A Structure for Transportable, Dynamic Multimedia Documents

Dick C.A. Bulterman (dcab@cwi.nl)
Guido van Rossum (guido@cwi.nl)
Robert van Liere (robertl@cwi.nl)

CWI: Centrum voor Wiskunde en Informatica
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

Abstract

This paper presents a document structure for describing transportable, dynamic multimedia documents. Multimedia documents consist of a set of discrete data components that are joined together in time and space to present a user (or reader) with a single coordinated whole. Transportable documents are those in which the document structure can be accessed across system environments independently of individual component input or output dependencies; dynamic documents are those in which the synchronization of document components are not staticly defined as an integral part of the data definition but are dynamicly defined as attributes of the general document structure.

The focus of this paper is the presentation of the basic building blocks of the CWI Multimedia Interchange Format (CMIF). CMIF is used to describe the temporal and structural relationships that exist in multimedia documents. In order to put our work in a concrete context, we start our discussion with a brief description of the portability requirements for documents used within the CWI/Multimedia Pipeline. We then provide a layered description of our document structure format; this format provides a means for expressing a document in terms of *synchronization channels*, *event descriptors*, *data descriptors*, *data blocks* and *synchronization arcs*, each element of which contains a set of appropriate descriptive attributes. The paper describes each of these concepts abstractly as well as in the context of a uniform example. The paper concludes with a discussion of our intended future direction in using the various attribute descriptors to control a broad range of activities within the CWI/Multimedia Pipeline.

1. Introduction

A visitor to any computer-industry trade show is immediately confronted with the state-of-the-art of multimedia systems. Even modest personal systems demonstrate an impressive ability to simultaneously manipulate a variety of (chiefly output) media in a way that provides a dazzling display of technological cleverness and audio/video wizardry. Birds flying across medium-resolution color screens can be frozen in mid-air, then cut out of their environment and pasted on top of a composite background with remarkable ease; images can be video mixed, then translated, rotated and scaled at a speed that once was impossible with even the highest-performing (and highest costing!) workstations. Novice users can take information from CD-ROMs that contain data of several media, giving promise to better teaching tools for children or clearer maintenance and repair guides for automobile mechanics. At first glance, current generation multimedia systems can do nearly everything that a user could hope for, with the implied promise that even not-yet hoped for things are just around the technological corner.

In reality, however, there are several major problems that confront the user of multimedia systems. One problem is that the elements being manipulated within multimedia systems consist of raw data rather than structured information; the bird flying in the example above usually is little more than a sequenced video FAX that has a representation but little inherent meaning. This can limit the amount of intelligent processing that can ultimately take place by application programs or support hardware. A second problem is that the representation and manipulation of data is highly machine and/or device dependent. This means that information can not easily be shared among different types of systems or devices. A third problem is that the synchronization present within multimedia applications is often implicitly encoded as a function of the speed of a particular system and interface, limiting the ways that interaction among elements can be expressed and (ultimately) implemented. The result of these problems is that it is often difficult to share documents among different applications or across similar applications that are implemented on different computing platforms. Even in those systems that allow sharing mechanisms, the fact that information on data and data formats is often intimately entangled with details of presentation timing or with details of presentation formats, screen resolutions, window placements, etc., makes it difficult for higher-level authoring tools to effectively coordinate the synchronization of a number of otherwise autonomous data streams and for manipulation tools to effectively separate data presentation/placement from data information encodings.

This paper presents a method for addressing the three problems above by focusing on the manner in which information in a multimedia application is represented; we then consider ways of using that representation to coordinate the manipulation of the composite parts of the application itself. In order to do this, we define CMIF, the CWI Multimedia Interchange Format [Rossum91]. CMIF encodes a multimedia document as a collection of data of potentially diverse media *and* as a set of structure and synchronization relationships that describe how the data components are to be presented and manipulated. CMIF has been motivated by two goals: first, to define a description that separates the temporal, spatial, and content-based aspects of multimedia documents, and second, to investigate means of using the document description rather than the data contents to control the interrogation and synchronization of one or more document sets. It was developed to provide the ‘‘glue’’ that binds together various components of the CWI/Multimedia Pipeline; this pipeline is briefly discussed below.

It should be noted that while the purpose of CMIF is to provide a means for expressing dynamic relationships within a document in a transportable manner, this does not imply that all systems that recognize the format will also be able to implement a particular document. (It would be impossible to implement the flying bird document from above if the target system had no display, for example.) What CMIF *can* provide, however, is a structured basis upon which a given system can determine whether it can support the requested document or not.

In the sections below, we describe CMIF in terms of its abstract properties and in terms of an example multimedia document. To motivate the particular requirements of our work, we start with a brief description of the CWI/Multimedia project. We then introduce the basic building blocks used within CMIF. Next, we describe our multimedia example, and use it as a way of exploring some of the more detailed aspects of the document format. This is followed by a more detailed look at CMIF, with special attention provided to the question of document synchronization. We conclude with a brief discussion of some of the ways in which a document structure can be used to manipulate multimedia documents without accessing the (potentially huge) multimedia data sets themselves.

2. The CWI/Multimedia Project

The CWI/Multimedia project is a new area of coordinated research that provides researchers in the areas of systems architecture, operating systems, distributed databases, user interface design and interactive systems to consider problems associated with the manipulation of multimedia data in a distributed computing environment. Areas of interest on multimedia systems can be described in terms of the collection of phases that exist during the lifetime of a multimedia document; these phases have been grouped together into the CWI/Multimedia Pipeline. Figure 1 gives a schematic of the components of this Pipeline.

The elements of the pipeline are:

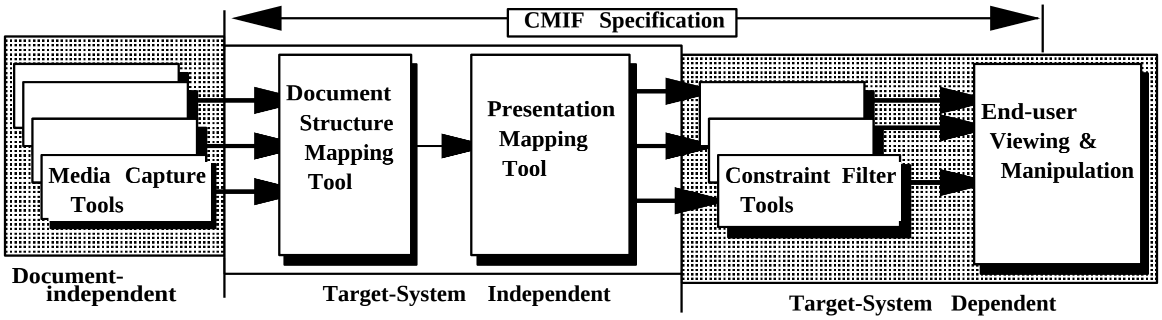


Figure 1: Schematic of the CWI/Multimedia Pipeline

- *Media Block Capture Tools:* a set of tools that will allow the user to iteratively capture (and edit) the atomic pieces of information that will be included in a composite document. In general, our concern is not with the hardware technology associated with the capture of a particular medium: we expect that equipment vendors or third-party organizations will do this better than we can. Instead, our focus is on providing descriptive tools that allow higher-level processing of various bits of collected information. (This processing may include scheduling, searching, comparing, editing, etc.) Note that the concept of an atomic item still allows for a complex structure; since the goal of these subsystems is chiefly to compile descriptors, a broad range of underlying systems can be supported.
- *Document Structure Mapping Tool:* this tool allows the user to express relationships among individual media blocks. The relationships are primarily temporal and spatial. Information from this tool is used by later components to support presentation, local system filtering and editing/reading of a multimedia document. The document structure mapping tool produces a document in the CMIF format.
- *Presentation Mapping Tool:* this tool allows portions of a document to be allocated to a *virtual presentation environment*. This tool is used to allocate virtual presentation ‘‘real estate’’ (such as areas on a display or channels of a loudspeaker) to a given multimedia document. Some of the mapping information may come from ‘‘preference’’ defaults provided with each atomic media block, or they may need to be added by this tool. In either case, this tool manipulates the definitions provided in the CMIF document and creates a presentation map that can be manipulated separately from the document itself.
- *Constraint Filtering Tools:* these tools allows the end-user presentation system to filter components of the document to meet local processing constraints. (This corresponds to a mapping of the document from the virtual presentation environment to a physical presentation environment.) Typical filterings may include 24-bit color to 8-bit color, color to monochrome, high-resolution to low resolution, full-frame-rate video to sub-sampled rate video, etc. As with all components, the assumption is that this tool *manages* a constraint mapping; the actual constraint implementation will be supported by user level, operating system, or hardware level modules.
- *Document Viewing and Reading Tools:* These tools present a document (based on the document structure map, the presentation map, and the local filter map) and provide a means for a reader to ‘‘view’’ or (possibly) edit a document. Note that the document structure map

provides a data-independent, position-independent and system-independent view of the multimedia document being read, acting as an internal table-of-contents function for subsequent reading and editing tools.

The detailed description of each of the elements of the pipeline are beyond the scope of this document. As of this writing, prototype designs of different elements are being undertaken in order to better understand fundamental problems during the processing of multimedia information. From the nature of the pipeline, however, it should be evident that the provision of a central document description is essential if information is to be shared cleanly among disjoint manipulation tools. It should also be evident that a rich document description can improve the performance of document manipulation tools by providing the summary information required by the virtual presentation and constraint tools without requiring the data itself to be manipulated. We return to this point below.

3. Introduction to the CMIF Structure

The purpose of CMIF is to provide a mechanism for describing the structural components that exist in a multimedia document and to describe the synchronization relationships among those structural components. This section provides a first look at the elements of CMIF and then relates it to other document structures used with a multimedia system.

3.1. Overview of the CMIF Building Blocks

The basic CMIF building blocks are summarized in the following table:

Building Block	*Function*
Data Blocks	The basic atomic element of single-media data
Data Descriptors	A set of attributes describing the semantics of the data block
Event Descriptors	A set of attributes describing the presentation of a data block
Synchronization Channels	A placement framework for sequential and parallel events
Synchronization Arcs	The specification of the interaction constraints among events

The relationships among data blocks, data descriptors and event descriptors are given in figure 2.

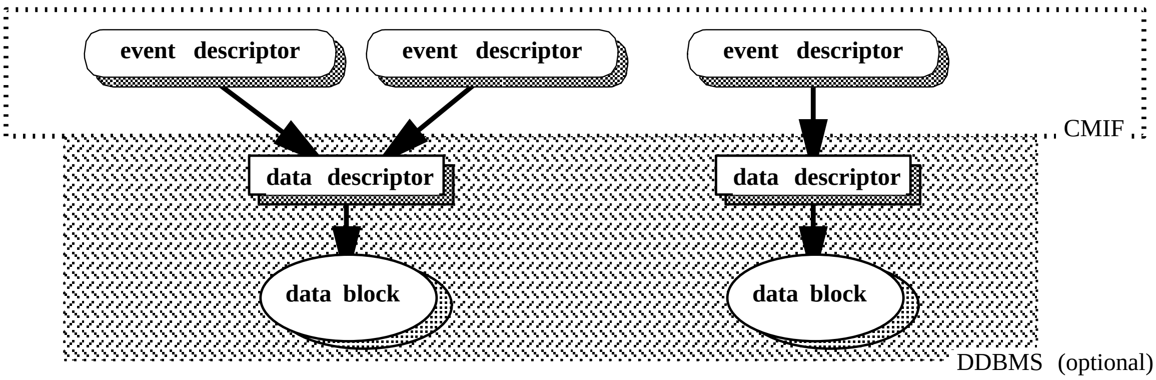


Figure 2: Data blocks, data descriptors and event descriptors

Data blocks contain data that is typically associated with a single medium. Examples may be sound clips, video segments, text blocks, graphics images, etc. They may also be programs that produce information of a particular type. (An example might be a graphics program that produces a rendered 3-D image.) The fundamental property that a data block has is atomicity: it is assumed that, for the purpose of a CMIF-based document, each data block can not be further decomposed or sub-scheduled.

Data block descriptors are collections of attributes that describe the nature of the data block. The CMIF format makes only minimal assumptions about the types of attributes that can be defined for a given block. It does this because it does not interpret the meaning of these attributes—it simply allows them to be passed on to the required system tools that are used to manipulate a multimedia document. Example attributes may be structure information on the data block (its format, its resolution, its length, the resources required to support it, etc.) Note that a database management system may be used to locate and access various data blocks based on the attributes in the data descriptors. This possibility is illustrated by the shaded region in the diagram.

Event descriptors provide a collection of attributes that describe how a single instance of a data block is integrated into a multimedia document. The attributes in the event descriptors allow synchronization information to be expressed in the document. They also provide a means for describing that subset of data descriptor attributes that are required for the efficient access and manipulation of the data block. Note that the primary difference between the data descriptor and the event descriptor is that the event descriptor can be used to define multiple uses of a single data descriptor. Each of these items are discussed further in section 4.

A CMIF description consists of the mapping of event descriptors onto one of a set of *synchronization channels*. Each channel describes how data of a single medium is manipulated in the document. It is possible to have several channels of the same medium type; all data of a type may also be placed on a single channel. The principal role of the channel is to provide a mechanism for event synchronization. Events that are placed on a single channel are synchronized in linear time order, with the start of the second of two events occurring at a (possibly constrainted) time after the completion of the first. Two events that are placed on separate channels may be executed in parallel, either simultaneously or at a (possible constrained) time interval offset relative to each other.

Synchronization information is encoded in terms of *synchronization arcs.* Each arc is a directed connection between two event descriptors, under the convention that the arc is drawn from the controlling event to the controlled event. Each arc has a set of synchronization attributed associated with it; these attributes indicate if the synchronization is strict or approximate. It also allows for the expression of synchronization ranges so as to account for different implementation environments. Synchronization arcs can be placed at the beginning of an event or at the end of the event. If detailed synchronization is not required, then the synchronization arc can be omitted from the description. In this case, event blocks on a channel follow one-another in some system convenient manner and events on different channels have an implied synchronization that is derived from their relative placement.

Figure 3 gives a schematic view of the document structure framework. This view is similar to one that would be expected in a document structure editor tool, although the view here has been simplified for purposes of illustration.

The discussion in this section is intended to provide an introduction to CMIF. Before presenting a more detailed view of the format, we first relate the overall goals of our work to other research in this area. We then present a short description of a simple example that can be used as the basis for a second look at CMIF.

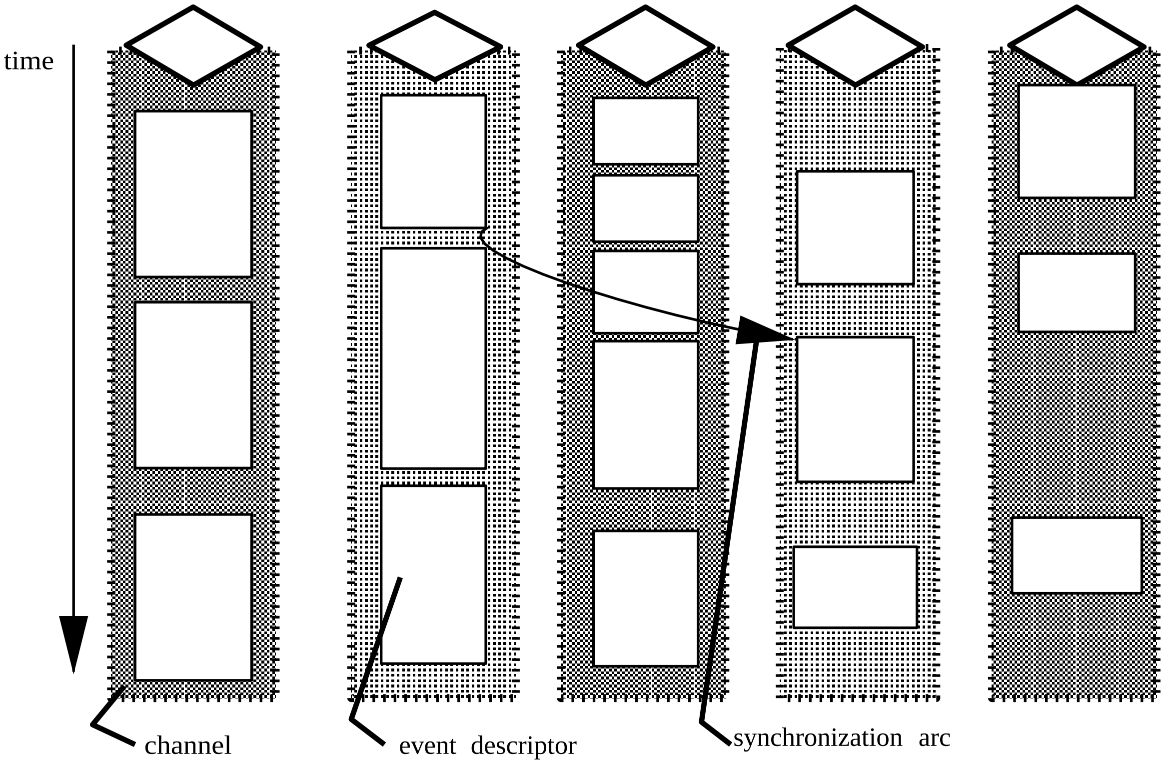


Fig. 3: Document structure components.

3.2. Relationship of CMIF to Other Formats

The main contribution of CMIF is that it provides an explicit means of describing the synchronization information in a multimedia document. It also allows for a clean separation of the various types of attributes required to differentiate the nature of a document from the definition of its components. CMIF is similar in purpose to a number of other document formats, although it differs from some of them in fundamental ways. One example is the Diamond project [Thomas85], where the use of a document structure is limited to the expression of textual and graphical data without explicit time constraints. Another example is Muse [Hodges89], where a time line concept is employed for synchronization, although Muse uses this solely to describe video sequences. There are also commercial document structure formats, such as the MIF format of Frame Technologies' FrameMaker [Frame89]; in general, these formats (while broadly defined) address only the issue of document structure rather than the specification of interaction and interaction constraints. Other formats, such as those for encoding commercial video or audio data, typically do not provide a method for coordinating different data types. This is also the case with research-oriented data descriptions (such as the image transfer format used by Dean, et al [Dean90]) for particular applications areas. Each of these formats are useful *within* a document, however; it is possible that a data descriptor attribute list may include a data encoding field that allows for the specification of a data block in a well-accepted format. This practice is encouraged even though the formats themselves are orthogonal with respect to each other.

Another point of comparison is the relation of our structure to that used in hypertext or hypermedia systems [Halasz90, Yankelovich89]. The entire question of hyper access to data is intimately related to the concepts of document presentation synchronization. Unfortunately, since hyper navigation implies a non-linear ordering of data, it is difficult to directly express all possible synchronization paths that can potentially emanate from a particular event descriptor. One approach that seems to address this problem in an interesting fashion is presented in HyTime [Dean90]; HyTime is a hypermedia document description language that includes the notions of time constraints along with hyperlinks for data. While we suspect that this general problem can be addressed via the definition of conditional synchronization arcs that point to events on separate channels, we have not developed these ideas in sufficient detail to discuss them here.

4. An Example: The Evening News

As an example multimedia document, consider a (pre-created) version of the evening television news. The structure of this document is relatively straight-forward: the news is divided into a number of separate program blocks, each of which consists of spoken text, a main video stream (showing the announcer's head or the usual dramatic on-location scenes), one view of a static background graphic illustration, and one labelling text stream (for identifying the composite screen image). If we further assume that a text-string is synchronized with the presentation for providing either multi-lingual broadcasts or captioning for the hearing impaired, then we begin to see how a dynamic multimedia document can be constructed for a collection of synchronized components.[1]

We find the evening news an interesting sample document for several reasons. First, it is an example that is broadly familiar; while individual customs may dictate order and content, the general structure of a TV news broadcast hardly varies from one country to another. Second, the contents of a news broadcast provides a collection of interesting synchronization problems that can be used to illustrate properties of the CMIF format. Examples of this synchronization include start synchronization across all blocks at the beginning of a story, block synchronization between the video and audio channels, offset synchronization between the graphic channel (where a map or an illustration may be placed) and the audio portion of the news, block synchronization between the visual blocks and the caption text, and synchronization between the placement of a label block and several of the other channels in the application. Note that we will define in more detail how synchronization is specified in CMIF later in this paper; this section is simply used to introduce the general concepts.

Figure 4 contains two views of one program block in the news. Assume that the image is a fragment out of a report on paintings that were stolen from a local museum. Fig. 4a shows a possible TV image and figure 4b provides a high-level view of the structure of a program block document. The broadcast in 4a is divided into five channels: one for the main video stream (showing the reporter presenting the story), one for the sound stream (shown as coming from the side of the display), one for the graphic frame (in this case, showing an image of the painting just stolen), one for the labelling frame (used to identify the story for those just tuning in) and one for the captioned-text string (in this case, presenting an English translation of the Dutch text coming through the speakers). The structure view of the document in 4b consists of blocks that identify the various events in the document. It further consists of a number of data blocks and several timing arcs which provide synchronization information. Note that, under normal circumstances, time moves from the top of each block toward the bottom and that blocks are either explicitly synchronized via timing arcs or implicitly synchronized by virtue of their relative placement.

[1] We include this example for purposes of illustration only. For an application system that address this issue more fully, see [Hoffert91].

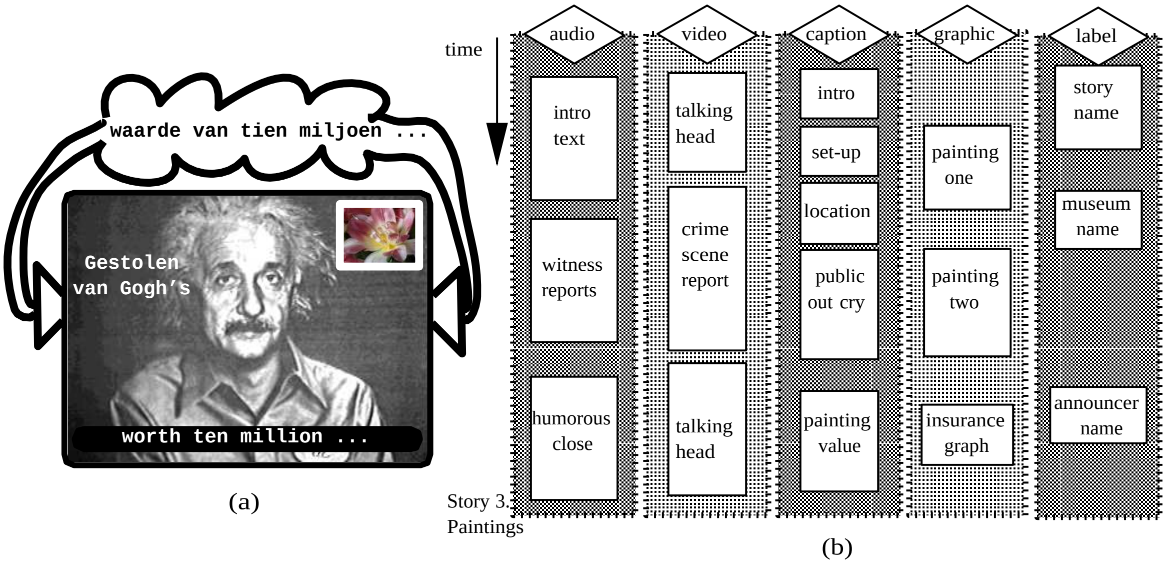


Figure 4: The Evening News as a document (4a) and as a CMIF template (4b).

For purposes of our example, we impose a few restrictions on the news: first, all data fragments are assumed to be pre-formatted. (That is, there is no notion of a spontaneous discussion among the data components and—perhaps more importantly—the length of each of the segments is known in advance.) Note that this restriction is not inherent to the document; we make it for purposes of simplification only. Our second restriction is that the synchronization among blocks is fully described by the components of the document structure. While it is possible to alter the rate of presentation (such as freeze-framing or using slow-motion), it is not possible to alter the order of events within the document by viewing it—re-ordering requires re-editing the document.

In an actual news example, each of the data blocks would need to be created via a creation tool (one for each medium) and several other tools need to exist to handle the allocation of resources to the application. We will disregard these activities for the moment, although we will return to them later in the paper. (An example of the tools necessary can be found in our description of the CWI/Multimedia Pipeline in section 2.)

In the following sections, we return to the basic building blocks of our document structure. We will use the example described in this section to illustrate the nature of each building block and to discuss synchronization issues in the document structure.

5. CMIF: A Second Look

Up to this point, we have described a document in CMIF format as consisting of a collection of data blocks that are accessed via a set of data descriptors. Each of the data descriptors contains the general attributes that describe the data in a manner consistent with the requirements of the users of that data. An event descriptor describes one particular use of a data block. Event descriptors must therefore contain information specific to a particular instance of the occurrence of a data item, including its relationship with synchronization information.

While this general description is correct, it does not fully describe how a multimedia document is structured internally by CMIF. A more complete description starts with the realization that very little of the information described within CMIF concerns actual multimedia data. Instead, CMIF defines a document tree that is used to encode the hierarchical and peer relationships among document events. The tree is a human-readable document that can be passed from one location to another with or without the underlying data. It can be analyzed by the various tools described in section 2 efficiently before any processing of information starts.

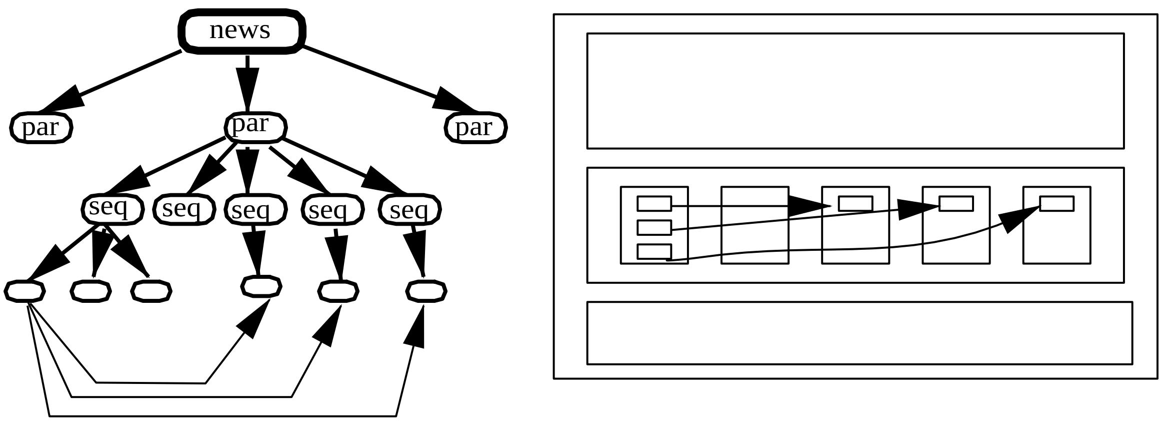
The CMIF tree can be represented as a conventional collection of nodes and branches or it can be represented as an embedded structure. Figure 5 provides a representation of these two views. Each tree consists of a collection of nodes, attributes and synchronization information. Each of these items will be considered in the following sections.

5.1. CMIF Nodes

At the root of the tree is a general node that describes the summary structure of a document. This node points to other nodes, each of which in turn point to still other nodes, until finally a leaf is reached that contains a pointer to a data block. The root node has a special function in the tree because it is a place where various directory attributes are found (see below) and because it provides an implied timing reference point for all other nodes in the document.

Each node in the tree can be one of four types:

- *Sequential Node:* each of the child nodes emanating from a sequential node is executed sequentially in a left-to-right order. The children may themselves be either data or complex nodes (that is, either leaf nodes or other sequential or parallel nodes). The parameters of sequential synchronization may be implicitly or explicitly defined. (See below.)
- *Parallel Node:* each of the child nodes emanating from a parallel node is executed in parallel with all of the other children of this node. The children may themselves be either data or complex nodes. The parameters of parallel synchronization may be implicitly or explicitly defined. (See below.)



(a) (b)

Figure 5: The CMIF tree in conventional (a) and embedded (b) forms.

- *External Node:* this is a leaf node that points to a data descriptor (and thus to an external data block). External nodes should have (or inherit) a file attribute specifying the data descriptor containing the data. If a *slice* attribute is present (see below) only the indicated part of the file is used. Separate *clip* or *crop* attributes may specify a further restriction of the data. The external node provides an indirect reference to data, allowing the structure of the data to be described separately from the structure of the document.
- *Immediate Node:* this is a leaf node containing data rather than a pointer to a data descriptor. The data is either text (the default) or another medium, as indicated by attributes associated with the node. This node is useful for encoding small amounts of data directly in a document *or* for transporting (large amounts of) data across environments that have no common storage server.

The general format of the four nodes is illustrated in figure 6.

In terms of our TV news example, the document tree may contain a collection of sequential nodes, each of which represents one story in the broadcast. (In some cases, commercial inserts or other transition material may also be placed in the document.) Each of the stories may consist of a number of internal components, some of which may be presented sequentially (such as an introduction or a transition), while others may consist of parallel nodes. Eventually, all of the nodes will point to some (multimedia) data that needs to be presented. This data will typically be described by an external node, but it may also contain immediate data (for example, for use with label text).

5.2. CMIF Attributes

Each of the attribute fields in the node contains a pointer to a list of attribute definitions. These definitions generally contain an attribute name, followed by an attribute value. The value field depends on the particular attribute type; clearly, not all attributes will be present in all nodes. Four example attribute value definitions are:

- *ID*, which contains a character value (without embedded spaces) for the attribute;
- *NUMBER*, which contains a numeric value for the attribute;
- *STRING*, which gives a character-string (in quotes, possibly with embedded spaces) value for the attribute; and
- *value** field, which provides a (set of) pointer(s) to other attributes.

seqnode	*attribute_list*	(node →)

parnode	*attribute_list*	(node →)

immnode	*attribute_list*	(data)*

extnode	*attribute_list*	(data_descriptor →)

Figure 6: CMIF node general formats.

One requirement of attribute lists is that each name may occur at most once in each list for each node.

In general, a node can have arbitrary attributes, although for some attributes a standard meaning and format is defined. For some (groups of) attributes, a global consistency rule exists which further restricts their format and/or placement. Some attributes are allowed on all nodes; others are allowed only on certain node types or when combined with other attributes. Some attributes set properties that are ‘‘inherited’’ by children (and arbitrary levels of grandchildren) of the node on which they are set unless explicitly overridden; others only affect the node on which they are present. Finally, there is one attribute, ‘‘style,’’ which is a shorthand for placing a set of attributes on a node. A representative list of standard attributes in given in the table in figure 7.

5.3. CMIF Synchronization

The ability to describe the interactions among events is of fundamental importance within a multimedia document. While it is obvious that the begin/end relationships among events is important, it is equally important to be able to specify tolerances within timing relationships if a transportable document is to be constructed. To this end, CMIF provides a mechanism for specifying several classes of synchronization primitives within a document. The synchronization information is usually implied rather than explicit, although a facility for increasing the granularity of timing relationships is provided through the synchronization timing arcs.

We divide our discussion on synchronization over four subsections. First, we provide a description of the basic synchronization concerns supported in CMIF documents. We then describe how these can be specified in terms of synchronization arcs and synchronization attributes. Next, we discuss the issues of synchronization conflicts: what can go wrong in a document. Finally, we relate our entire synchronization discussion to the TV News.

5.3.1. General synchronization concerns.

The basic tree structure of CMIF documents imposes a default synchronization that is based on the node type of the ancestors of a data (leaf) node. Within a sequential node, a default synchronization arc exists from the starting node of the arc to its sequentially first child. There are also arcs from the end of leaf nodes to the start of the successor leaf. Finally, an arc exists from the last child of a sequential node to the end of its parent. Parallel nodes have default arcs from the parallel parent node to each of the children of that parent. Similarly, synchronization arcs also exist from the end of each of the children to the end of the parent. In a sequential node, the synchronization relationship between the source and destination of the arc is simply a ‘‘start the successor as soon as possible’’ relation. In a parallel node, the relationship across the arcs is a ‘‘start the successor when the slowest parallel node finishes.’’

The provision of explicit synchronization arcs gives a document authoring system fine-grain control over the relationships between source and target nodes. As we will see, explicit synchronization arcs allow for bounded delay times relative to a global reference point, as well as an ability to define offsets from the source at which activation can be scheduled to take place. Synchronization arcs also provide the ability to describe either a rigid or a relaxed synchronization constraint—this is especially useful for documents that need to run on diverse sets of hardware.

An explicit synchronization arc can be considered to consist of three basic elements: *reference times, minimum acceptable delays,* and *maximum tolerable delays.* Each is defined as follows:

- *Reference time:* a relative or absolute reference definition for event synchronization. Absolute reference times are specified relative to the root of the document; relative reference times are specified relative to the start or end of a controlling event. Note that a reference time that is at a relative offset of zero from the start of end of the controlling event typically

Attribute	Description
Name	This attribute assigns a name to the current node. Names are optional, and relative to their parent: no two (direct) children of the same parent may have the same name, but otherwise a name may occur more than once in the tree. Names are used by synchronization arcs to reference their source and destination nodes.
Style Dictionary	This attribute defines one or more new styles. It should currently only occur on the root node. The attribute consists of a nested set of names that identify the styles for later reference by *style* attributes. Style definitions may refer to other style definitions as long as no style refers to itself, directly or indirectly.
Style	This attribute specifies one or more styles to be applied to the current node. At runtime, each style name is looked up in the style directory of the root node.
Channel Dictionary	This attribute defines one or more synchronization channels. It should currently only occur on the root node. The names identify channels for later reference by channel attributes. Each channel definition defines the medium used by that channel.
Channel	This attribute specifies to which channel the current node's data should be directed. The name should name one of the channels defined in the root node's channel list. This attribute is inherited by children unless explicitly overridden.
File	This attribute specifies the file to be used by external nodes. It is inherited, so that multiple external nodes can refer to subsections of the same file. It identifies the data descriptor used to reference data; the data may be accessed indirectly via a database system.
T_Formatting	This attribute contains a shorthand list of various formatting parameters that specify how text material will be sent to the text formatting channel. Examples are: font, size, indent, and vspace. Note: it is wise not to use these attributes directly but to place them in a style definition; they are included here for use in exceptional processing situations.
Slice	This attribute specifies a subsection of the file to be used by an external node specifying binary data.
Crop	This attribute provides a mechanism to specify a subimage of an image.
Clip	This attribute provides a mechanism to specify a part of a sound fragment.

Figure 7: Attribute examples.

indicates a coincident parallel relationship (that is, things start or end at the same time). It usually not possible to specify a timing relationship in which the destination starts before the source, although this might be possible to a limited degree if an implementation environment supports pre-fetching and pre-scheduling of events.

- *Minimum acceptable delay time:* a period (possibly zero) that specifies the minimum delay that can be allowed in the synchronization relationship. A minimum delay of 0 units indicates a hard synchronization relationship. A negative delay represents the ability to start the target node sooner than the indicated reference time. A positive delay has no meaning.
- *Maximum tolerable delay time:* a period (possibly infinite) that specifies the maximum delay that can be allowed in the synchronization relationship. A maximum delay of 0 units indicates a hard synchronization relationship. A positive delay gives an upper bound on the permissible delay in starting an event relative to the reference time. A negative delay has no meaning.

The notion of delay times introduces flexibility in the ability to schedule nodes; this is important for transporting documents across different implementation environments. The general synchronization equation is:

$$t_{ref} + \delta \leq t_{actual} \leq t_{ref} + \varepsilon,$$

where δ is the minimum acceptable delay and ε is the maximum tolerable delay. The use of the minimum and maximum delay intervals is illustrated in figure 8.

Before considering the structure of synchronization arcs in detail, it is interesting to note that the concept of the synchronization arc maps well onto the threading concept on multi-processor systems. Default synchronization arcs correspond to *fork* and *join* operations. Bounded (explicit) arcs can correspond to forks and joins that are either synchronous or asynchronous. Alternatively, the use of synchronization arcs can be mapped to the use of remote procedure call invocations in a loosely-couple multi-processor environment.

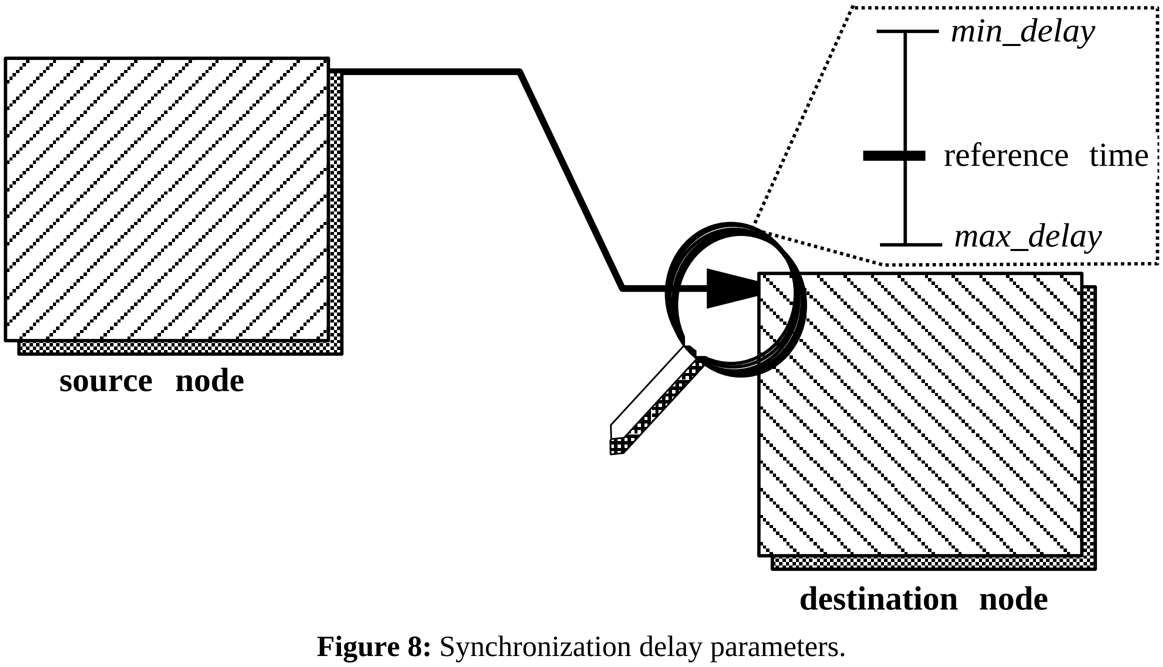


Figure 8: Synchronization delay parameters.

5.3.2. Specifying synchronization arcs.

Strictly speaking, synchronization in formation within a CMIF document is described via an *synchronization arc* attribute list. Given the importance of the synchronization arc attribute, however, we treat it separately in this section. The general structure of a synchronization arc is given in figure 9. Synchronization arcs are placed between two nodes, with the source being the controlling node. All nodes have an implied synchronization arc with the root node. Each of the fields consists of one or more attribute values that define how two nodes act relative to each other. The definition of each field is:

- *Type:* each type field has two components: an indication whether this synchronization arc concerns the beginning or the end of the event block being synchronized and an indication whether the synchronization is a ''must'' type or a ''may'' type. The meaning of begin/end is obvious. The meaning of May/Must is as follows: *May* synchronization is an indication to the implementation environment that the requested type of synchronization is desirable not but essential. This may be the case for the placement of label text at the beginning of our News example; if the label is a little late, then there is no reason for panic. *Must* synchronization is a stricter form. It tells the implementation environment that it (the environment) should do all it can to implement the requested type of synchronization, even at the expense of overall system performance. Considering the example once again, it is strictly required that each of the blocks associated with a story are presented together; the implementation environment has no freedom to delay sending one set of data if the request cannot be immediately honored.
- *Source, destination:* the source field specifies a relative path name in the tree (by using named nodes) for the controlling reference of an arc. The destination field specifies a relative path name in the tree (by using named nodes) for the target reference of an arc. The empty name specifies the current node itself.
- *Offset:* this field allows the synchronization to be defined relative to an integral positive offset from the start of the controlling node. Offsets may be expressed in terms of media-dependent units (such as seconds, frames, bytes, etc.).
- *Min-delay, Max-delay:* these are the minimum acceptable and maximum tolerable delay, as specified in section 5.3.1. These allow a document to compensate for different implementation environments.

CMIF allows absolute references to be defined by specifying an arc from the root with a minimum and maximum delay time of zero. Relative references are achieved by setting the source of the arc to be a predecessor node. Coincidental scheduling is possible by setting the source to be a peer node.

5.3.3. Potential synchronization conflicts.

There are three general synchronization conflicts that can arise in processing a multimedia document. First, an unreasonable synchronization constraint may have been defined (directly or indirectly) by a user. Second, device characteristics may limit the ability of a particular

type	source	offset	destination	min_delay	max_delay

Figure 9: Synchronization arc (in tabular form).

environment to support a given document. Third, in navigating through a document, a reader/viewer/listener may want to fast-forward (or fast-reverse) to a document section that contains a number of relative synchronization constraints for which the source or destination are not active. In general, each of these conflicts fall outside the scope of a document definition; they will be more appropriately handled by either user interface tools or document construction tools. Even though the document structure simply acts as a messenger in delivering documents for later processing, there are still several aspects of the document definition that can aid in detecting (if not correcting) the three conflict situation.

In the first case, an ‘‘incorrect’’ specification may have been made of timing relationships between two events. In the News example, the writer of the captions block may have a constraint that text must be displayed long enough to be readable by a user who is also interested in looking at the graphic material. In this case, a content-based checking tool may be required to signal conflicts. The document structure can assist in this task by providing a separation of the attributes that describe a data block from the block itself. Obviously, if the attribute information does not include timing information, there is little the document structure can do to help.

The second case is similar to the first, except that here the problem lies with the presentation of information rather than the contents of the information being presented. Here the solution is more straightforward. A local-constraint tool should be able to flag the conflict by studying information in the synchronization arcs. The implementation environment can then make a decision on whether or not to support the entire block or to perform a cut-off function or perhaps a ‘‘stretch’’ function on other data in the system. Once again, CMIF plays a role in signalling problems, allowing other mechanisms to provide solutions.

The third problem concerns itself with the implementation of the logical structure of the document. Because an internal tree is used to describe the data, the parents of a synchronization node can be traced until the common ancestor containing the source and destination of the arc is found. We support the general notion within relative arcs that the source of the arc must execute in order for a synchronization condition to be true; if this is not the case, all incoming synchronization arcs are considered to be invalid.

5.3.4. A synchronization example.

In order to bring together several of the concepts discussed above, consider the (contrived) fragment out of our newscast on stolen paintings presented in figure 10: In this example, each of the five synchronization channels presents a part of a complete story: the audio channel carries the announcer’s voice with information (in Dutch) on the robbery; the video channel initially shows the announcer, then gives a view of the scene of the crime; the graphic window shows views of three of the missing paintings; the captioned text presents a translated version of the announcer’s text; finally, the label field identifies the general scene with an occasional title.

In our example, the graphic channel is synchronized with the start of the audio portion of the report. Within the graphic channel, each illustration is sequentially synchronized. There is implied synchronization between the first and the second illustrations of the paintings and explicit synchronization among the second and third.

The captioned text is start-synchronized with the video portion of the display. (It is not synchronized at all with the audio; this allows one story to be presented for local consumption and another for global presentation.) A synchronization arc is drawn from the end of the second caption block to the start of the second graphic; this illustrates the use of an offset within an arc. At the end of the fourth caption block, an arc is drawn to the video portion to indicate that a new video sequence may not start until the caption text is over. This may require a freeze-frame video operation to support the synchronization.

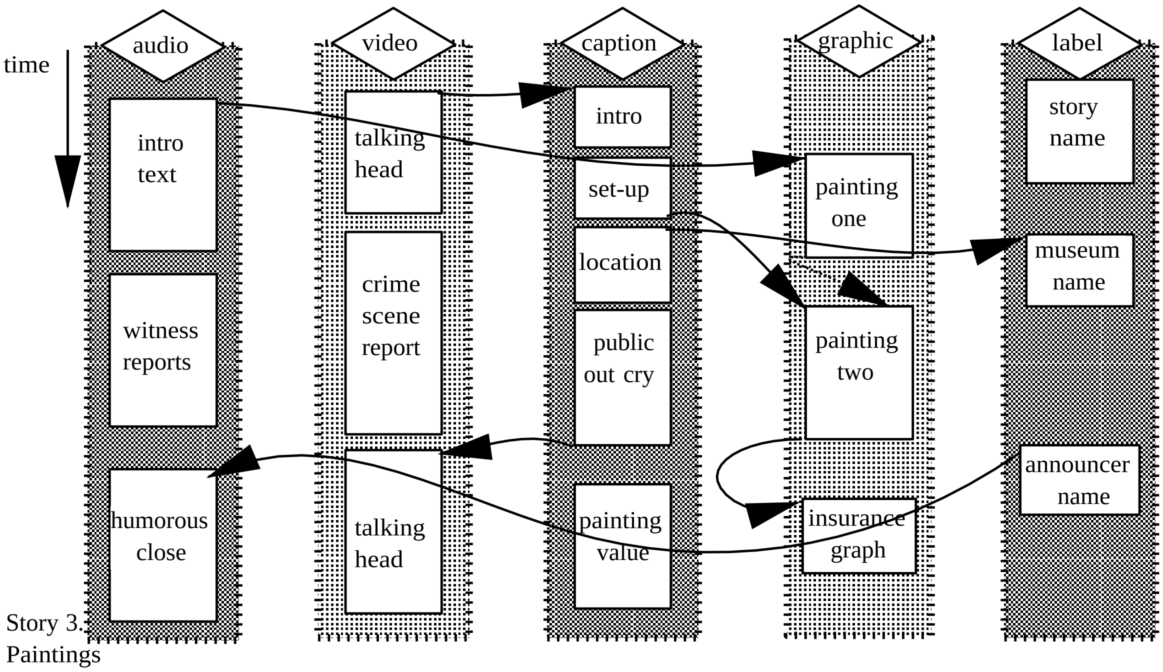


Figure 10: News report fragment structure.

Finally, the label channel occasionally displays generic titles that are linked to other portions of the display.

6. Research Directions

The sections above have provided an introduction to the CMIF structure that has glossed over a number of interesting problems that must be dealt with in a general multimedia environment. We will briefly review these problems in this section.

One of the first issues encountered in the specification of a transportable document is the resolution used in the definition of elements such as delay times, sound coordinates, sampling frequencies, etc. This is a particular aspect of a much wider problem: the specification of system-independent attributes for a document. While the temptation is great to minimize the number of attributes provided with either a data descriptor or an event descriptor, it is typically better to include as many attributes as is possible. This can be efficient for an application program (such as a constraint filtering tool) because such a tool will be given as much information as is possible upon which to make performance-related decisions. It is true that the support for a large number of attributes can make documents appear to be structurally heavy. This may give the perception that layers of attributes make a document large (and thus inefficient) as well as difficult to interpret. We do not believe this to be true for two reasons. First, by selecting appropriate attributes, much of the work associated with manipulating a document can be based on relatively small clusters of data (the attributes) rather than the often massive amounts of media-based data itself. Second, although we have created CMIF documents to be human-readable, our expectation is that the documents themselves will be created and viewed using appropriate user interface tools. The use of many attributes can also serve to make a document more understandable even without a sophisticated viewing tool; unlike text strings, for example, most people find it difficult to meaningfully interpret bitmaps or audio sequences based on uninterpreted data encodings. (One could almost consider attributes to be an efficient document documentation technique.) In all cases, however, the difficult part of attribute support remains the selection of appropriate attributes to cover the needs of a wide range of media and implementation environments.

Given the selection of an appropriate attribute set, an intriguing question is the degree to which the attributes can be used to manipulate documents in a manner that is totally independent of the data itself. For example, if the attributes contain search key information, then many time consuming activities relating to finding detailed information in large multimedia database may be simplified. Another example is the use of attributes to include content-based links to other data structures. In both of these cases, the attribute list can be used as an interface between the raw data contained in a database and the encoded information manipulated by an application. Note that while the organization of a particular database is not necessarily related to the structure of a document using that data, an investigation into the ways in which manipulation of information can be made more efficient requires substantial coordination between data/event descriptors and the data management system.

One problem that is related to the two issues above is management of the location of data in a transportable document. While it may occasionally be necessary to move massive amounts of information from one computer to another (especially in situations where the computers are not directly interconnected), we also feel that the use of both distributed databases and distributed operating systems support is vital to the efficient implementation of multimedia systems. As with common documents (i.e., computer files), the value of document sharing and multiple access to information is vital to the effective sharing of information.

Each of these areas are receiving attention at CWI. In particular, we are investigating the use of the Amoeba distributed operating system [Mullender90] as a base for a distributed multimedia system, with integrated support for a distributed database mechanism to manage document storage across the multimedia environment. The selection of appropriate attributes and the efficient manipulation of document descriptions rather than document data is of primary concern to use. The over-all goals of this project are described in [Bulterman90].

7. Summary

The key to our approach of transportable document structure is in differentiating those aspects of a document that should be available in all environments from those that apply only to specific environments. We do not dwell on storage structure or on methods of encoding/compressing data for sharing in a heterogeneous environments. For us, a much more interesting problem is defining a document that can be used to control the processing of information rather than being a slave to that information itself. In doing so, it was important to classify the notion of time and the manner in which events took place relative to each other. It was also important, with an eye towards future research, to be able to build an abstraction for a data block that could then be manipulated separately from the (large) volume of data that such an abstraction represents.

8. Acknowledgments

The general ideas presented in this paper have benefited from discussions with many people at CWI and with several of our academic and industrial contacts. Dik T. Winter of CWI contributed substantially to a refinement of the presentation on synchronization issues. Lynda Hardman of OWL, Ltd provided a detailed and constructive review of the final manuscript. The Einstein illustration used in figure 4 was taken from a public-domain GIF file, and the iris illustration in the same figure was obtained from an image database provided by Silicon Graphics, Inc.

9. References

[Bulterman90] *Bulterman, D.C.A.,* ‘‘The CWI van Gogh Multimedia Research Project: Goals and Objectives,’’ CWI Report CST-90.1004, 1990.

[Dean90] *Dean, Mascio, Ow, Sudar, and Mullikin,* ‘‘An Image Cytometry Data File Standard,’’ Lawerence Livermore National Laboratory report, 1990.

[Frame89] *Frame Technology Corp.,* ‘‘The Frame MML Reference Manual,’’ Documentation accompanying the FrameMaker 2.0 Publishing Software, Frame Technology Corp., San Jose CA, October 1989.

[Halasz90] *Halasz and Schwartz,* ‘‘The Dexter Hypertext Reference Model,’’ NIST Hypertext Standardization Workshop, Gaithersburg MD, 1990.

[Hoffert91] *Hoffert and Gretsch,* ‘‘The Digital News System at EDUCOM,’’ Communications of the ACM, Vol. 34, No. 4 (April 1991).

[Hodges89] *Hodges, Sasnett, and Ackerman,* ‘‘A Construction Set for Multimedia Applications,’’ IEEE Software, Vol. 6, No. 1 (January 1989).

[Kipp90] *Kipp, Newcomb, and Newcomb,* ‘‘HyTime Review: Standard for Hypermedia/Time-Based Document Interchange,’’ Submitted to Communications of the ACM for publication, 1990.

[Mullender90] *Mullender, van Rossum, Tanenbaum, van Renesse and van Staveren,* ‘‘Amoeba: A Distributed Operating System for the 1990s,’’ IEEE Computer Magazine, Vol. 23, No. 5 (May 1990).

[Rossum91] *van Rossum, van Liere, Winter and Bulterman,* ‘‘The CWI Multimedia Interchange Format (CMIF) Reference Report,’’ CWI Report VCST-91.502, 1991.

[Yankelovich89] *Yankelovich and Kahn,* ‘‘A Hypermedia Bibliography,’’ Brown University Institute for Research in Information and Scholarship (IRIS) report, 1989.

[Thomas85] *Thomas, Forsdick, Crowley, Schaaf, Tomlinson, Travers, Robertson,* ‘‘Diamond: A Multimedia Message System Build on a Distributed Architecture,’’ IEEE Computer, December 1985.