, and then presumably return to the point at which the link was taken. A more challenging case is to define the behavior of a collection of media objects when a link activates one or more target objects: should the source objects be frozen, should they continue or should they be terminated. In the AHM, we define this behavior in terms of a link's *context*. When link context is defined, the semantic structure of the document becomes richer. A detailed discussion of link context is beyond the scope of this paper; interested readers are invited to consult [14].

3.4 Embedded Video Scheduling and Control

Once a document is activated, the presentation, synchronization and interaction constraints must be met in terms of the dynamics of a particular runtime environment. The advantage of separating event and data block descriptors from the data blocks is that individual events can be pre-fetched based on a forward-interrogation of document components. This can help balance resource use or it can serve to implement synchronization relationships among data items. (Note that the utility of pre-fetching events will depend on the degree of hyper-navigation performed by the user: a data block may be pre-fetched by the runtime environment, only to be discarded because the user follows a link.) A second advantage of externalizing data block properties is that the authoring system can use them to perform static document layout and event scheduling operations.

The following sections describe how the CMIFed system shown in Fig. 4 and discussed in the Appendix, supports embedded video.

Document activation. When a CMIF document is started, a scan is made over (a portion of) the specification and an internal event map is created that contains a partial ordering of document events. The CMIF document consists of a tree-structured document that contains parallel and sequential structure elements and event descriptors, as well as channel maps and interaction control structures. The document also contains structure elements used to constrain scheduling and resource use.[1] Fig. 8 shows a simple document fragment that we will use to explain how we schedule embedded data.

The scheduler looks at a sub-tree structure and issues activation requests to each channel thread required within that fragment. For the fragment in Fig. 8, the scheduler would look at the local root and then construct a partial ordering of those data nodes that were 'immediately visible.' The immediately visible nodes in the figure are the children labelled *d*, *c*, and *a* of the parallel node at left, containing a video sequence, headline text and formatted text block—all of which are presented together—and the video block labelled *b* at the right of the tree, which is presented after the parallel node has finished. The scheduler also sees a sequential node at the scheduling boundary (represented by the dotted line), but it does not allocate any resources or initiate any scheduling activity for the nodes inside this boundary. (The nodes inside the boundary are a conditional group that only get activated as the result of specific user control actions, or they when they are a target of an external link. For scheduling purposes, it is only important to note that they do not get scheduled until they are explicitly activated.)

The scheduler next determines which channels are associated with each of the nodes *d*, *c*, *a* and *b*. For simplicity, assume all of the video data in the example is presented by the same channel thread: this means that all of the segments are presented in the same region on the screen and are of the same size. The scheduler

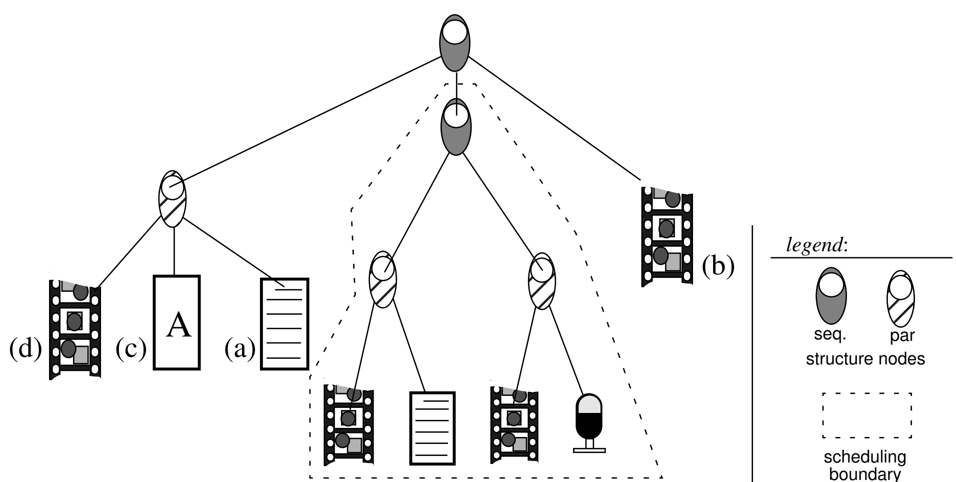


Figure 8: Document scheduling/activation.

1. These elements, called *choice nodes*, also have a role in determining link behavior and general navigation control [14]. We discuss only those aspects relevant to scheduling.

would *arm* each of the channels to prepare them to transfer data and, once armed, would signal each channel thread to start actual data transfer. (Note that if there were synchronization constraints across a set of channels, these could influence the exact start time of each data node.) Once transfer had started, the scheduler would see if there were subsequent events scheduled for active channels; if so, these are *pre-armed* so that they are ready for activation. The behavior of a channel associated with arming and pre-arming is channel dependent; we discuss video channel processing in the following section.

After all of the active events in a document fragment have completed, the scheduler activates the next set of events. If a control action changes the current focus of the document to another portion, then the scheduler restarts its scheduling activity. The scheduler can also control the document clock in response to user interaction (fast-forward/slow-motion), with each active channel thread implementing media-specific changes.

Video channel support. The video channel performs three functions. First, it implements data access and local device control. Second, it implements (pre)arming of events. Third, it changes data rates in response to dynamic system behavior.

Accessing data and interfacing to local devices is done through the facilities of the local operating system and the data server. In most cases, the video data will be stored in the system's file system, although it could also be accessed through a database server. Video data can also be scaled or transformed by the channel thread.

Video pre-arming activity depends on the location of the data and access statistics maintained by the video channel. For local data, the file holding the video is accessed and a fixed number of frames are read into an internal buffer; this assumes that local activity is well understood. For networked data, the file is opened and one or more frames are fetched, based on statistics gathered for earlier accesses to that particular server. (If there were no previous accesses, a single frame is obtained.)

During video transfer, the number of frames received are compared with the number of frames expected based on the document clock (set by the scheduler) and the actual time of the system clock. The fetch rate of the video data can be increased or reduced depending on the constraints of the document.

Document interaction. Standard VCR interaction is handled by the scheduler and the media channels as described above. When a link is followed, the scheduler and the AHM link processor determine the behavior of the active nodes at the link's source (these can be paused, replaced or can continue) and then transfer or co-schedule control at/to the target of the link. (In terms of Fig. 8, the area within the dotted link might be activated.) The target can be either a single data event or a group of events, usually contained within a scheduling boundary. Once the link is activated, the scheduler continues normal processing.

At present, CMIFed supports linking to video sequences that are contained in a document sub-tree, or from parallel nodes that include video data but where the link source is in a static data type.

3.5 Implementation Concerns

The scheduling of a document remains a complex activity, especially if system dynamics are considered in making scheduling decisions. The primary advantage of the channel approach to scheduling is that it can be used to tailor documents for particular execution environments, with intelligent processing contained inside each channel thread. Since multiple type of threads can exist for each media type (each with different processing approaches—and processing overhead), the author and the user can flexibly define dynamic system behavior.

We have found the development of runtime solutions to document scheduling both rewarding and frustrating. It is nearly impossible to adequately pre-compute event schedules for use in network environments (where server and operating system resource allocation is difficult to influence) and even the ‘simple’ case of local execution from local data sources includes an unpredictability resulting from the use of hypermedia control. Instead, a series of heuristic scheduling algorithms seems to provide the best short-term solution to implementing control across channels.

In implementing control communication between the various media channels, it is also clear that local processing of all presentation, synchronization and interaction constraints should not be left solely to the most local level. As a result, an intermediate processing layer (the *global manager*) is used to coordinate messages among the various media players. The global manager is separated from the channel scheduler and the link processor to enhance portability; processing of control requests nearly always involves some form of machine dependencies, which can be isolated to the global manager. The various local managers translate global (often time-based) interaction points to values appropriate for various media formats and further isolate channel control from the functions of media access libraries. This situation is sketched in Fig. 9. The global manager can be thought of as common constraint implementation code, where a video player could be instructed to perform a low-level action at a certain time. When this time occurs, the local manager can signal the global manager for further instruction. (Each local manager can implement a suite of control algorithms, with a document author determining what kind of recovery action he would like the local manager to take.) While the overhead in sending

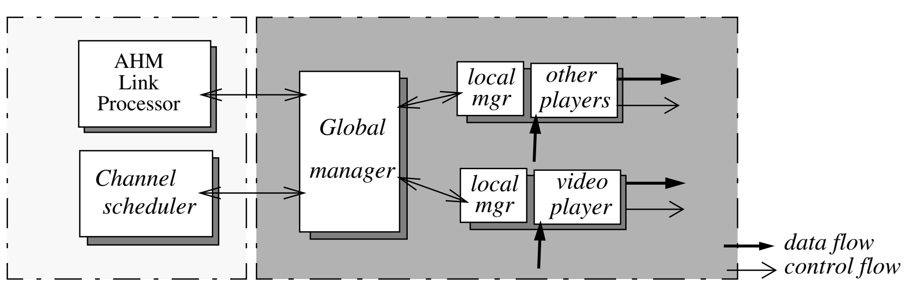


Figure 9: Partitioning of channel control management.

control messages is too high to guarantee very regular, highly detailed synchronization, it does provide enough flexibility to implement coarser check-point synchronization, such as making sure that an audio and embedded video segment remains synchronized at 10% sampling intervals.

4. ADAPTIVE PROCESSING OF EMBEDDED VIDEO

The work described in the previous section gives an overview of how we handle embedded video in a relatively conventional environment. CMIFed has focused on having an author specify desired behavior, having the user indicate actual behavior and having the local environment implement the presentation and synchronization requirements. The extensions to CMIFed described in this section take a more adaptive approach to resolving runtime conflicts by have multiple projections of embedded data available and then selecting one of these dynamically, based on the state of the system (including the local and remote hosts, and the network infrastructure) when an object is accessed.

While some degree of adaptability has been studied for applications such as plan-based document synthesis [9] and automated generation of documents/interfaces [27], [28], our approach differs because we do not assume that there is a single 'correct' view of the information being presented. We also do not try to optimize content to realize a particular task (from an knowledge-based perspective). Instead, we are interested in managing the complexity of general-purpose presentations when a series of alternative representations exist that can reduce the systems impact of fetching, transferring and presenting information. The burden for specifying alternatives remains with the document author; the enhanced system simply tries to find the best fit that it can based on the state information it has available at runtime.

4.1 Defining and Managing Alternative Representations of Embedded Video

In Fig. 5, we presented an *event -> data block descriptor -> data block* hierarchy. The utility of this approach is that it provides the authoring system and the runtime environment with an explicit mechanism for making static and dynamic decisions on the use and constraints of particular events. The same mechanism, coupled with extensions to the CMIF channel model, provides the basis for adaptive system behavior support.

We illustrate our approach with the following non-video example. Consider the two event descriptors that point to a common data block descriptor, which points to a text-based data block containing our cheesecake recipe (Fig. 10(a)). Event descriptor *a* uses the *HQ_NL audio* channel and defines a processing filter (called *text2dutch*) to make a *projection* of the text block onto spoken Dutch output, while event *b* uses the *LQ_MC audio* channel and defines a filter to map the text to a Morse code output string. In the channel architecture that is supported by CMIFed, the user can make an explicit selection of either channel (or both) at runtime. The choice is made based on *user preference*.

An alternative to user preference is *system preference*, as illustrated in Fig. 10(b). Suppose, for example, that only a single channel was defined that could produce

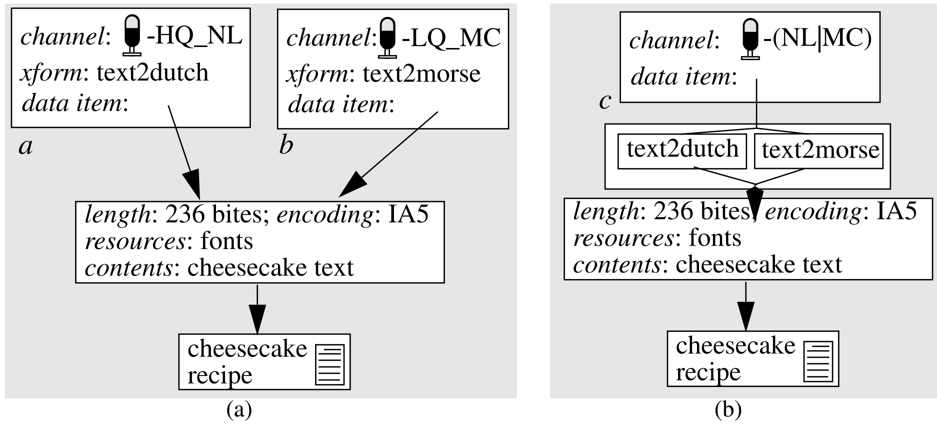


Figure 10: Multiple event projections of a single data block.

either high- or low-quality audio output. Rather than having the user explicitly select which of the available encodings he wanted, the runtime system could select an encoding on the audio channel based on the classification of the available hardware. Given an office workstation, for example, the highest quality encoding would always be used. (In this case, the Dutch output.) If the same application was accessed from an airborne laptop using a low-bandwidth connection, the low-quality channel would be used, yielding a Morse code presentation. The user would not have to explicitly participate in the runtime selection process: instead, the environment could be configured for any user limitations or preferences, and the decision could be made dynamically by the presentation system.

Although video is not a data type that lends itself to dynamic projections from a single data block at runtime, several possibilities exist for pre-creating the alternative projections illustrated by Fig. 11 . For example, special-purpose processors could create a set of reduced-size encodings (using MPEG1-4, JPEG, etc.) of the original video sequence. Another option is to define projections that use other media such as text, audio or still images, each of which can be generated from the scripts and (extensive) production notes that are required for the creation of even relatively simple videos. Given the availability of these projections, the runtime environment can choose among the reduced-size encodings or the text, audio and video projections based on the available resources, the priority of each encoding type and the

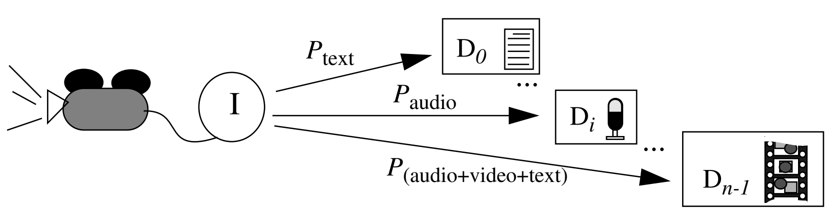


Figure 11: Data Projections of Abstract Information.

characteristics of the target workstation much in the same way that multiple projections could be generated from a single encoding of text.

4.2 Adaptive Information Object Model

One of the principal control operations in extended environment is a freedom to select individual data representations of an adaptive information object (AIO [2],[4]) at runtime. Our abstraction for the AIO shown in Fig. 12.

In this illustration, three alternative representations of a video clip are shown: the video clip itself (R_0), an audio description with a set of still images (R_1) or a text block with captioned stills (R_2). Selection of a particular data set is supported through a set of control interfaces. These interfaces include:

- *representation control*: The choice of a particular AIO data projection is governed by the constraints of the application and the nature of the environment. In general, support is provided for heterogeneous target environments or environments supporting a range of data quality (for example, high-resolution images or stereo sound for high-end workstations; low-resolution images or text substitutions and low-quality audio for the low end. The goal is to provide a natural way for supporting heterogeneity within the environment without overloading the application author or user.
- *resource control:* while many of the basic operations of the AIO relate to one-time selection of a representation for a particular instance of the object, it may be necessary to control the delivery of data because of limitations of the environment. These limitations may occur at the server or they may come from the target host. They may also come from other AIOs (on other source hosts) who are not able to meet their data requirements or from global resource limits.
- *synchronization control*: the timely delivery of data, either by a single object or by several objects working in concert, is of fundamental importance to the operation of multimedia computing. The synchronization control interface provides a general mechanism for inter- or intra-stream synchronization. The AIO may also initiate control communication with other components via this interface.

As shown in the figure, a particular AIO may be complex, having many representations available, or it may be a simple server that knows how to present controlled delivery of a single data type. Each AIO may be a separate entity or it may be man-

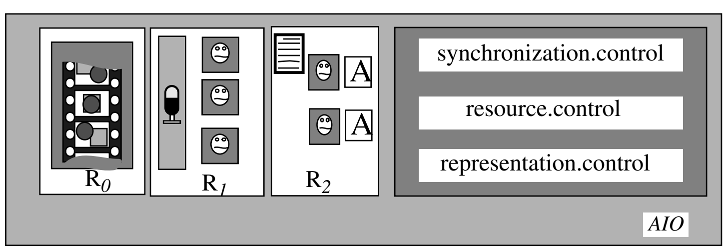


Figure 12: An Example Adaptive Information Object.

aged via an object server. The overall role of the AIO is to manage the implementation of constraints on the presentation of data.

The AIO model is more than a simple extension of the local manager illustrated in Fig. 9. Where the local manager processed data based on activity on the local presentation system, the AIO involves client-server negotiation to determine a desired representation under given resource and synchronization constraints. Our future plans include integrating local and remote operating system support into the constraint resolution process, but no specific low-level constraint management algorithms have been developed to date.

4.3 User-Level Quality-of-Information Support

In order to normalize the processing of AIO control interfaces, a constraint interface is being developed to implement a user-level quality of service specification. In order to distinguish this from the low-level quality-of-service (QoS) parameters that have been the topic of extensive study, we term these higher level concerns *quality of information* (QoI) constraints. The intention of the QoI control is to provide the author/reader and the runtime environment a selection mechanisms for AIO access. These constraints fall into two broad categories: *static QoI constraints*, whose impact on a particular projection selection is known when the document is defined, and *dynamic QoI constraints*, whose impact on projection selection is not known until the document is accessed. (Fig. 13.)

Static constraints on the selection of a projection are those that are defined when the document or information object is authored/created. While the resolution of these constraints occurs at runtime (that is, a particular projection can be selected based on the alternatives provided with the document), the alternatives usually can be analyzed before presentation begins. Examples are:

- *information encodings*: information can be mapped to one of several projections, each of which may use one or more types of presentation media. While selecting among projections is a runtime task, the range of projections available are assumed to be know statically—even though the information itself may be synthesized at the time it is referenced. The projections may be kept together in an information object database, they may be stored separately in media-related databases or they may be stored as files in a file system. Access/charging constraints may also exist.

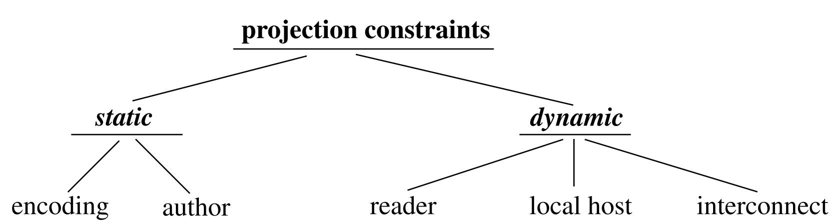


Figure 13: Coarse projection-constraint classifications.

- *author preferences*: while defining a presentation, the author can specify the representation(s) that best project the information under a variety of circumstances. These circumstances will have both syntactic and semantic components, such as "end together" or "place object A to the left of B because the content of B is identified as being to the right of A." They can be conditional:

  if audio is active **and** if video is active
  then delay until end of longest sequence
  else delay *n-units* of time;

  or they can involve user interaction. Author preferences are static because they are defined before a particular presentation takes place; the selection of an alternative depend on the combination of user preferences and system states.

Dynamic projection constraints are those that depend on the combination of a particular user of a document and a particular presentation environment. Here, support is provided for heterogeneous presentation systems *and* heterogeneous users. In addition, we also include constraints based on the runtime state of the interconnection environment. Dynamic factors include:

- *reader preferences/abilities*: for a given presentation, the needs of readers in consuming the information will depend on a variety of factors, including general user preferences and basic user ability limitations. Style preferences may dictate that a user would rather receive text-based projections of information than audio-based projections, while functional preferences may dictate that, because a user is blind, audio data is required in place of text;
- *system preferences/abilities*: for a given presentation, a homogeneous presentation platform cannot be assumed. Some presentation platforms will support a wide range of input and output devices, but others will contain only a subset of those potentially available. As with readers, constraints on the presentation environment are analogous to style-based or ability-based, where style constraints are local preferences, while ability constraints would be based on the presence/absence of a particular type of input/output functionality;
- *environment preferences/abilities*: for a given presentation, the basic ability of the support environment to provide information to the user will be constrained by resource availability across the environment. While, for example, a particular user may prefer to see a sequence of video images instead of a block of text, the underlying environment—including information servers, transport networks, intermediate buffering hosts, local operating systems, etc.—may not be able to provide this service (even if the local presentation environment can).

The nature of constraints is not yet well understood, and the list above is not complete. What is clear, however, is that some mechanism is required to specify system behavior in a more direct way than is currently available. The approach of having an author specify a set of interpretation constraints allows for a degree of extra information on intent and content evaluation that can be important in making resource allocation decisions across the network. Similarly, having execution environments specify the limits on their resource availability should allow allocation decisions to be made at a lower level than is currently available. This philosophy is more important than any single set of constraint characterizations.

5. CLOSING COMMENTS

There are a wide range of problems that need to be addressed before video data can be stored and accessed with the same flexibility as text, or even with the limited flexibility currently available for pictures and audio data. Obviously, video data will need to be created and edited, and it will need to be stored and made available to applications on demand. In this process, the artistic and design considerations in producing effective video sequences may prove to be the dominant challenge; given adequate bandwidth and affordable video decompression hardware, the actual delivery of video data should not present a significant future problem.

If there is a technology-related challenge to video processing, it will be in describing and implementing the synchronization and coordination of embedded video data into the broader context of a document. Where traditional video support has focused on the application-independent transfer of bytes from a server to a client in a time-constrained manner, the use of embedded video implies a shift to processing the video data in the context of the application. As such, low-level services will need to be able to receive information on that context at the time the request for a video transaction is initiated. The short duration of each fragment means that the overhead required to support conventional, connection-based transfers may not be justified for embedded video: a long video setup time during which transfer resources are allocated and admission control is provided may be incompatible with the needs of presenting 15 seconds of video data. In addition, the combination of embedded video with other data types makes synchronization within the presentation multi-dimensional: along with intra-stream synchronization, the contents of each stream must also be synchronized with the presentation's other events. While this is not a new problem, the need to synchronize collections of short-duration, time-sensitive data elements can present a significant drain on system resources.

When coordination and synchronization is required, the quality needs of the application will probably dictate how video data will be encoded. If, for example, true frame-by-frame synchronization of audio and video is required, we expect that this will be done near the encoding of the data (much in the same way that it is done in 'real' video and film productions). These tracks may be produced separately, but they will be merged prior rather than during document processing. Where synchronization requirements are less exacting (such as defining background sounds, multilingual audio/text captions or events that require point rather than continuous synchronization), the presentation environment will have a much greater role to play.

At present, the CMIFed environment allows a set of models to be used to externalize the behavior of applications using embedded video, audio, image and text data. In highlighting the problems and issues involved in the presentation of embedded video, we were struck by the fact that our environment provides relatively few features that were specifically video related. Rather than this being a deficiency, it is a reflection of our view that 'video is no big deal.' In nearly all areas of hypermedia data processing, video does not require new conceptual models—the size and encoding restrictions of video may make implementation tedious, but the techniques for handling and synchronizing video are similar to those required for other presentation-sensitive data. We are convinced that the focus needs to shift to high level integration of data, where some sort of semantic coupling is required of (possibly read-only) data encodings and some form of higher-level interaction descriptions.

The work described in this paper presents an approach embedding video by balancing the needs for quasi-real-time presentation of data with the demand for high level control by the document player to implement the desires of the document author. We feel that the CMIFed environment provides a good framework for addressing these problems, although it is clear that much more needs to be done to seamlessly integrate various independent data streams (including but not limited to video) into a single document. Perhaps the most pressing need is to understand the role that adaptive presentations can play in defining truly portable documents. Given the broad diversity in user needs and presentation environments—a diversity that will increase rather than decrease in the future—we feel that tools and techniques are required to support document adaptability from initial authoring through final presentation. This remains the focus of our group's efforts.

ACKNOWLEDGMENTS

The CMIFed environment has been developed during the past four years by a programming team including Guido van Rossum (of *Python* fame), Jack Jansen and Sjoerd Mullender. This group remains responsible for the steady stream of changes and enhancements (most of which have been improvements) that have made the environment what it is today. Lynda Hardman has been the chief architect of the Amsterdam Hypermedia Model and has patiently shepherded many of the model's features—most notably, the concept of link contexts—into the execution environment. Lynda also gave freely of her time and energy in contributing improvements to this paper, for which I am very grateful. Dik Winter contributed implementation support for AIO processing and Maria Theodoridou of FORTH in Greece has provided a recent reminder that much still needs to be done before our implementation meets all its specifications.

This work was supported in part by the STEN project (funded under the European Community's RACE program), by the MAGUS project (funded in part by the Dutch Ministry of Economic Affairs) and by CWI. Continued support for this work is provided by the ESPRIT-IV project CHAMELEON, for which CMIF, the Amsterdam Hypermedia Model and CMIFed form the core.

APPENDIX: GENERATING EVENT RELATIONSHIPS FROM STRUCTURE

In order to illustrate our approach to externalizing the behavior of an application document, we describe how the CMIFed authoring environment generates and implements event relationships using information about the application's structure. The motivation of the CMIF project at CWI was the development of a document model and manipulation environment that allows the adaptive presentation of hypermedia documents in networked, heterogeneous environments. While a critical component of this work is a timeline-like description of the low-level interactions among data blocks, our timeline is generated automatically from a higher level description of how components within the document interact. The higher level description (which we call the *hierarchy view*) allows a document to be described and manipulated at a structural level rather than as a collection of independent media events. This structure is then mapped onto a *channel* architecture, where each channel describes a grouping of data items that can be manipulated by the user or presentation environment in a common manner. In the following subsections we describe the basic nature of our document model and then discuss how media events are structured and mapped to a presentation timeline.

A.1 Overview of the CMIF document model.

The data organization model used by CMIFed is based on a hierarchy of events. Within this hierarchy, events can be defined to occur in parallel with or before/after other events. When displayed, the presentation flows from the root of the hierarchy through all of the nodes. Individual data blocks can be fetched by the player based on the current location within the hierarchy—this gives the scheduler an ability to plan ahead and allocate resources for groups of pending events. A hyperlink-based navigation facility is also supported, allowing the user to jump from one point in the presentation to another.

Our approach to data organization is presented in Fig. A.1, which illustrates a document structure containing nine media events (leaf nodes A-I) of three data types, ordered as a collection of parallel and sequential events. In this example, a fragment of video data is presented in parallel with two sequential audio fragments

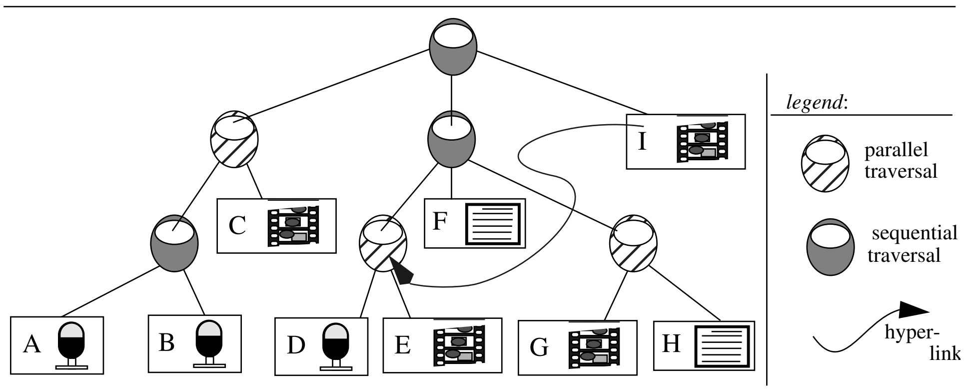


Figure A.1: An application specification as a hierarchy of media events.

(nodes A, B and C). Next, an audio fragment is presented in parallel with a video fragment (events D and E), followed by a block of text data (event F). Next, a video fragment and a text fragment (events G and H) are presented, followed by a single video fragment (I). A hyper-link has been inserted by the author between the video fragment in event I and the composite events D and E (the behavior of this operation is described in section 4).

As part of the definition of each event, the document's author specifies a *resource management channel* (or, more simply, a *channel*) that is used for presentation of the document. The channel framework consists of an activation timeline that is generated automatically by the authoring system. Working at the channel level, a series of fine-grained interactions can be specified between events using *synchronization arcs*; these indicate (relative) presentation constraints of a particular object. The full implications of this action will be considered in section 3.

A.2 The Hierarchy View for Structure Manipulation.

The hierarchy view shown in Fig. A.2(a) is the primary CMIFed authoring window, providing a means of displaying and manipulating the document describing a multimedia presentation. The hierarchical structure is represented in the hierarchy view as an embedded block structure; this embedded structure has proved to be much more manageable that manipulating an actual tree, since it allows us to easily hide the (possibly complex) structure under any one node. Individual media events are placed as leaf nodes in the tree, represented as the lowest level of detail in the hierarchy view. We assume that each media object is created using external editors (available directly from within the authoring environment) or from an object store.[1]

As part of the definition of the hierarchy, each data event is assigned to a channel, a logical output device which is mapped by the player at runtime to a physical output device—i.e. an area on the screen or a loudspeaker. Rather than mapping all events of a particular media type to a single media-based channel, we allow multiple channels of the same type to be supported. Associated with each channel is a resource management policy that can be used by the player to resolve conflicts that arise in a networked, heterogeneous environment. (See section 3.)

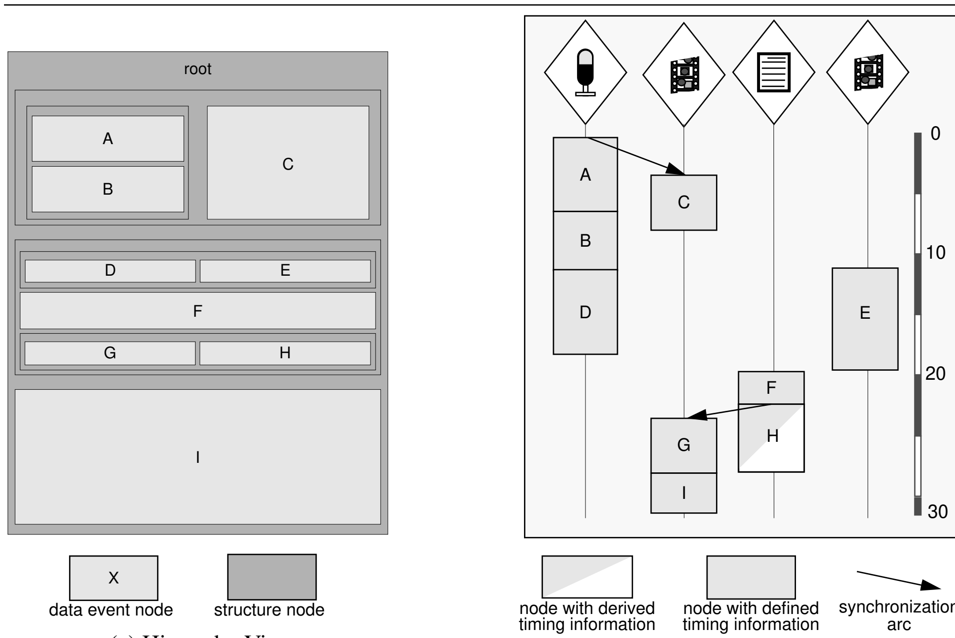
The structured approach to authoring provides two primary advantages. First, editing of a group of related events—at any point in the hierarchy—is a simple operation. We need only select the common parallel or sequential node and change its location using a single cut/paste operation pair. In so doing, we wind up reordering all events in a single operation—including the movement of hyper-links. (This may seem like a simple and intuitive operation. To understand the impact of this feature, readers are encouraged to go to their local computer store and try doing this with any conventional editing system.)

1. Creating the data blocks is the most time consuming part of authoring—especially for video data—but this process is becoming easier as data editing tools become more sophisticated. We explicitly factor editing out of the authoring process to allow for non-embedded tools to be used.

The second advantage of our approach—especially for including embedded video fragments—is the ability to delay timing considerations until relatively late in the authoring process. By concentrating on higher-level constraints, such as parallel and sequential placement, the details of exactly how individual events are related can be saved until a fine tuning phase near the end of the production cycle. Note that the definition of the data block can occur in parallel with the authoring process: stubs for non-existent data fragments can be easily inserted in the document structure as place-holders. Also, since timing information is generated based on the contents of the actual data blocks (or the block descriptors), a change in the source data length integrates seamlessly into new versions of the document.

A.3 The Channel View for logical resource allocation.

While the hierarchy view provides a means of organizing the structure of a presentation, it provides only an indirect way of specifying logical resource use. To provide the author with an explicit time representation of the document and control of



(a) Hierarchy View

(b) Channel View

(a) The hierarchy view is an embedded representation of a document hierarchy, illustrating both the data events and their structural relationship. The hierarchy from Fig. 1 is shown in a fully-expanded view (that is, no information hiding is used, although this is possible).

(b) The channel view shows a mapping from document hierarchy to activation time; this mapping is generated automatically. The event assigned to a channel is represented as a box beneath the channel header diamond. The height of a box represents its duration. A fully-shaded box has its duration explicitly defined, either through its data definition, an author-defined default duration or via synchronization constraints. A box with a shaded triangle inherited its duration from its parent in the presentation's structure. (The use of two video channels is discussed in the text.)

Figure A.2: Hierarchy View and Channel View the available resources, the channel view provides a view of the media objects mapped onto a set of logical resources. By supplying an extra layer above the physical resources, the author is able to guide the presentation of a document in a system-independent way.

The channel groups data events by their abstract properties; these can include:

- *presentation* properties (font use, video resolution or screen real-estate);
- *semantic* properties (languages used within a channel or data events of a particular content granularity); or
- *scheduling* properties (such as high priority, low priority, incidental data or absolutely-necessary-must-be-shown-at-all-costs data).

It is up to the player software, optimized for a particular hardware configuration, to interpret the logical channels and assign the media objects to the available physical output devices.

The channel view shows the timing relations derived from the structure defined in the hierarchy view. Since timing is derived, the author is not required to count frames or be aware of the sample-rate/clock-frequency relationships on any one target system. The correspondence between the structure shown in the hierarchy view and the data nodes in the channel view is shown in Fig. A.2(b). The channel view provides an independence between logical resources required by a document and physical resources available at the time the document is played.

Although most of the timing relationships between and among components can be derived from structure, it is sometimes necessary to define more detailed timing constraints between components. The *synchronization arc* (or *sync arc*) [2],[4] is our mechanism for supporting this definition. Inter-channel sync arcs specify the bounds on activation of two events; this includes relative starting times and synchronization priority. Intra-channel sync arcs could be used for transition processing (such as wipes, fades, etc.), although we are investigating a separate structure to support these operations.

REFERENCES

[1] Buchanan, M.C., & Zellweger, P.T., Automatically Generating Consistent Schedules for Multimedia Documents. ACM/Springer-Verlag *Multimedia Systems Journal*, 1(2), 1993.

[2] Bulterman, D.C.A., Specification and Support of Adaptable Networked Multimedia. ACM/Springer-Verlag *Multimedia Systems Journal*, 1(2), 1993.

[3] Bulterman, D.C.A. and Hardman, L., Multimedia Authoring Tools: State of the Art and Research Challenges, in *Lecture Notes in Computer Science* #1000 (J. van Leeuwen, Ed.), Springer-Verlag, 1995

[4] Bulterman, D.C.A., Van Rossum, G. & Van Liere, R., A Structure for Transportable, Dynamic Multimedia Documents. In *Proc. Summer 1991 Usenix Conference*, Nashville TN, June 1991.

[5] Burrill, V., Kirste, T. & Weiss, J., Time-varying Sensitive Regions in Dynamic Multimedia Objects: a Pragmatic Approach to Content-Based Retrieval from Video. *Information & Software Technology 36(4), 1994* 213-223.

[6] European Commission, CHAMELEON: Authoring Support for Adaptive Multimedia Documents, CEC ESPRIT-IV Project 20597 (1995); URL:http://www.cwi.nl/chameleon/
[7] Dey, J.K., Salehi, J.D., Kurose J.F. & Towsley D., Providing VCR Capabilities in Large-Scale Video Servers. In *Proceedings of ACM Multimedia 94* (San Francisco, California, October 15-20, 1994), pp 25-32.
[8] Diaz, M. & Senac, P., Time Stream Petri Nets: A Model for Multimedia Streams Synchronization. In *Proc. Multimedia Modelling '93*, World Scientific Co., Singapore Nov. 1993.
[9] Feiner, S.K. and McKeown, K.R., Automating the Generation of Coordinated Multimedia Explanations. *IEEE Computer,* 24(10), pp. 33-41, 1991.
[10] Gemmell, J., & Christodoulakis, S., Principles of Delay Sensitive Multimedia Data Storage and Retrieval. *ACM Transactions on Information Systems*, 10(1):51-90, 1992.
[11] Gong, F., Multipoint Audio and Video Control for Packet-Based Multimedia Conferencing. In *Proceedings of ACM Multimedia 94* (San Francisco, California, October 15-20, 1994), pp 425-432.
[12] Halasz, F. & Schwartz, M., The Dexter Hypertext Reference Model. *Comms. ACM 37(2)*, Feb. 1994.
[13] Hardman, L., D.C.A. Bulterman & G. van Rossum, The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model. *CACM 37(2)*, Feb. 1994.
[14] Hardman, L., D.C.A. Bulterman & G. van Rossum, Links in Hypermedia: the Requirement for Context. *Proc. ACM Hypertext '93*, Seattle, WA (USA), Nov. 1993.
[15] Internet Engineering Task Force, HTML document specifications. Working documents available from the IETF electronically viaURL: http://www.cnri.reston.va.us/
[16] ISO, HyTime. Hypermedia/Time-Based Structuring Language, ISO/IEC 10744:1992.
[17] ISO, MHEG. Information Technology Coded Representation of Multimedia and Hypermedia Information Objects (MHEG) Part 1: Base Notation (ASN.1), ISO/IEC CD 13552-1:1993, Oct 15 1994.
[18] Liestol, G., Aesthetic and Rhetorical Aspects of Linking Video in Hypermedia. In *Proc. ACM ECHT '94*, pp 217 - 223.
[19] Little, T.D.C., Ahanger, G., Folz, R.J., Gibbon, J.F., Reeve, F.W., Schelleng, D.H. and Venkatesh, D.A., Digital On-Demand Video Service Supporting Content-Based Queries. In *Proceedings of ACM Multimedia 93* (Anaheim California, August 1-6, 1993), pp 427-436.
[20] Macedonia, M. & Bruzman, D., MBONE, the Multicast Backbone. *IEEE Computer*, April 1994.
[21] Meyer-Boudnik, T. and Effelsberg, W., MHEG Explained, *IEEE MultiMedia*, 1(4), pp. 26-38, Spring 1995.
[22] Miller, G., Baber, G. & Gilliland, M., News On-Demand for Multimedia Networks. In *Proceedings of ACM Multimedia 93* (Anaheim California, August 1-6, 1993), pp 383-392.
[23] Newcomb, S.R., Kipp, N.A. and Newcomb, V.T., HyTime: The Hypermedia/Time-Based Document Structuring Language, *Communications of the ACM*, 34(11), pp. 67-83, 1991
[24] Nicolaou, C., An Architecture for Real-Time Multimedia Communication Systems. *IEEE J.SAC*, 8(3), pp. 391-400, April 1990.
[25] Postel, J., Media Type Registration Procedure, Internet RFC 1590, March 1994.
[26] Ramakrishnan, K.K., Vaitzblit, L., Gray, C., Vahalia, V., Ting, D., Tzelnic, P., Glasser,

S., and Duso, W. , Operating System Support for a VIdeo On-Demand FIle Service. In *Proceedings of 4th International Workshop on Network and Operating Sytems Support for Digital Audio and Video* (Lancaster, UK, Nov 1993). Springer-Verlag LNCS 846.

[27] Rist, T. and André, E., From Presentation Tasks to Pictures: Towards a Conceptual Approach to Graphics Design. *Proc. ECAI-92* (Vienna, Austria), Wiley, pp.764-768, 1992.

[28] Roth, S.F. and Hefley, W.E., Intelligent Multimedia Presentation Systems: Research and Principles. In *Intelligent Multimedia Interfaces* (M. Mayberry, Ed.), AAAI Press, Menlo Park CA, pp. 13-58, 1993.

[29] Van Rossum, G., An Introduction to Python for UNIX/C Programmers. In *Proceedings of USENIX Symposium on Very High Level Languages* (Sante Fe, NM), October 1995. (See also `URL: http://www.python.org/`)

[30] Van Rossum, G., Jansen, J., Mullender, K.S., & Bulterman, D.C.A. (1993). CMIFed: a presentation environment for portable hypermedia documents. In *Proceedings of ACM Multimedia 93* (Anaheim California, August 1-6, 1993), pp. 183 - 188.

[31] Schooler, E.M., Case Study: Multimedia Conference Control in a Packet-switched Teleconferencing System. *Journal of Internetworking: Research and Experience*, 4(2), pp 99-120, June 1993.

[32] Steinmetz, R., Analyzing the Multimedia Operating System. *IEEE MultiMedia*, 2(1), pp. 68-84, Spring 1995.

[33] Trigg, R., Link anchoring in dynamic media: A user interface challenge. In *Proceedings ECHT '90* (INRIA, France, November) 1990, Cambridge Univ. Press, pp 359-361.

[34] Vin, H.M., Goyal, P., Goyal, Alok & Goyal, Anshuman, A Statistical Admission Control Algorithm for Multimedia Servers. In *Proceedings of ACM Multimedia 94* (San Francisco, California, October 15-20, 1994), pp 33-40.

[35] Yu, P. Chen, M.S. & Kandlur, D.D., Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management. In *Proceedings of 3rd International Workshop on Network and Operating Sytems Support for Digital Audio and Video* (San Diego, CA, Nov 1992), pp 38 - 49. Springer-Verlag LNCS-712.