THE AMSTERDAM HYPERMEDIA MODEL:

On the surface, hypermedia is a simple and natural extension of multimedia and hypertext: multimedia provides a richness in data types that facilitates flexibility in expressing information, while hypertext provides a control structure that supports an elegant way of navigating through this data in a content-based manner. Unfortunately, the concepts that apply to collections of static information do not all translate well to complex collections of dynamic information. What does it mean, for example, to follow a link in a hypermedia presentation when the source node consists of nonpersistent data? Does the source presentation freeze, does it go away, do parts of it go away? Similarly, how should the synchronization relationships within and among elements in a composite component be defined? Is this part of the hypermedia model or part of a data storage or data presentation model? In our view, a general hypermedia model needs to be able to specify both the conventional link-based navigation elements of hypertext and the complex timing and presentation relationships found in multimedia presentations. Such a model is presented here.

Intuitively, support for hypermedia can be defined by taking the Dexter Hypertext Reference Model [7, 8], and augmenting it with multimedia data types within its storage layer. The Dexter model provides a facility for creating links within a document, with the links being anchored inside document components. The notions of links, anchors and components are basic to hypertext systems, and since they are not tied to any particular type of implementation or to any particular type of data, integrating them into a hypermedia system would seem to be a straightforward task. (Note that this does not make the task easy: finding a general way to indicate the presence of a link in a dynamic data, or providing means of "clicking" a portion of video or audio, remain difficult and unsolved problems.) Although such a "marriage of convenience" can provide immediate results, the underlying control assumptions of the Dexter model make it unsuitable for describing and supporting generalized hypermedia documents.[1] In particular, this approach cannot adequately support complex temporal relationships among data items, specifications that support high-level presentation semantics, or a notion of "information context" that specifies global behavior when a link is followed—all elements that are of fundamental importance in supporting hypermedia.

This article presents the Amsterdam Hypermedia Model (AHM), a general framework that can be used to describe the basic constructs and actions that are common to a wide range of hypermedia systems. We present the AHM in the context of the Dexter model's hypertext technology and terminology. We do this because hypertext is—in the time scale of information technology—a relatively mature discipline, with a well-defined model of general system behavior. The AHM is developed as an extension to the Dexter model in order to capitalize on its contributions to understanding hypertext systems. (In particular, the Dexter model allows the composition of hierarchical structures, the specification of links between components, and, through the use of anchors, allows data-dependent information to be recorded separately from the hyperstructure.) The AHM extends the Dexter model by adding to it the notions of time, high-level presentation attributes and link context. A portion of this extension is based on the basic aspects of the CMIF multimedia document model [3, 4]. By combining the relevant aspects of the Dexter and CMIF models, we are able to take the existing notions of hypertext presentations and add to these the implementation-independent behavior model of multimedia found in CMIF.

We start our consideration of hypermedia systems with a discussion of the basic requirements for a hypermedia model. We then present the AHM, considering both the general model structure and its implementation implications. We conclude with a discussion of the current state of our hypermedia implementation experiments. (Throughout our discussion, the body of this article will refer to a running example of a hypermedia presentation; this example is described in the sidebar "A Hypermedia Example.")

Requirements for Describing Hypermedia

The use of dynamic media is not unique to multimedia systems. The inclusion of time-based information (such as audio or video data) can also be supported within conventional

[1] We use the following conventions throughout this article to describe hypermedia: a *document* is a complete collection of related *components*. Each component can be built recursively from other components or from primitive *data elements* of various types, also called *entities*. A *presentation* is the active form of a document. In normal use, the terms document and presentation are nearly interchangeable, as are (to a lesser extent) entity and component. Generally, context should clarify usage.

ADDING Time AND Context TO THE DEXTER MODEL

hyper MEDIA

LYNDA HARDMAN
DICK C.A. BULTERMAN
GUIDO VAN ROSSUM

hypertext. While rudimentary support for "time" is not difficult to retrofit into existing systems, the development of a more general model for managing the elements in a presentation will be required, since more applications make use of loosely coupled collections of dynamic and static objects fetched from potentially distributed sources, all of which need to be synchronized in some nontrivial manner. To put the requirements of such a generalized model into context, a brief overview of the defining characteristics of hypertext, multimedia and hypermedia is given followed by the requirements for modeling time, links (and link context), and global presentation semantics in a hypermedia.

Hypertext, Multimedia and Hypermedia

Figure 1 provides a high-level review of the essential aspects of the hypertext, multimedia and hypermedia models. In Figure 1a, a hypertext is modeled as a network of components related through a set of links anchored in source and destination components. While various implementations of hypertext may impose different constraints on the internal structure of a component or the exact nature of a link/anchor set, all systems support the notion of "visiting" a component for a user-determined amount of time, with that visit either terminated by the end of the application or interrupted/replaced/augmented by following a link to one or more other components. Note that the meaning of visiting a component—that is, the visual effects displayed to the user in terms of pieces of text, graphics, sounds, and so forth—is usually considered an internal property of the data.

Figure 1b illustrates a generic multimedia presentation. Like Figure 1a, the presentation is made up of a collection of components. Unlike Figure 1a, the components are meant to be presented in some author-defined relative order. The existence of such an ordering relationship depends on an explicit notion of time in the model. While the user still may have control over the selection of components to be visited, the components selected and presented can change without direct user intervention because of this notion of time. (That is, the model not only defines the components of the presentation, but it also defines an ordering relationship that specifies when the components are presented relative to one another.)

Multimedia systems typically support two types of navigation facilities that provide a user with control over the presentation. One method adjusts the current time reference in a presentation, indicated by the heavy horizontal line in Figure 1b; by using a control interface similar to that of an audiocassette or compact disc player, the user can stop/start/fast-forward/rewind (and sometimes search through) the presentation. The second—and less common—navigation type is similar to a rudimentary hypertext link, indicated by the anchor and link arrow in the figure. Here, following the link would bring the user to the time point that is indicated by the dotted line in the illustration. This is essentially equivalent to a fast-forward operation, except that the 'stop point' is defined by the document author rather than by the user.

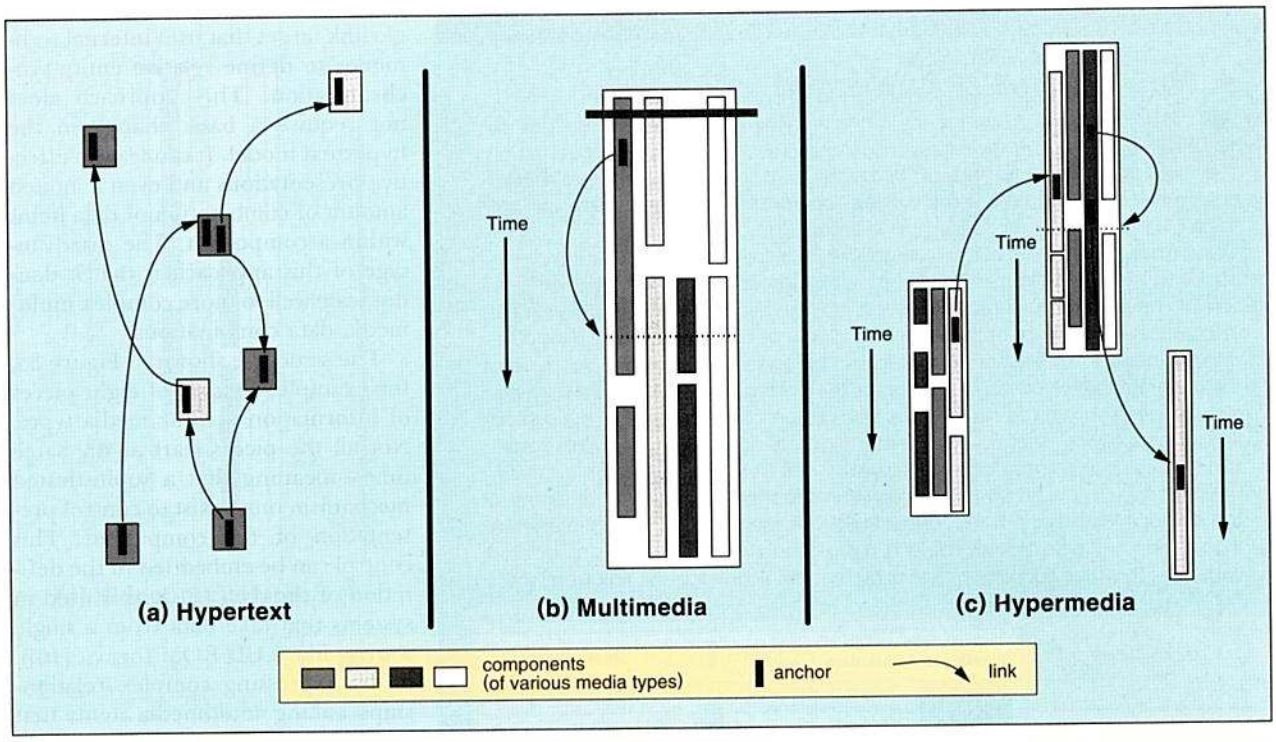
Figure 1c gives a high-level description of one way of combining hypertext and multimedia: by having each component of the hypertext model be a self-contained multimedia presentation. This model addresses two sets of concerns: those that relate to the hyperstructured navigation through the document, and those that relate to the multimedia presentation of information. For many simple forms of hypermedia support, the sketch in Figure 1c presents an adequate model of system behavior. As we will see, however, this view limits the flexibility of the author in defining a presentation and the user in viewing one.

Temporal Information

In general, a hypermedia model should be able to specify how individual pieces of information relate to one another at any level that a document author feels appropriate. This level would depend on the way the data was stored and on the presentation/navigation abilities of the runtime systems available to users. The specification of timing constraints within a document thus depends on the nature of the underlying data elements and on the way these can be combined for presentation. The internal structure of data will be beyond the scope of the hypermedia model, but the composition of components remains central to the model.

In conventional hypertext, time is addressed indirectly, often in terms of a presentation's behavior as it follows a hypertext link. Such an approach is unfortunate because it binds together two separate aspects of a presentation. Instead, we partition the temporal relationships among data items in two broad classes: those related to the identification of the components that are to be presented together, and those that relate to the relative order in which these components are presented. We call these classes *collection* and *synchronization*. At a coarse level, the Dexter model provides support for collection via the hierarchical definition components. As is illustrated in Figure 2, Dexter's composite components can be used to collect a set of atomic components that are to be presented together. (In terms of the Amsterdam tour example described in the sidebar "A Hypermedia Example," we could define a composite component that consisted of the various data entities that make up each screen.) Since the definition of the composite does not provide a mechanism for specifying any relative timing relationships among the entities, however, this approach is not sufficiently rich for general use. The problem to be overcome is that, once collected, the components need to be synchronized, typically both *within* and *across* components. Such synchronization can be based on structural information—that is, by manipulating data representations within components—or it can be based on the content of a component.

Consider the lower-left screen in the Amsterdam tour example. This view of street musicians can be defined as a (possibly structured) collection of files/objects/database entries that make up the components in the screen. These elements could be synchronized based on the author's



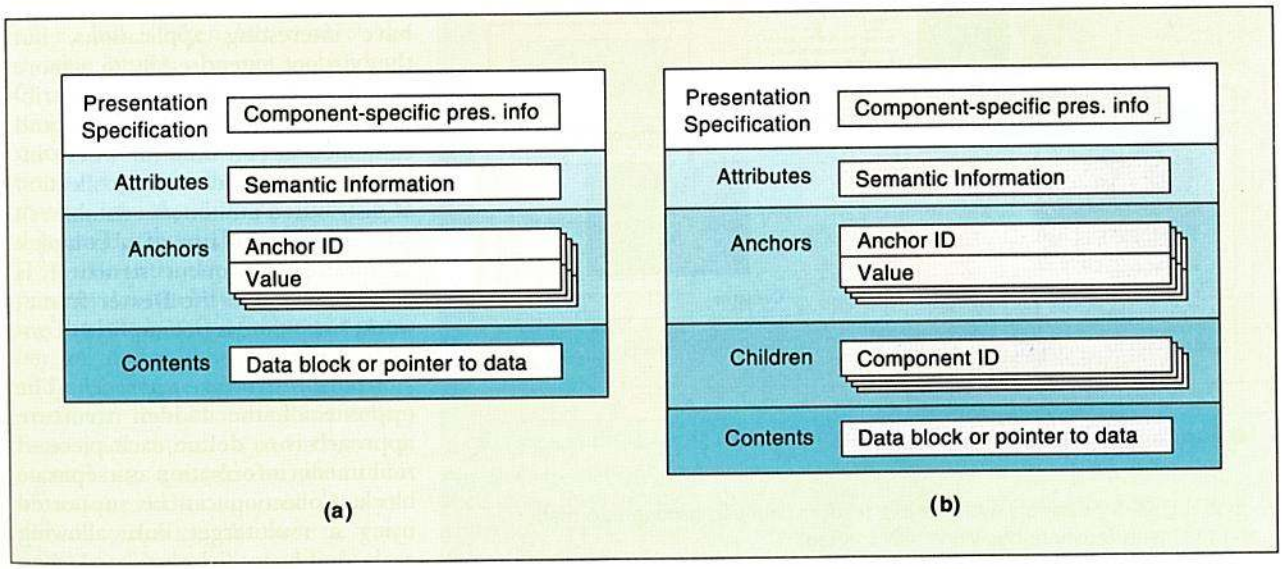


Figure 1. Hypertext, multimedia and hypermedia

Figure 2. Dexter model atomic (a) and composite (b) components

(or authoring system's) knowledge of the sample and frame rates of each item. They could also be synchronized in a content-based manner, such as specifying that the two elements needed to end at the same time, or by specifying that a cymbal crash in the soundtrack needed to be synchronized with the associated event in the video segment.

There are several ways to approach the problems of collection and synchronization within hypermedia. Three general strategies are shown in Figure 3: we call these the *hidden structure*, the *separate structure* and the *composite structure* approaches.

Hidden structure approach. The most basic (and most prevalent) way to handle time-based data in a hypertext context is to place all the data—and all the data interpretation—inside the content portion of a component. In Dexter terms, this solution pushes multimedia information into the within-component layer. Collection (in terms of defining a set of entities that are to be shown at the same time) is solved by having a sin-

A Hypermedia Example

Hypermedia presentations can take many shapes and forms. They can be models of flexibility, enabling the user to pick and choose among the data items presented, or they may be rigidly structured presentations that mimic—perhaps too closely—broadcast television, where the user has a binary interaction option: turn it on or turn it off.

One form of a multimedia presentation is depicted in Figure A. The three "screens" shown in Figure A make up part of a tour of Amsterdam. Each screen is divided into a number of regions, with each region holding a simple or composite stream of multimedia data. Some of the regions have data containing anchors (the anchors are indicated by dotted lines around the regions).

A general table-of-contents portion of the presentation containing five regions appears in the top screen: one title region at the top-left of the screen (in this case saying "Welcome to Amsterdam"), a logo-region (containing the CWI logo), a video region (shown displaying one frame of a video on a canal house in the city), a text frame displaying a set of paragraphs on the nature of the demo, and an audio track containing supplemental information. The entire logo region and a portion of the text region contain visual buttons associated with links to other portions of the presentation. Following the link from the

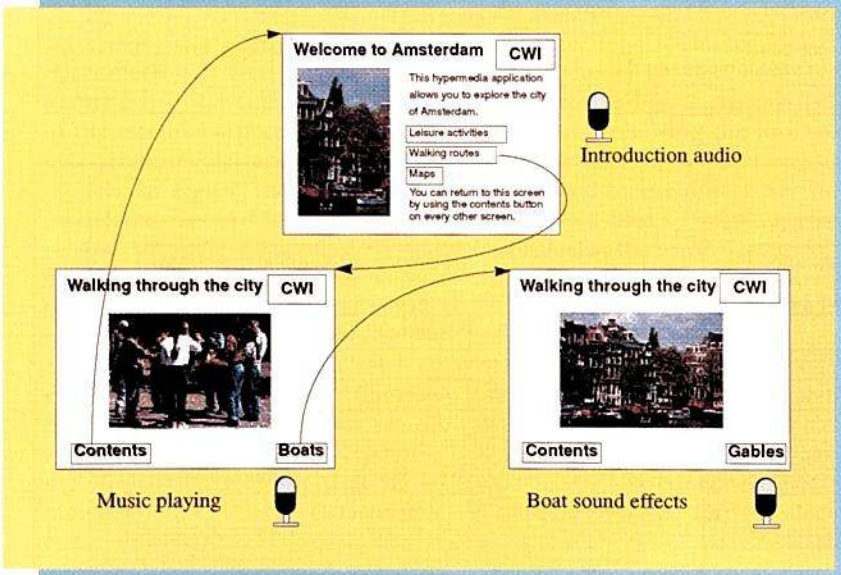


Figure A. An example hypermedia presentation

logo region, for example, will give the reader some background information on the institute where this video was created.

The lower-left screen shows a portion of the "walking tour" section of the presentation, showing a collection of street musicians entertaining commuters at the city's central railway station. This screen has a title region, a video region, an audio region and two navigation regions. One navigation button will return the user to the upper display, while the other allows a user to jump to a part of the walking tour that describes the types of boats one is likely to see in the canals. The screen at lower right is structured similarly to that at left, except that the contents of several of the regions have been changed: a new video atomic component has been used (to show boats on the canals), a new sound track is used (with boat sound effects and commentary), and a *Gables* button replaces the *Boats* button.

The presentation outlined here could have been constructed with a number of multimedia authoring systems, using a variety of underlying architectures. From a hypermedia perspective, there are a number of information structuring issues that need to be considered in describing presentations of this type. These issues, which include the logical and structural relationships of screens to one another, are discussed in the body of this article.

gle link target that uses internal techniques to define relative entity synchronization. This approach does not require a basic change in the hypertext model. It can lead to effective presentations and even a limited amount of combination of data items within a component. The disadvantage of this approach is that it does not scale well to more complex multimedia data combinations.

The structure shown in Figure 3a, for example, consists of eight pieces of information of four media types. Not all the pieces start at the same time—meaning that a sophisticated mechanism must exist to control presentation of the component. This control can be embedded in the definition of the data (such as is used in systems that take data from a single source, like a CD-ROM format [16]), or by expressing complex relationships among multimedia items that are subsequently accessed as a single entity [2]. Both of these approaches have interesting applications, but they do not extend easily to a more general case in which data is distributed over several file servers and combined at run time (or where information is stored across a collection of distributed multimedia databases). In all cases, the use of a complex composite component structure is impossible within the Dexter framework, since time is not explicitly considered in the model.

Separate structure approach. The opposite of the hidden structure approach is to define each piece of multimedia information as a separate block. Collection can be supported using a multitarget link, allowing each component to be activated when the link was followed. In terms of our Amsterdam tour example, this would mean that the top-middle picture would be made up as the target of a link containing five separate objects. When this link was selected, the runtime system would start the objects simultaneously. (Such an approach is similar to that taken by the Intermedia project [15].) While collection can be supported in this way, synchronization becomes a significant problem: in a generalized model, not all components can be expected to start at exactly the same instant. Figure 3b illustrates a combination of compo-

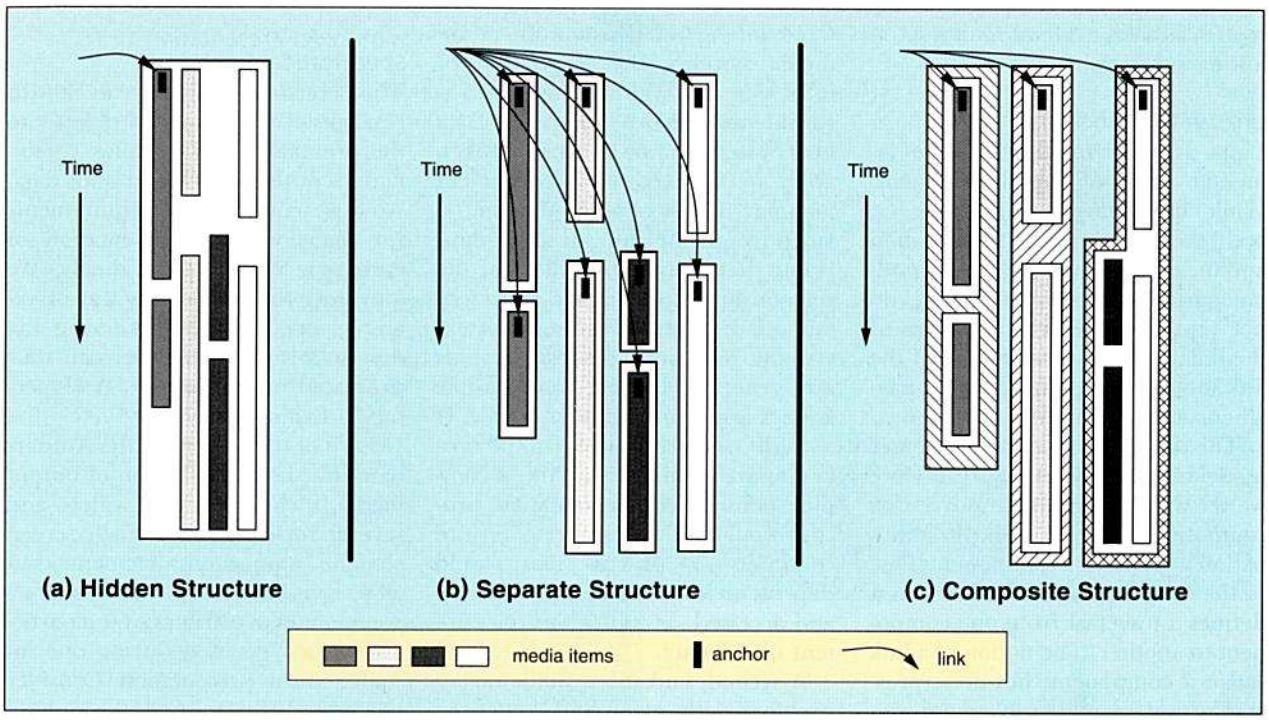


Figure 3. Time and structuring issues

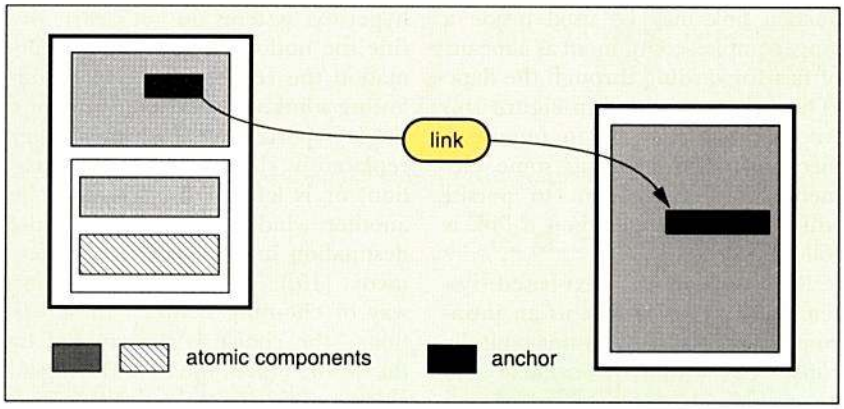


Figure 4. Dexter link components

nents that all need to be activated after a link is taken, but in which the starting times are not coincident. Clearly, some method is required to state when each component should start relative to the others. One solution is to place relative timing information into the link structure, as is done in Harmony [5] and Videobook [13]. As with the Intermedia approach, this can be useful for specifying the interaction of a limited number of nodes, but it requires the author to define and maintain information in links that combine collection, synchronization and navigation control into a single construct. This increases authoring and maintenance complexity because a great number of links will need to be created and maintained by the authoring software. These approaches also have the disadvantage that the content association among components is lost, and that any user selectivity or customization of a presentation requires complex user-interface software.

Composite structure approach. A compromise solution to the problem of collecting components and synchronizing pieces is shown in Figure 3c. Here, several elements have been grouped into three composite components, one of which hides internal synchronization and collection of two media types and two of which manage data of a single media type. Collection is less complex than in the separate structure approach, and the synchronization problem is also reduced by internalizing any complex relationships. While direct application of such an approach may prove adequate for simple applications, it does not solve the general problem of specifying the time-based relationships within the hypertext model: it simply reduces the problem to a number of subproblems, each solved by one of the two approaches described in the two preceding paragraphs. Grouping of components is a useful way of specifying collecting structured components *if* this is accompanied by a well-developed notion of time within the general

model, as will be discussed in the following subsection.

Links in Hypermedia

Links are perhaps the most fundamental notion of hypertext systems. While links have an influence on both the order of the presentation and the components displayed simultaneously, these temporal aspects do not represent the central nature of the link. Instead, the function of the link should be to define a logical navigation mechanism in a document.

The use of links within the Dexter model is illustrated in Figure 4. Here we see a link that has its source in a composite component and its destination in an atomic component. One of the basic aspects of a link is that it defines a traversal from one component to another. The notion of a link *within* a component, however, presents a fundamental problem not considered within the Dexter model. Such a link may be used inside a long, complex component as a means of fast-forwarding through the data. (This is how it is used in Figure 1b.) Another use may be to impose a mechanism for allowing some elements of a component to persist while others change when a link is followed.

For conventional text-based systems, the use of a link as an intracomponent structure is not typically considered within the context of a general model, since blocks of text can be broken into arbitrarily small collections of characters (such as books, chapters, sections, paragraphs, sentences, clauses, words, and letters). This allows new components to be created whenever a link is defined by dividing an existing component at some "convenient" boundary. (Specification of such a boundary is supported by the Dexter anchoring mechanism, which allows a general method for specifying offsets into components.) This approach works well within hypertext systems because the logical element making up the bulk of the data—the character—is closely associated with the structural element used to define the data itself, the byte. It is also aided by the relationship between two successive blocks of characters not being typically constrained by a fixed timing relationship. In multimedia systems, the association of links with and within components is considerably more complex. Data items often cannot be easily broken apart or indexed in a convenient manner. For example, dividing a video fragment into a scene-shot-frame hierarchy is possible, but the relationship of content to representation is less clean than in hypertext systems. (For films, scenes and shots are generally content-based, while frames are representation-based.) If a single video sequence is to be broken apart to support a link, a complex editing and presentation process would be necessary to ensure continuity—a process that would depend on how information is stored and accessed, as well as on the content of the data.

A second, and more fundamental problem with extending hypertext links to hypermedia is that current hypertext systems do not clearly define the notion of how much information the reader leaves when following a link. Most systems present a single hypertext node which is either replaced by the destination information, or is left on the screen while another window is created for the destination information (e.g., Intermedia [15]). There is, however, no way of choosing between these options—the choice is determined by the destination of the link. Having an either-or model is useful for text (where most readers can only focus their attention on one block of text at a time), but it is less useful for multimedia presentations, where a user can follow a link from one block to another while continuing to listen to a spoken commentary or watch a video presentation. In hypermedia, it is important that the model define a control framework so the user can specify the behavior that is appropriate to the needs of the application.

In order to address this problem, a method needs to be defined that allows portions of a component to be treated as minicomponents for linking purposes, or a new construct needs to be added to support more generalized linking behavior in hypermedia. Our solution to this problem is to introduce the notion of a *link context*.

High-Level Presentation Specification Attributes

The preceding topics have dealt with concepts that are related directly to the presentation of complex combinations of time-based and static data. Now we consider the requirements for managing the presentation of each type of data individually. We introduce the problem by way of example, once again considering the description of the Amsterdam tour presented in the sidebar "A Hypermedia Example."

Each of the screens of the Amsterdam tour uses a collection of output media, including audio, video, and several forms of text. Each screen contains unique data elements, but all screens have elements that share a common set of attributes for a particular data type used during one instance of the presentation. Consider the audio output used throughout the presentation: in each screen, different audio data sets will be used, but each screen will share attributes such as volume or tone-quality settings. Note that some of these attributes will depend on the workstation presenting the information, such as the audio quality of the local devices or the ability to support stereo/mono output. Other attributes may depend on the preferences of a particular viewer/listener to the presentation, such as the volume level of the output. Since both of these types of attributes would be defined on a per-presentation basis, not a per-screen basis, a mechanism should exist that allows their global definition.

Just as word processors allow the definition of global styles for different structures (for example, section heading or paragraph body), hypermedia documents require a higher-level way of specifying presentation information than at the per-component level. Such attributes would ensure the audio volume would not need to be reset by the user for each screen opened, or that the system's audio drivers would not need to be reinitialized for every data block sent to a device. Such global attributes would not be attached to a single component, but would instead be associated with a class of media/information types. The attribute definitions could consist of default set-

CMIFed—An AHM-Based Authoring Environment

CMIFed is an authoring and presentation environment for hypermedia documents based on the AHM [18]. CMIFed uses a hierarchy structure to represent the document; this is shown in Figure B, part (a). The use of a hierarchical document structure enables presentations to be designed where persistent objects, such as titles and logos, need to be placed only once and are retained on the screen throughout the scene (in fact, for the duration of the structure within which they are defined). This has the result that authoring work is reduced through the use of the hierarchy—items that remain for a greater part of the presentation can be defined in the higher levels of the structure.

There are two main ways of viewing a CMIF document when authoring—the hierarchy view and the channel view. The hierarchy view shows the document as a structured collection of components that are played in series or in parallel. The channel view displays the atomic components mapped onto abstract channels; it is presented in the form of a presentation time line, showing the synchronization information. A document, once created, can be presented by the CMIFed player, which maps the abstract channels to real devices based on user preference and system capabilities.

The hierarchy view. The purpose of the hierarchy view is to display and manipulate the structure of a hypermedia presentation. The hierarchical structure of the presentation is represented in the hierarchy view as an embedded block structure, shown in Figure B, part (b). Here, the rectangles enclosing the boxes represent the hierarchical structuring of the document. The outermost rectangle is the root of the tree. Components next to each other are started in parallel (unless otherwise constrained by explicit synchronization arcs), while components displayed higher in the diagram are activated before those displayed lower in the diagram.

A presentation is created by defining the structure of the presentation and assigning atomic components to the structure. Timing information is deduced from the hierarchical structure and the durations of the components within the structure. (Fine-grained timing constraints can be added in the channel view.) When a component has no explicit duration, as in a text item, it is presented for the duration of its parent. The author can navigate around the hierarchical structure and zoom in on nested structures. This not only reduces the screen space required for representing the structure, but gives a focused view on any part of the structure. Within the hierarchy view, the author can select any atomic or composite component and cut, copy or paste it, or interrogate it for more information, such as component name, data file referred to, channel used, explicit duration, comment, highlight color. An external editor can be called up to edit the data. The editing program invoked in this way depends on the media type and the choice of editor is configurable. CMIFed can read most common data formats and can be easily extended to support more.

The channel view. The channel view, Figure B, part (c), shows a transformation of the hierarchy view in terms of abstract media channels. This view is presented as a time line, with placement determined automatically by CMIFed-based on information in each component definition. The use of synchronization arcs allows fine-grained timing constraints to be specified between and among groups of components. The atomic components making up the presentation are displayed in their own channel along with their precise durations and timing relationships. If the author changes the timing in any part of the presentation, via any of the views, the channel view is updated immediately to reflect this.

The player. The CMIFed player interprets a hypermedia document and plays the presentation on the available hardware. The player allows the user to select which channels should be played, for example to select one of a number of voice-overs in different languages, as in Figure 8. The player is closely integrated with the hierarchy and channel views

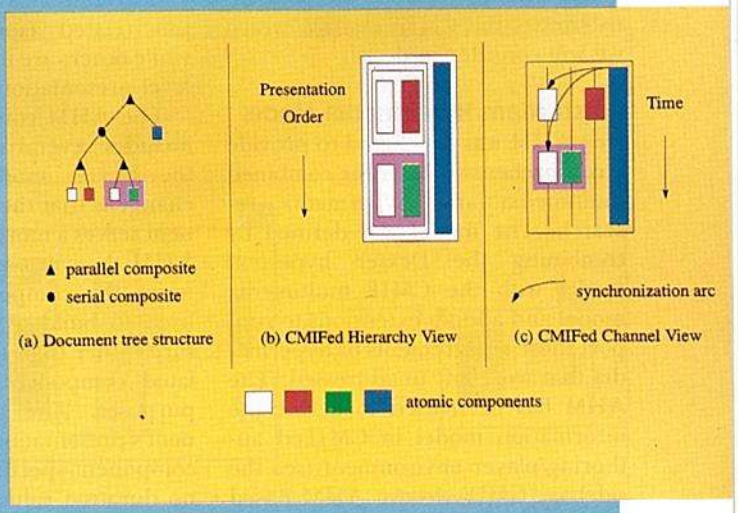


Figure B. CMIFed hierarchy and channel views

and allows the author to play a selection from the hierarchy view (atomic or composite component) or the channel view (atomic component). This facility allows the previewing of a small section of the presentation without having to start at the beginning of a long sequence.

When the system has sufficient time it looks ahead and fetches data that will be needed in the next part of the presentation. The stages of this pre-arming are shown by highlighting the atomic components in different colors in the channel view.

Implementation. CMIFed is implemented in Python [17], a high-level interpreted, extensible object-oriented prototyping language. The language has a number of practical advantages, in particular it has a good interface to the graphics facilities and user-interface toolkit on the initial target machine (the SGI Indigo workstation). Python's extensibility has been used to efficiently handle the data formats and I/O devices needed for audio, image and video processing without losing the advantages of using a very high-level language (e.g., powerful string-handling operations, shorter and clearer code, and a much faster edit-run cycle).

tings that could be overridden when necessary, or as a set of basic attributes that could be augmented on a component-by-component basis.

One motivation for providing such attributes is to support users accessing documents by allowing preference to be set once instead of at each data reference. Another is to be able to define documents having a measure of portability by abstracting presentation information that may be machine-dependent from an individual component. Although such global attributes are not considered directly in the Dexter model, they do capture some of the intent of separating presentation information for data within a component. The mechanism we use to manipulate these attributes is the CMIF *channel*, which we will consider in detail.

Amsterdam Hypermedia Model

The AHM was developed to provide a comprehensive basis for combined multimedia and hypermedia research. The model was defined by combining the Dexter hypertext model with the CMIF multimedia model and adding extensions to support those requirements of hypermedia that were left unaddressed. The AHM has been used to define the information model in CMIFed authoring/player environment (see the sidebar "CMIFed—An AHM-Based Authoring Environment"), which has been used to create several hypermedia applications.

AHM Components

Figure 5 shows the conceptual data structure for a) atomic and b) composite components. (The names and general structure of these components reflect the AHM's Dexter heritage; also see Figure 2.) The atomic component, shown in Figure 5a, contains metainformation that refers to a particular data block, while the composite component defines such information for a collection of atomic or composite blocks. Note that the composite component does not contain support for data in the composite's definition—data can only be referenced via an atomic component. This has three advantages over the Dexter approach. First, it localizes information on timing and presentation to an atomic component and concentrates information on presentation structure to the composite component; this should ease the task of document maintenance. Second, it promotes reusability of data by forcing all items to be separately maintained. Finally, it more closely models the way the bulk of multimedia information will be stored: in external databases or file systems.[2]

AHM atomic components, like their Dexter counterparts, contain presentation information, component attributes, link anchor information and a contents field. The significant addition to the atomic block is an expanded presentation information section. A portion of this expanded information is used to model time-related aspects of the block, while others are used to encode high-level presentation attributes.

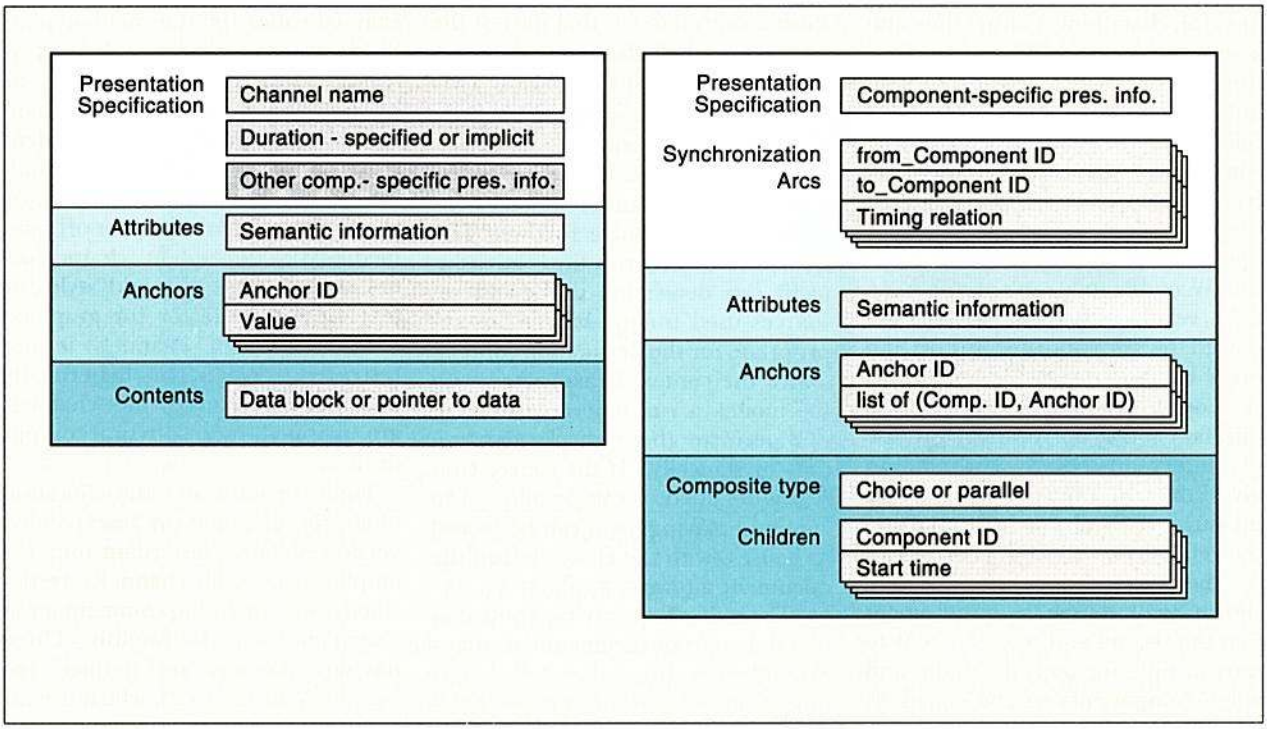
The AHM composite component includes several items not found in the Dexter model. The principal change is that the composite component serves a more specific role in the AHM than in typical hypertext systems: the composite component is used to build a presentation structure rather than to simply collect related components for navigational purposes. The composite component's presentation attributes contain component-specific information but no duration value: these can be obtained from the collection of atomic components used by the composite. Instead, the presentation specification contains a collection of *synchronization arcs*, which are structures that define fine-grained relative ordering information. (Synchronization arcs are defined later.) The anchor and attribute sections of the AHM are essentially the same as those in Dexter, except for a dereferencing of anchors to a list of <Component ID, Anchor ID> pairs. The specification of the components of the composite has been expanded to include timing offsets among the children and a *composite type* attribute. The type of a composite can be either parallel or choice; parallel composite components display all of their component parts, while choice composites will display at most one of their children. (The selection mechanism is implemented by the run-time support environment.) As noted, the composite component does not include any data directly.

[2]Individual implementations may choose to support small amounts of in-line data as an optimization; our system allows this for text blocks.

Temporal Relations

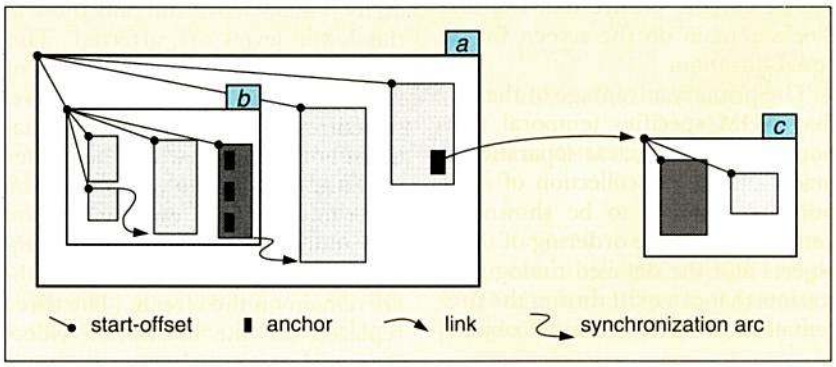
The principal mechanism for supporting temporal relationships among entities is the composite component. The definition of the composite supports collection via a list of child components, each of which may be a composite or atomic component. Synchronization among components is supported in two ways: one for coarse-grained synchronization and the other for fine-grained synchronization. Coarse-grained synchronization consists of constraints defined between the children of a composite component, such as the relative starting time of each child within the composite; this information is given explicitly with the child definition. Fine-grained synchronization consists of constraints among either sibling or nonsibling (nested) children within the composite component; these constraints are specified using synchronization arcs. Note that additional synchronization is possible within the definition of an atomic component's data; this is not considered part of the hypermedia model, but is instead a characteristic of the data object.

Figure 6 summarizes the timing control within the AHM. It shows three composite components (*a*, *b*, and *c*), and eight atomic components. Each of the atomic and embedded composite components has a start time offset, represented by the line from the upper-left of the composite definition. (Note: the length has been exaggerated for purposes of clarity.) This offset gives the relative start time of each component. Each atomic component contains a duration attribute (or an estimate, for synthesized data); this duration is not shown in the figure. Two synchronization arcs are shown in the figure, the meanings of which are discussed in the following paragraph. Figure 6 is a crude approximation of the top screen in our Amsterdam tour example: component *a* represents the



screen as seen by the user, which is made up of two top-level atomic items and one composite. The top-level atomic items of *a* represent the CWI logo and a headline text block containing "Welcome to Amsterdam." The composite *b* contains two sequential audio tracks giving an introduction to the tour, a video showing parts of the city, and a text block containing instructions and three anchors. The composite *c* contains information about our institute. (Note that the sequencing within the composite *b* depends on the ability to separately schedule events and then to constrain aspects of their run-time behavior.)

The synchronization arc (or *sync arc*) allows an author to specify fine-grained synchronization information among components [3]. It is **not** used as a navigation aid or as a type of link. It is a constraint that the run-time system should support on the behavior of two or more components. The sync arc is a construct that provides a flexible mechanism for establishing relationships in a multimedia system. Figure 7 shows the basic elements of a sync arc in detail. The end-point component IDs are given, as is the timing relation to be supported. The timing relation is given in terms of a synchronization



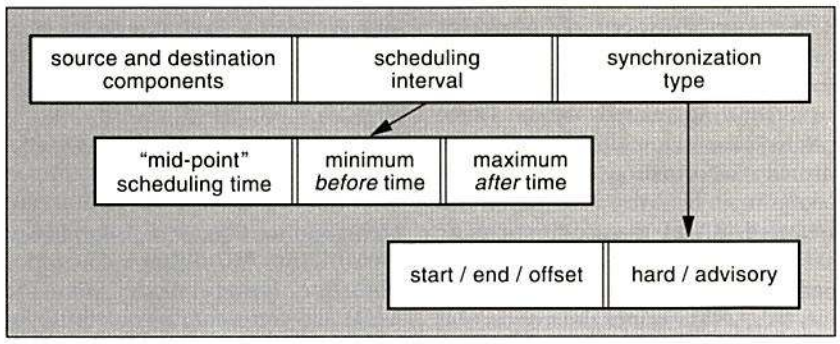


Figure 5. Amsterdam Hypermedia Model (AHM)

Figure 6. AHM components and timing relations

Figure 7. AHM synchronization

interval, containing a target time and acceptable deviations, and a synchronization type. The type consists of an indication of whether the interval is relative to the start or end of the component, or whether it is an offset from the start. The second portion of the type is an indication of whether the relationship must be met or if it is simply an advisory relationship. (If a *hard* relationship cannot be supported by the run-time system, an error condition exists; if an *advisory* relationship cannot be met, the application will keep running; the use of interval-based synchronization gives the run-time environment a measure of flexibility in supporting the relationship.) In terms of Figure 6, the intracomponent sync arc shown may define a requirement that the second audio block needs to start in time for both the audio and video components to end together, while the intercomponent sync arc can be used to ensure that two text blocks remain on the screen for an equal duration.

The primary advantage of the way that AHM specifies temporal relationships is that a clear separation is made among the collection of components that are to be shown together, the relative ordering of those objects and the detailed timing constraints that can exist during the presentation of a collection of objects.

AHM Link Context

The temporal aspects of the AHM component architecture provide a convenient method of grouping and synchronizing related objects. Composite components do not, however, provide information on how each component behaves when a link is followed out of that component or within a composite component. In order to describe this type of behavior, the AHM defines the notion of a *link context*. A context is a (typically composite) component that contains a collection of composite or atomic components affected by a linking operation. (While the Dexter model already allows the specification of composite components with a link, these are not used explicitly to define a context.) A *source context* for a link is that part of a hypermedia presentation affected by initiating a link, and a *destination context* as that part of the presentation which is played on arriving at the destination of the link.

The context mechanism allows specific display options to be associated with each link. In particular, the source context can be retained or replaced when a link is followed. If it is replaced, the run-time environment can determine if the old resources used for the source are appropriate for the destination context. (Since the context is associated with the model, a run-time environment can perform these checks dynamically or statically.) If the source context is retained, it can be allowed to continue playing, or it can be forced to pause—with the choice left to the document author.

A benefit of specifying context is that only part of the document structure needs be affected on following a link. Components of the presentation higher in the composition hierarchy remain active and only those at the lower levels are affected. This reduces the authoring burden of repeating the same higher-level structures for different presentations. For example, when the reader activates the *Boats* anchor in the lower-left screen of our Amsterdam tour, only three of the atomic components are replaced while the others remain on the screen. (The three replaced are the illustration video, the audio track and the component specifying the next subject in the sequence, in this case the *Gables* button.) The document in the figure uses only two levels but the model imposes no restriction on the depth of the hierarchy. (Contexts are discussed in greater detail in [11].)

AHM Channels: Encoding High-Level Presentation Attributes

Both the Dexter model and the AHM allow presentation attributes to be specified for individual atomic and composite components. While these attributes are sufficient for local control, they are inadequate for specifying more global attributes of a document. Such attributes are defined in the AHM using *channels*.

Channels are abstract output devices for playing the content of a component. Associated with each channel are default presentation characteristics for the media-type(s) displayed via that channel, such as *font* and *style* for a text channel, or *volume* for an audio channel. Channels store media-type independent specifications, such as background, foreground and highlight colors, or whether the channel is on or off; and media-type dependent characteristics such as font size and style for text, or scaling factor for graphics. The number of channels is not restricted. When a hypermedia document is played the channels are mapped onto physical output devices.

Figure 8 illustrates the allocation of abstract channels for an expanded version of our Amsterdam tour example. Two audio channels are defined, one containing commentary in Dutch and one in English. Three linking channels are defined (labeled *L0*, *L1* and *L2*), which are associated with the atomic components containing the text *contents*, *begin route over* and *next*. Two captions channels are defined (for holding Dutch and English text strings) and one screen channel is defined for the video data object. Note that the on/off channel attribute allows a user to specify which of the channels are active at run time; this allows a full presentation to be defined (for a variety of audiences), with particular instances of the presentation available on demand, customized to user preferences and to local hardware facilities on the presentation host. This feature allows the fragment shown in Figure 8 to be used to present either Dutch language audio and English language subtitles (or vice-versa), with the selection supported by the run-time environment. (While a presentation providing English and Dutch audio and English and Dutch captions would be possible—assuming enough screen "real-estate" and audio bandwidth existed—the usefulness of such a selection would depend on the nature of the user.)

The use of channels also allows the same document to be presented in different ways by respecifying the styles rather than by changing the presentation specification for every item. Just as in paper documents, this encourages consistency throughout the presentation. Flexibility is main-

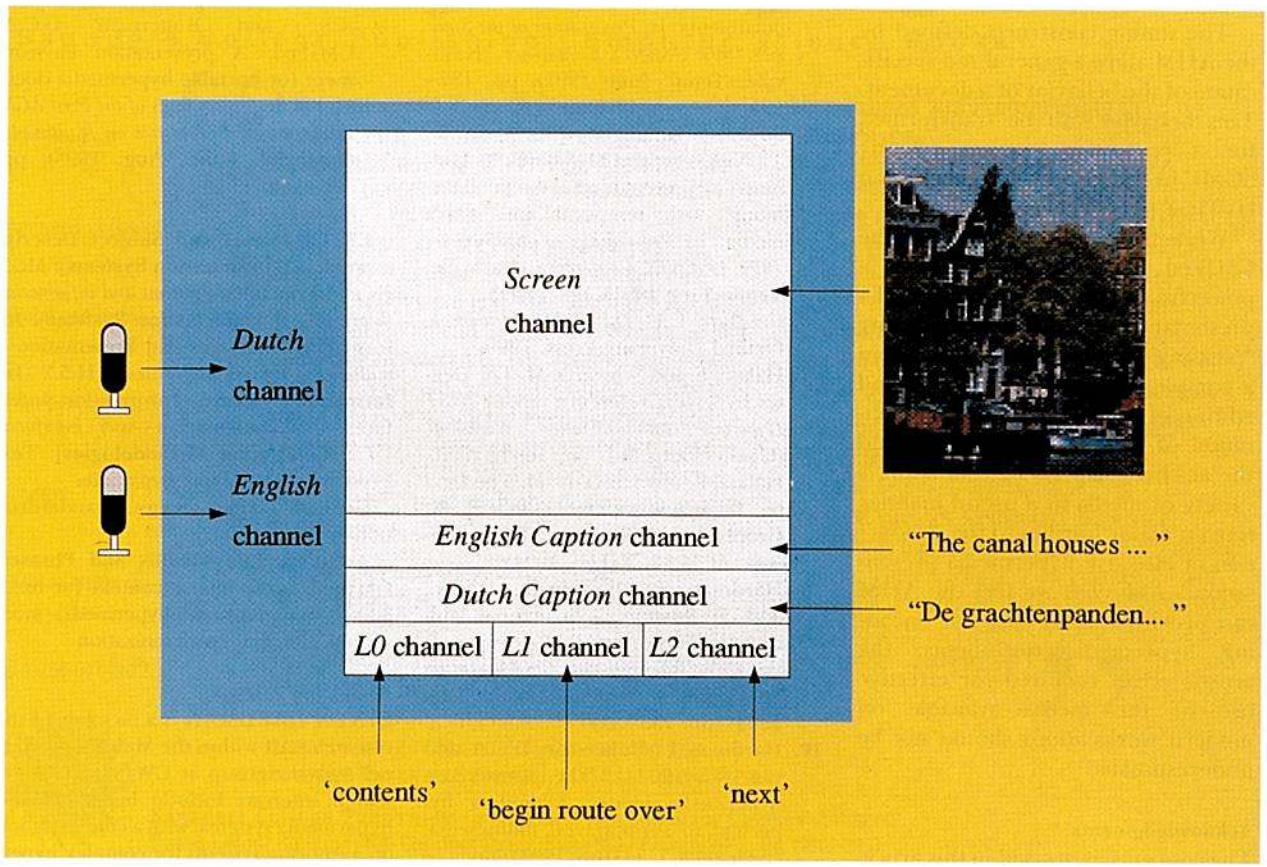


Figure 8. AHM channel architecture

tained by allowing overrides for individual components.

Current Implementation Status

The CMIFed authoring and runtime environment described in the sidebar "CMIFed—An AHM-Based Authoring Environment" is used to experiment with an implementation based on a modified version of the AHM. The most important difference in composition between the AHM and the model supported in CMIFed is that the AHM parallel component is split into two types of composition constructs—parallel and sequential. The purpose of supporting an extra type of composition in the authoring system is to allow the convenient definition of how a group of components are to be displayed during the presentation (that is, the parallel and sequential hierarchical structures are used to derive first-order timing information for the presentation). The derivation of this timing information removes the requirement for explicitly specifying timing information for each child of the composite.

Support for links and anchors in CMIFed is less general than the AHM. The source of a link in CMIFed is an anchor within an atomic component; the destination can be an anchor, or an atomic or composite component (i.e., we do not support the source of a link being a composite component). This has implications for assigning the contexts for the ends of a link. We currently derive the source and destination context from the structure around the ends of the link—the context is taken to be the child of the nearest ancestor choice component containing the end of the link (for further detail on the support for contexts see [11]).

Finally, anchors within CMIFed are of different types: normal, pausing and destination. Normal anchors define the areas which are made active for the reader. Pausing anchors are also active but require that the reader take some action (typically 'clicking') before the presentation can proceed beyond the point at which the anchor was displayed. Destination anchors are used in composite components and refer only to the complete component, removing the requirement for an explicit value. CMIFed does not yet support composite anchors.

Conclusion

We feel that the AHM presents a flexible framework for studying hypermedia implementations. The goal of the model—the union of the basic concepts of hypertext and multimedia—has been met, at least within the limits of existing technology. Through the use of the CMIFed editor, we have been able to create a collection of hypermedia presentations that approximate the richness of the full model, and we continue to investigate new ways of providing better implementation support for hypermedia. The novelty of the CMIFed-supported approach to hypermedia is that the underlying structure can be freely manipulated, unlike many current multimedia authoring systems. This allows for quick and easy creation, copying and altering of presentations [9].

The timing constructs defined by the AHM allow a generalized specification of the behavior of a document. This behavior can be transformed for a particular instance to any SGML-like form [6], including HyTime [12, 13].

While the AHM, together with the CMIFed environment, provide a powerful way to create hypermedia presentations, it is clear that making "pleasing" presentations will remain a complex task. The gathering and editing of source materials, the definition of links in a document, and the aesthetic aspects of combining a variety of media in a useful manner remain major hurdles to the production of effective hypermedia presentations. Although we feel the AHM can provide a solid basis for encoding hypermedia documents, the artistic effort required for effective use of the media available on modern workstations should not be underestimated.

Acknowledgments

The concepts expressed in this article were developed while working with CMIFed, implemented by Guido Van Rossum, Jack Jansen and Sjoerd Mullender. The work was carried out as part of the MAGUS project, funded by the Dutch Ministry of Economic Affairs. We wish to thank the issue editors for their helpful comments on earlier versions of this article. ◻

References

1. Brown, P.J. UNIX Guide: Lessons from ten years' development. In *Proceedings of the Fourth ACM Conference on Hypertext*. D. Lucarella, J. Nanard, M. Nanard and P. Paolini, Eds. *ECHT '92* (Milano, Italy, Nov. 30–Dec. 4 1992), pp. 63–70.
2. Buchanan, M.C. and Zellweger, P.T. Specifying temporal behavior in hypermedia documents. In *Proceedings of the Fourth ACM Conference on Hypertext*. D. Lucarella, J. Nanard, M. Nanard and P. Paolini, Eds. *ECHT '92* (Milano, Italy, Nov. 30–Dec. 4 1992), pp. 262–271.
3. Bulterman, D.C.A. Specifying and support of adaptable networked multimedia, 1, 2 (1993) *Springer-Verlag/ ACM Multimedia Systems*, pp. 68–76.
4. Bulterman, D.C.A., van Rossum, G., and van Liere, R. A structure for transportable, dynamic multimedia documents. In *Proceedings of the Summer 1991 USENIX Conference*, (Nashville, Tenn., June 1991), pp. 137–155.
5. Fujikawa, K., Shimojo, S., Matsuura, T., Nishio S., and Miyahara, H. Multimedia presentation system 'Harmony' with temporal and active media. In *Proceedings of the Summer 1991 USENIX Conference* (Nashville, Tenn., June 1991), pp. 75–93.
6. Goldfarb C.F. *The SGML Handbook*. Oxford University Press, 1990.
7. Halasz, F. and Schwartz, M. The Dexter hypertext reference model. *NIST Hypertext Standardization Workshop*, (Gaithersburg, Md., Jan. 16–18 1990).
8. Halasz, F. and Schwartz, M. The Dexter hypertext reference model. K. Gronbæk and R. Trigg, Eds., *Commun. ACM 37*, 2 (Feb. 1994).
9. Hardman, L., Bulterman, D.C.A., and van Rossum, G. Structured Multimedia Authoring. In *Proceedings of the First International Conference on Multimedia* (Anaheim, Calif., Aug. 1993), pp. 283–289.
10. Hardman, L., Bulterman, D.C.A. and van Rossum, G. The Amsterdam hypermedia model: Extending hypertext to support *real* multimedia. *Hypermedia 5*, 1, (July 1993), pp. 47–69.
11. Hardman, L., Bulterman, D.C.A., and van Rossum, G., Links in hypermedia: The requirement for context. In *Proceedings of Hypertext 93* (Seattle, Wa. Nov. 93).
12. International Standards Organization, Hypermedia/Time-based structuring language, *ISO 10744*, 1992.
13. Newcomb, S.R., Kipp, N.A., and Newcomb, V.T. 'HyTime' the hypermedia/time-based document structuring language. *Commun. ACM. 34*, 11 (Nov. 1991), pp. 67–83.
14. Ogawa, R., Harada, H., and Kaneko, A. Scenario-based hypermedia: A model and a system. In *Hypertext: Concepts, Systems and Applications. Proceedings of the European Conference on Hypertext (ECHT '90)*. A. Rizk, N. Streitz, and J. André, Eds. Nov. 1990, INRIA France, pp. 38–51.
15. Palaniappan M., Yankelovich N., and Sawtelle M. Linking active anchors: A stage in the evolution of hypermedia. *Hypermedia 2*, 1 (1990), pp. 47–66.
16. Ripley, G.D. Digital video interactive—A digital multimedia technology. *Commun. ACM 32*, 7 (July 1989), 154–159.
17. van Rossum, G. and de Boer, J. Interactively testing remote servers using the Python programming language. *CWIQ 4*, 4 (Dec. 1991), pp. 283–303.
18. van Rossum G., Jansen J., Mullender K.S., and Bulterman D.C.A. CMIFed: A presentation environment for portable hypermedia documents. In *Proceedings of the First ACM International Conference on Multimedia* (Anaheim, Calif., Aug. 1993), pp. 183–188.

CR Categories and Subject Descriptors: H.1.1 [**Information Systems**]: Models and Principles—*systems and information theory*; H.5.1 [**Information Systems**]: Information Interfaces and Presentation—*multimedia information systems*; H.5.2 [**Information Systems**]: Information Interfaces and Presentation—*user interfaces*; I.7.2 [**Computing Methodologies**]: Text Processing—*document preparation*

General Terms: Design, Standardization

Additional Key Words and Phrases: CMIFed, composition, context for links, editing environment, hypermedia models, multimedia, synchronization

About the Authors:

LYNDA HARDMAN is a member of the research staff within the Multimedia Kernel Systems group at CWI. Current research interests include hypertext and hypermedia systems, with a concentration on authoring systems for complex hypermedia presentations. email: lynda@cwi.nl

DICK C.A. BULTERMAN is head of the department of Computer Systems and Telematics at the Dutch National Center for Mathematics and Computer Science Research (CWI), and project leader of the Multimedia Kernel Systems project, which investigates fundamental models for supporting distributed multimedia systems. Current research interests include user-level and operating systems support for transportable and adaptable multimedia specifications. email: dcab@cwi.nl

GUIDO VAN ROSSUM is a member of the research staff within the Multimedia Kernel Systems group at CWI. Current research interests include the development of models and architectures that implement multimedia synchronization relationships. email: guido@cwi.nl

Authors' Present Address: Centrum voor Wiskunde en Informatica (CWI), PO Box 94079, 1909 GB Amsterdam, The Netherlands.