SRC Technical Note

1997-3

May 6, 1997

Hypermedia Presentation and Authoring System

Jin Yu and Yuanyuan Xiang

Systems Research Center
130 Lytton Avenue
Palo Alto, CA 94301
http://www.research.digital.com/SRC/

1. Introduction

The tremendous growth in the number of Internet applications is driven by the use of rich media such as images, audio, and videos for representing and exchanging information. The result is the merging of hypertext and multimedia, or hypermedia. Hypermedia documents (eg. HTML files) function as containers for hypermedia objects, which are network transparent entities with a variety of media types, such as MPEG, GIF, RTF, etc. The composition and presentation of hypermedia documents presents us with many new challenges, because of the dynamic nature of multimedia and the large variety of media types. Specifically, a hypermedia system must meet the following requirements [1]:

1. high-level temporal synchronization between hypermedia objects
2. the provision of time-dependent spatial synchronization within the hypermedia document architecture
3. the integration and transparent handling of various media types
4. network transparency in document and object retrieval
5. a simple but powerful user interface manipulating the temporal and spatial information within the hypermedia document

Existing HTML authoring and browsing tools provide static spatial management by using relative geometry based on the underlying text. Emerging technologies such as Java Applets, JavaBeans, and Netscape Plug-ins extend the basic capabilities of HTML tools. However, the combination of the technologies and the HTML tools is still not sufficient to represent temporal-based multimedia presentations with associated dynamic spatial layout, because they are constrained by the static nature of HTML. Although many extensions have been made, HTML still provides primarily static layout controls.

Our approach eliminates the traditional concept of page, and provides temporal synchronization of distributed media objects within a hypermedia document. With this approach, our system allows the spatial layout of the document to be changed dynamically with respect to time. In addition, the integration of the browsing and authoring environment gives users a powerful yet simple way to graphically construct and manage hypermedia objects and their containing documents.

The HPAS environment supports the structure-based composition and the dynamic presentation of the HPA (Hypermedia Presentation/Authoring) document architecture, which conforms to the SGML standard. An HPA document is composed of a list of HPA objects, which are identified by Uniform Resource Locators [4]. Each object can have an associated media stream (but this is not required) with an appropriate MIME type. In addition, the objects within the document are temporally related and every object carries its own spatial layout information.

The next section outlines the architecture of the system. Sections 3 and 4 discuss the temporal and spatial management. Section 5 describes HPA objects in detail. Finally, we briefly discuss the implementation issues and future work in sections 6 and 7.

2. Architecture and System Overview

The global picture of HPAS is shown in Fig.1. The presentation subsystem comprises the components on the left hand side, and the authoring subsystem comprises the components on the right hand side. The temporal and spatial managers are shared by both subsystems. The network manager provides both uploading and downloading services to the rest of the system. Our model permits multiple instances of those components. For example, several browser windows, or presentation managers may be active at the same time.

The presentation subsystem is the browser part of HPAS. Coordinating with the temporal and spatial manager, the presentation manager drives the various object presenters to display objects into the browser window. The authoring subsystem allows authors to graphically construct HPA documents. The composition manager communicates with temporal and spatial managers to synchronize the objects within HPA documents. It also manages the object editors, which are responsible for the authoring of object media data.

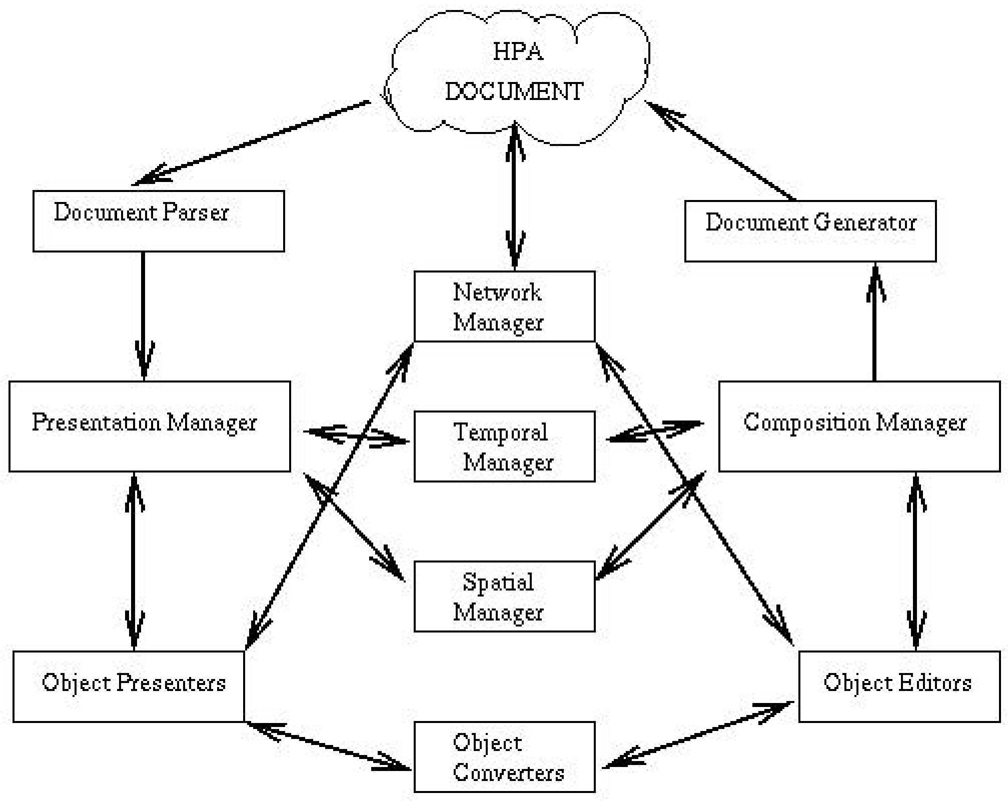


Fig.1 General Architecture of HPAS

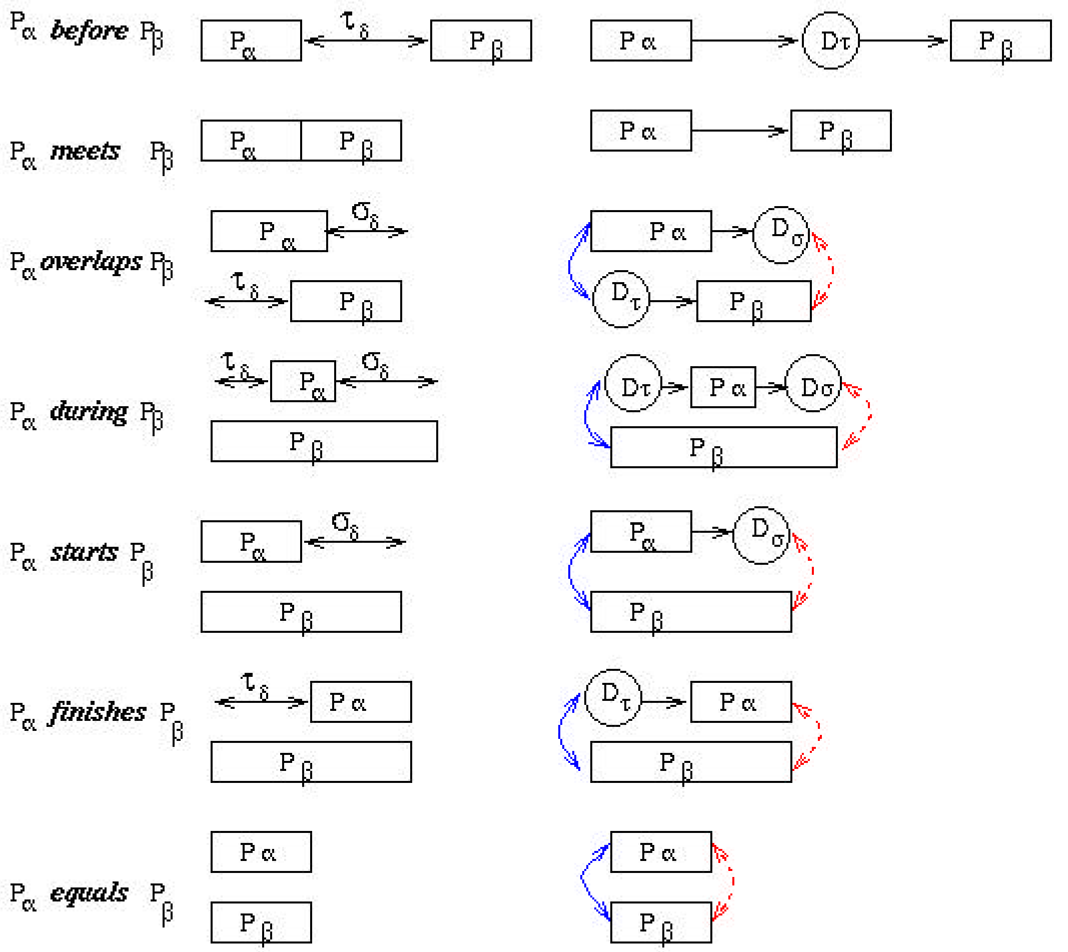
3. Temporal Synchronization

A crucial step in designing multimedia applications is to guarantee that the composition of multiple monomedia is synchronized temporally. There are two distinguished time synchronization levels between objects in a hypermedia document, namely low-level media-specific synchronization and high-level interobject synchronization. The low-level synchronization requires media-specific timing constraints to be satisfied. For example, speech (audio) should match the motion of lips (video). On the other hand, the interobject synchronization specifies only the temporal constraints of the construction, lifetime, and destruction of objects. Currently, we are concerned only with interobject synchronization, because low-level synchronization requires the knowledge of specific media types.

Traditionally, there have been several approaches to achieving temporal synchronization between objects. Scripting-based systems allow authors to explicitly program timing information. During presentation, the systems interpret the scripts and perform actions specified in the scripts. This approach is powerful, but usually requires proficiency in programming, which severely limits the range of authors. Timeline-based systems model a conceptual timeline. Authors simply place objects to be presented on this timeline. However, since this approach requires objects to be placed relative to the time axis, it is not well suited for operations such as moving, copying, or deleting parts of the presentation. Furthermore, neither of the approaches allows direct and intuitive manipulations of the temporal structure of the hypermedia document. In our approach, temporal information is described in terms of object relations; that is, each object is described in terms of other related objects. This is well suited for presentations with distributed media objects, since the elapsed time of each object depends on both the network bandwidth and CPU speed. In addition, HPAS also provides users with a powerful interface to access and manipulate the temporal information in HPA documents. The following sections describe the temporal synchronization mechanism in detail.

3.1 Temporal Interval and MRG

In our framework, each object is associated with a temporal interval, which is characterized as a nonzero duration of time in seconds. Given any two temporal intervals, there are thirteen mutually exclusive relationships [2]. According to Little and Ghafoor [7], the temporal relations can be represented as Fig.2a. The figure shows only seven of the thirteen relations since the remaining ones are inverse relations. For instance, *after* is the inverse relation of *before*.



P_α before P_β
P_α meets P_β
P_α overlaps P_β
P_α during P_β
P_α starts P_β
P_α finishes P_β
P_α equals P_β
P_α
P_β
τ_δ
σ_δ
D_τ
D_σ

Fig.2 **a) Temporal relations** (after [7]) **b) Corresponding MRG**

To represent our temporal model, we define a set of *links* between two objects, p_1 and p_2:

- (p_1 *SerialLink* p_2): the temporal interval $I(p_1)$ *meets* $I(p_2)$. In other words, p_1 is the parent of p_2, and p_2 is the child of p_1.
- (p_1 *StartSync* p_2): the temporal interval $I(p_1)$ and $I(p_2)$ share the same starting point.
- (p_1 *EndSync* p_2): the temporal interval $I(p_1)$ and $I(p_2)$ share the same ending point.

Besides *links*, each object has a user-defined attribute `time to live', or *ttl*, which specifies how long the object may be active on the screen. In addition, our model also defines the notion of *dummy object*. A dummy object does not have any content, nor does it have an associated type. Such an object can be used to represent a time delay by defining its *ttl* attribute.

With the definitions of *SerialLink*, *StartSync*, *EndSync*, *ttl*, and dummy object, we can now introduce the notion of Media Relation Graph (MRG), as shown in Fig.2b. The MRG in Fig.2b corresponds to the temporal relations illustrated in [7] (Fig.2a). In our MRG, a one-way arrow denotes the *SerialLink* operator, where the left hand side operand is the object at the starting end of the arrow and the right hand side operand is the object being pointed to by the arrow. Similarly, the *StartSync* operator is denoted by a two-way solid arrow, and the *EndSync* operator is represented by a two-way dashed arrow. Finally, a rectangular node represents a regular object and a round node represents a dummy object. By using the notions of *SerialLink*, *StartSync*, *EndSync*, *ttl*, and dummy object, we can use MRG to represent all the thirteen temporal relationships defined by Allen [2].

3.2 Object Activation and Deactivation

This section describes the object activation and deactivation policies. Before we state the policies, several concepts need to be defined:

- The activation time of an object, or *atime*, is the time when the object **appears on the screen**. At that time, the object is said to be *activated*.
- The deactivation time of an object, or *dtime*, is the time when the object **disappears from the screen**. At that time, the object is said to be *deactivated*.
- The elapsed time of an object, or *etime*, is the difference between *dtime* and *atime*; *etime* is said to be the temporal interval of the object.
- The content time, or *ctime* of an object, is the time needed to present the object's entire media stream.

Note that if an object's *ttl* value is greater than its *ctime*, it will stay idly on the screen after *ctime* is reached and before *ttl* is expired.

We also define the starting point of an HPA presentation to be the root object, which is a dummy object with the value of its *ttl* attribute being zero. The root object is deactivated once the presentation starts. In addition, for every object in the presentation (excluding the root object itself), an implicit *SerialLink* exists between the root and the object.

Now we can introduce the activation and deactivation policies:

- An object **may** be activated if both of the following hold:
    1. The object's parents and the parents' *EndSync* peers are all deactivated.
    2. All of the object's *StartSync* peers satisfy the condition above.
- Object deactivation is governed by the following rules:
    1. The object must be deactivated with all of its *EndSync* peers at the same time.
    2. For the object and all of its *EndSync* peers, *etime* is greater than or equal to *ttl*.
    3. The priority of *ttl* is higher than that of *ctime*.
    4. If *ttl* is not specified, the value of *ctime* is assigned to it.

Even at the time that both activation policies are satisfied, the system might not be able to activate the object immediately, since at that time the object's media stream may not be readily available from the network. This situation will be explained further in section 6.

3.3 MRG Example

Let's consider the following sample code from an HPA file:

```
&ltobj id = 0 name = "Root" ...
     serialLink = '1 2 3' timeToLive = 0> </obj>
&ltobj id = 1 name = "Bird Show" type = "video/mpeg" ...
     startSync = '2 3' endSync = '2' timeToLive = 10> </obj>
&ltobj id = 2 name = "Bird Walk" type = "video/mpeg" ...
     serialLink = '5'  startSync = '1' endSync = '1' timeToLive = 6> </obj>
&ltobj id = 3 name = "Dummy" ...
     serialLink = '4' startSync = '1' timeToLive = 4> </obj>
&ltobj id = 4 name = "Bird Intro" type = "text/html" ...
     endSync = '6' timeToLive = 20> ... </obj>
&ltobj id = 5 name = "Seagull" type = "image/gif" ...
     seriallink = '6' timeToLive = 5> </obj>
&ltobj id = 6 name = "Bird Song" type = "audio/basic" ...
     endSync = '4' timeToLive = 5> </obj>
```

In the code segment above, each *obj* element describes an HPA object. The *serialLink* attribute of an object specifies the object's children, which are represented by the list of object IDs. The syntaxes of *startSync* and *endSync* are identical to that of *serialLink*. Object IDs are a sequence of non-negative integers starting from zero, which represents the root object. The composition manager is responsible for assigning object IDs.

Table 1 shows another representation of the objects described in the code fragment, and Table 2 illustrates the temporal relations of the objects, defined in terms of the terminologies used by Allen [2].

Table 1. Object descriptions

Obj Name	Type	Obj Id	Time To Live
Root		0	0
Bird Show	video/mpeg	1	10
Bird Walk	video/mpeg	2	6
Dummy		3	4
Bird Intro	text/html	4	20
Seagull	image/gif	5	5
Bird Song	audio/basic	6	5

Table 2. Temporal relations

before	
meets	(2,5), (5,6)
overlaps	(1,4)
during	
starts	(3,1)
finishes	(6,4)
equals	(1,2)

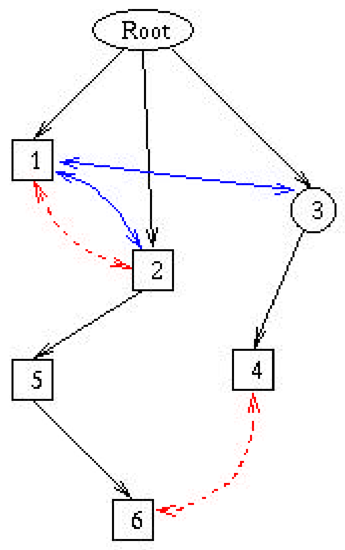
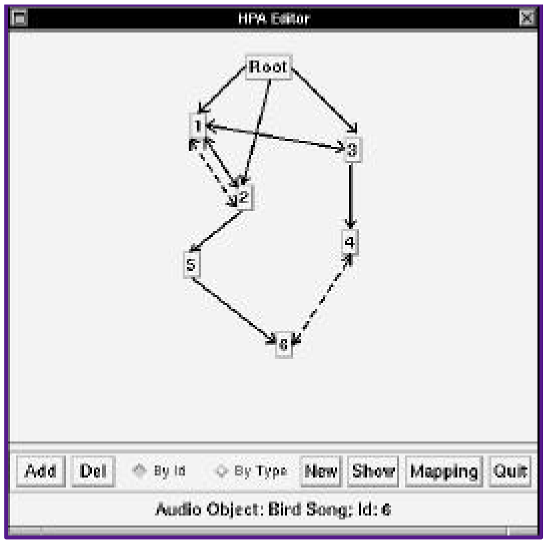


Fig.3 Example MRG

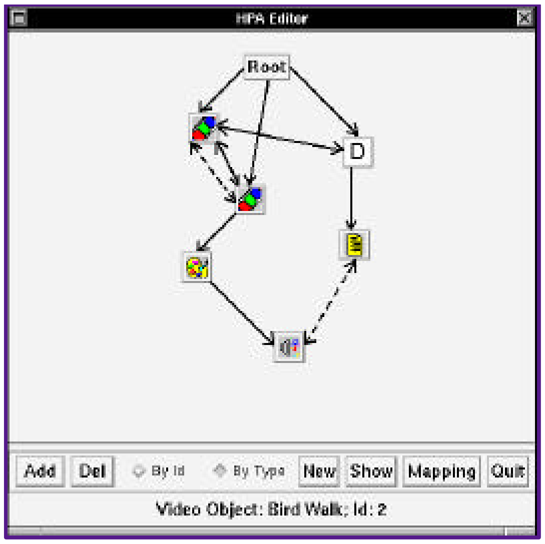
Finally, Fig.3 describes the MRG representation which corresponds to the temporal relations in Table. 2.

3.4 Using Our Temporal Framework

For supporting authors with editing the temporal structure of HPA documents, the composition manager utilizes the services of the temporal manager and provides a tree-like temporal editor, whose content exactly reflects the temporal graph structure (MRG). The two images below are screen shots of the temporal editor. The image on the left shows objects by their IDs, and the one on the right shows objects by their types, with color pixmaps corresponding to major MIME types (text, image, video, audio, etc.). Notice the similarity between the screen dumps and Fig.3.



Audio Object: Bird Song; Id: 6



Video Object: Bird Walk; Id: 2

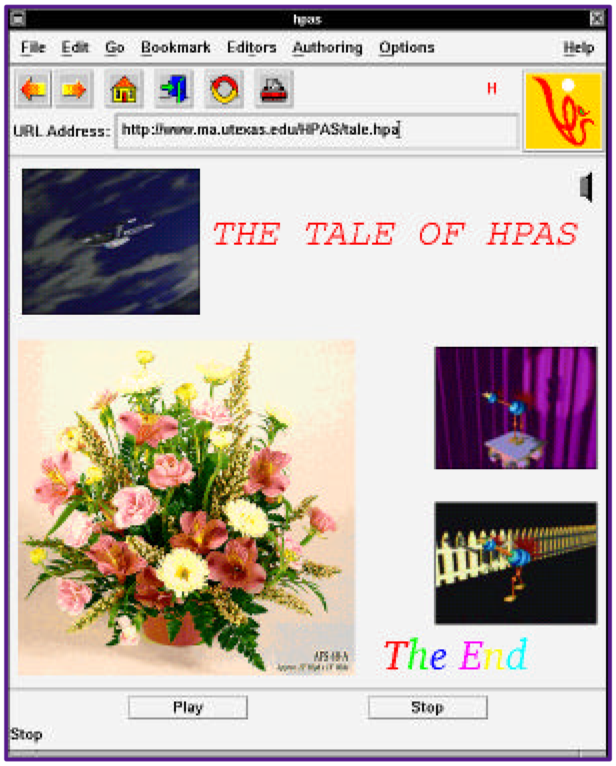
The temporal editor reflects only the relationships between objects, not the content of each object, that is, the content need not exist in the temporal authoring process. This property enables authors to choose either a *top down* or a *bottom up* approach to composition [6]. In the *top down* approach, authors first build the temporal structure by creating nodes which represent objects in the MRG. Authors then create the synchronization links between objects by drawing lines between the nodes in the graph. The final step is to provide the content of objects by using various media-specific object editors. By contrast, the *bottom up* approach allows authors to create the content of objects first, and then create the temporal relations between them.

The temporal authoring process starts with the creation of nodes in the MRG (the nodes correspond to the buttons in the screen shots). Those nodes represent the objects in an HPA document. Next, one-way arrows are drawn between nodes, reflecting the *SerialLink* relationships between corresponding objects. Similarly two-way solid arrows and two-way dashed arrows are drawn to denote *StartSync* and *EndSync* relations, respectively. The system also allows authors to create the three types of links in different orders; authors may first create *StartSync*, then *SerialLink*. While adding every link between objects, the composition manager verifies that the resulting graph is free of temporal conflicts. The attributes of an individual object may be specified by right clicking on the corresponding button and selecting the appropriate menu option in the temporal editor. This is the crucial step where an author specifies the *ttl* value of the object. Finally, the document generator is responsible for generating the text representation of the document structure, which has been passed to it by the composition manager. As a side note, authors are discouraged from editing HPA documents manually (using a text editor) because of the inherent complexity of temporal logic. However, it is perfectly legal to change simple object attributes such as *src* (described in section 5) by using a text editor, since it is convenient to do this kind of modification offline (without invoking HPAS).

In the presentation process, the presentation manager receives the parsed document structure from the document parser, then coordinates with the temporal manager to eliminate temporal inconsistencies. This is necessary because the manual editing of HPA documents is allowed and errors could therefore be introduced. The resulting documents may contain temporal relations not verified by the composition manager.

Its close integration with the temporal manager also allows the presentation manager to provide facilities that supports start, pause, resume, and step operations. If the temporal validation of the document structure is successful (or at least part of the document is validated), the presentation will be started and objects will be activated according to the temporal information in the document. The *pause* operation temporarily stops the presentation and all the active objects, while the *resume* operation continues the presentation and all the objects stopped by *pause*. Finally, performing the *step* operation immediately terminates (*deactivates*) all the active objects on the screen, and their children are started if they satisfy the activation policy.

A screen shot of HPAS' presentation window (browser) is shown in the next picture. It shows three video objects (the starship `Enterprise' and the two birds), an image object (the flower), two rich text objects, and an audio object (at the upper-right corner).



4. Spatial Synchronization

Many approaches exist for managing spatial synchronization in hypermedia documents. The simplest approach is using absolute layout specification, where the geometry of an object is defined by the coordinates(x, y, width, height). HTML takes a different approach. Based on the underlying text, it relies on control elements (list, table, center, etc.) to manage the document geometry. However, these approaches are not sufficient for supporting temporal-based multimedia presentations. In particular, HTML's layout elements (such as table) are static and do not change over time.

Our approach allows the dynamic placement of objects by providing a time-dependant layout scheme, which uses the notions of *cell* and *area*. While authoring, the composition manager evenly divides the document window into a grid of cells, with the number of horizontal and vertical cells specified by the author (so the size of each *cell* is fixed within a presentation). A rectangular group of cells forms an *area*, which may be occupied by one object. The object is constrained by the *area*. The *area* does not resize itself to accommodate the object; instead, the object is responsible for resizing itself to fit into the *area*. The layout and the number of *area*s on the grid change with respect to time. At any point in time, there may be zero or more *area*s on the grid, corresponding to zero or more regular objects. Dummy objects do not have *areas*.

For each object and its associated *area*, several geometric attributes may be defined. Here is an HPA file fragment which describes the geometry of an object:

```
&ltobj id = 1 ... area = '0 0 5 4' geomUnit = grid
     topOffset = 1 align = hcenter ...> ... </obj>
```

The *area* of the above object starts at the cell in the upper-left corner of the document window and spans 5 cells horizontally and 4 cells vertically. The top border of the object is 1 cell from its *area*'s top border, and the object itself is centered horizontally within its *area*.

The scheme above specifies only how an object may be placed within its *area*. Since several objects may overlap one another at any time, we also need a mechanism for guaranteeing that associating an object with its area does not produce the undesired overlapping effect. So obviously some form of validation is needed when an author uses the spatial manager to define an object's *area*:

An object *o* can occupy an *area a* if either or both of the following hold:

- *a* has no intersection with the *area*s of any other objects.
- For every object *c* where *c*'s *area* intersects with *o*'s *area*, *c* belongs to the union of *P* and *Q*, where *P* is the set of objects formed by *o*'s ancestors and the ancestors' *EndSync* peers, and *Q* is the set of objects formed by *o*'s descendants and the descendants' *StartSync* peers.

It is obvious that the objects in *P* are deactivated before or at the activation time of *o*, and the objects in *Q* are activated at or after the deactivation time of *o*. Therefore, it is not possible for *o* to overlap the objects in *P* or *Q*, since they live in different time frames.

With the criteria above, a given object can occupy only a limited set of cells. This is too restrictive in some situations. Therefore, we further extend our document architecture to incorporate the concept of *scene*. In the spatial aspect, each scene defines its own grid layout, therefore objects in one scene are free from the spatial constraints associated with objects in another scene. In the temporal aspects, each scene has its own temporal graph, and there are no temporal relations between objects in two adjacent scenes, except that the objects in the later scene are activated at or after the deactivation times of the objects in the previous scene.

In brief, *scene* provides logical groupings of objects within HPA documents, and the objects in each scene form a complete presentation. Moreover, the breaking of a complicated HPA document into multiple scenes is analogous to the dividing of a book into chapters. It provides a better organization of the document and allows authors to edit subsections (scenes) of the document in any order. Furthermore, viewers may step through scenes or randomly access particular scenes at will.

5. Framework and Objects

In contrast to traditional large monolithic applications, the HPAS environment provides a framework for embedding components. Similar component programming models exist, such as CI Labs' OpenDoc and Microsoft OLE. However, these models are heavy-weight and are not available on many Unix platforms. Netscape's Plug-ins framework is light-weight and portable on most platforms, but it does not provide native support for media conversion and editing, which are essential for the extensibility of the applications on top of the framework. Our framework provides a simple but powerful application programming interface (API). The API is specific to our environment and hence it is small and efficient. The system is not aware of any specific media types. This design facilitates the transparent handling of different media streams. The components of our framework include object presenters, object editors, and object converters. The framework itself does not implement any components; it only provides a set of basic services to the components. The main ones are network management, temporal and spatial synchronization service, which we have discussed in the previous sections. We will describe network service in the implementation section.

The basic element of our environment, the HPA object (or *hobject* for short), is a network transparent entity uniquely identified by a URL [4]. Each hobject has an associated type, which conforms to the MIME [5] standard. There are two major categories of hobjects: ones with media streams and ones without media streams. Examples of stream-based hobjects are JPEG images and MPEG videos. Streamless hobjects are typically application configurations, such as a game setting for a multi-user tank game. Hoject attributes are specified in both the temporal and spatial authoring processes, but there are several attributes that are not related to temporal or spatial management. For example:

```
&ltobj id = 2 type = "video/mpeg" name = "Love Bird"
     src = "http://www.some.com/mpeg/bird.mpg"
     anchor = "http://www.goldfish.com/gold/" ...> ... </obj>
```

The *type* attribute specifies the MIME type of the hobject, the *src* attribute denotes the location of the content of the hobject, and the *anchor* attribute defines the anchor represented by the hobject. As a side note, an hobject with no type and no content is a dummy object.

5.1 Media Handler

Fundamental to our design is the close interaction between hobjects and media handlers (or *mhandler*s for short), which are dynamically loaded modules that implement all of HPAS' media-specific capabilities. Hobjects are passive entities; they contain only attributes and data streams. Mhandlers are active entities; they provide operations such as displaying and editing hobjects. The system itself does not implement any mhandlers. All handler modules are loaded into the system at run time. Mhandlers use the basic network service provided by HPAS, and they are also free to implement their own network services. A single mhandler may choose to support more than one MIME type, and it may implement presenting features, editing features, or both. Finally, any hobject attributes that are not understood by HPAS are passed to mhandlers for further processing. This allows authors to pass media-specific information to the mhandlers.

5.2 Presenter, Editor, and Converter

The components of the HPAS framework are object presenters, object editors, and object converters. Among them, presenters are implemented solely by mhandlers; editors are either existing standalone applications or modules implemented by mhandlers. Converters are simply external filters. The framework provides a plug-and-play configuration interface to these components, so one immediate advantage is that adding or removing components requires no reconstruction of the system. Furthermore, this design greatly increases the system's extensibility, since the types of media objects that can be handled by HPAS are virtually unlimited.

Presenters are responsible for the inline displaying of objects, and editors are responsible for authoring the content of objects. To implement a presenter, apart from being able to display a particular type of object, an mhandler must provide certain operations, such as *pause* and *resume.* These two operations are called when the presentation manager pauses and resumes an HPA presentation. To implement an editor, apart from providing editing functions, an mhandler must be able to inform HPAS of its content modification state. Common operations such as construction/destruction, stream downloading/uploading, and printing are required for both presenters and editors. If an mhandler implements both presenter and editor, it should provide different interfaces for the two. In particular, the editing part should have a popup window and a menubar.

Providing an HTML presenter allows HPAS to be used as a normal HTML browser, which is necessary because the current Web infrastructure is based on HTML files. Also, it is possible to implement a Java Applet presenter. In this case the Applet mhandler will implement a singleton Java interpreter, which is to be shared by multiple instances of Applet presenters.

Object editors are usually well developed media-specific applications. This allows HPAS to exploit the powerful features of existing applications. However, the drawback of this approach is the loose integration between HPAS and the editors. For example, when HPAS wants to close an editor, it has no way of knowing whether the content of the editor has been saved or not.

When no presenters or editors exist to handle a particular object stream, an object converter will be invoked to operate on the stream, creating a stream with a different MIME type. The resulting stream is then passed over to an appropriate presenter or editor for further processing. Converters are external programs written for generic media conversion purposes.

5.3 User Interaction

System level user interactions are provided by the presentation manager through *anchor*s. An anchored object is just like any other object, except clicking on it will typically overlay the browser window with the content of the entity pointed to by the anchor. The anchor is addressed by a URL [4], and may contain a fragment ID in the form *#fragment_id*, which points to a subpart of the target entity. If the target entity is an HPA document, the anchor may refer to a particular scene or even to an hobject within the scene. This allows a viewer to start viewing the presentation from that particular scene or hobject.

Object level user interactions are solely implemented by presenters. Presenter writers may provide viewers with any type of user interaction, as long as these interactions are permitted by the window system.

6. Implementation Issues

The framework implements a *temporal scheduler* to manage the activation and deactivation of hobjects. We considered three approaches to accomplish this. In the first approach, upon the activation of each hobject, we record the system time as the *atime* of the hobject. A polling loop periodically checks system time and compares it with the *atime* of the hobject to decide if the hobject's *ttl* value is expired; among other things, the polling loop also examines the three *link* attributes of the hobject, and activates and deactivates various hobjects as appropriate. The second approach is completely event-driven. Since most user interface toolkits provide some form of event loop, we rely on the event loop to provide scheduling services, in the form of timer events. The third approach is to implement each active hobject as a thread, and synchronize the threads accordingly. Using a polling loop as in the first approach is a waste of CPU time, since much computation is done within the loop, and most rounds of the loop are wasted (no hobjects are activated or deactivated). In addition, thread interfaces are different on many operating systems, and not all systems support re-entrant system calls. Therefore, we decided to use the event-driven model to implement our temporal scheduler.

Besides the temporal scheduler, we also multiplex the network manager into the user interface event loop. This approach allows the system to provide timer, network, and interface events in a single event loop. When timer/network events are dispatched, the corresponding timer/network callbacks are called. An hobject may be activated in either a timer callback or a network callback, whichever is later. (There are several types of timer and network events; we are talking about the ones that may activate hobjects.) On the other hand, the deactivation of hobjects can be triggered by timer events, media access events (such as end of media), and exception events (such as network and media error).

The network manager is designed to provide generic network services. It listens to the network all the time. Depending on the preferences of individual mhandlers, it can deliver data in a progressive fashion or write all the data into files and then pass the filenames to the mhandlers. The most commonly used services from the network manager are the HTTP `GET' and `POST' methods [3]. In addition, mhandlers may implement their own network services, such as RTP [8] for video related mhandlers.

Mhandlers are implemented as shared libraries, which can be loaded into HPAS at runtime. Since HPAS is single-threaded, media handlers should avoid using blocking system calls. When implementing a presenter, an mhandler is responsible for saving its state while paused by the presentation manager. In particular, the mhandler must save the excessive stream delivered to it while the presentation is paused. The reason is that in order to improve performance, the network manager keeps sending bytes to the mhandler regardless of the state of the presentation.

The system is written in C++, with the mhandler API in C, since we allow mhandlers to be written in C or C++. The user interface is built with OSF/Motif. We have implemented an MPEG mhandler based on the MPEG2 decoder from the MPEG Software Simulation Group, an image mhandler, a rich text mhandler and an audio mhandler based on the EuroBridge Widget Set. Finally, an HTML mhandler is implemented by using NCSA Mosaic's HTML widget.

7. Conclusions and Future Work

In the past one and half years we have been working on the HPAS project to support the creation and presentation of hypermedia documents. The system provides services for integrating pluggable components such as presenters, editors, and converters. The basic elements of the environment, hobjects, are synchronized both temporally and spatially during the authoring and presenting processes of HPA documents. The system is particularly suited for presenting dynamic (but not text intensive) information on the Web, such as product/service advertising, guided course work, etc.

In the future, we will enhance the mhandler API. In particular, we will try to provide some communication mechanism between different mhandlers, which makes low level media-specific synchronization a possibility in our framework. A simple example is *send*, which sends a message from one mhandler to another. Using this primitive, the source mhandler may transfer data to the target mhandler, or it may request certain operations to be performed on the target media handler. The *send* primitive should be flexible enough so that both synchronous and asynchronous invocations are allowed. Finally, we will provide support for a wider range of applications by developing more mhandlers.

8. Acknowledgment

We are very grateful to Harrick Vin and Donald Fussell. Without their support and encouragement, the project would not exist. We would also like to thank Zaijin Guan for his participation in the early stage of the project, and Wei Wei and Biao Zhang for their various suggestions and help throughout the development of the project. Finally, we would like to give special thanks to the reviewers Paul McJones, Marc Brown, Marc Najork, Krishna Bharat, Cynthia Hibbard, Hseuping Chen, Xue Lu, and Huiqun Liu for their timely advice on the paper.

References

[1] Philipp Ackermann.
Direct Manipulation of Temporal Structures in a Multimedia Application Framework.
ACM Multimedia 94 Proceedings, pp. 51-58, October 1994.

[2] James. F. Allen.
Maintaining Knowledge about Temporal Intervals.
Communications of the ACM, vol.26. no.11, pp. 832-843. November 1983.

[3] T. Berners-Lee, R. Fielding, and H. Frystyk.
Hypertext Transfer Protocol -- HTTP/1.0, RFC1945. May 1996.

[4] T. Berners-Lee, L. Masinter, and M. McCahill.
Uniform Resource Locators (URL), RFC1738. December 1994.

[5] N. Borenstein and N. Freed.
MIME (Multipurpose Internet Mail Extensions), RFC1341. June 1992.

[6] Lynda Hardman, Guido van Rossum, and Dick C.A. Bulterman.
Structured Multimedia Authoring.
ACM Multimedia 93 Proceedings, pp. 283-290, August 1993.

[7] Thomas D.C. Little, and Arif Ghafoor.
Synchronization and Storage Models for Multimedia Objects.
IEEE Journal on Selected Areas in Communications, vol.8, no.3, pp. 413-427, April 1990.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson.
RTP: A Transport Protocol for Real-Time Applications, RFC1889. January 1996.