SRC Technical Note
1997 - 027a
April 1, 1998

A Simple, Intuitive Hypermedia Synchronization Model and its Realization in the Browser/Java Environment

Jin Yu

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

A Simple, Intuitive Hypermedia Synchronization Model and its Realization in the Browser/Java Environment

Jin Yu
DEC Systems Research Center
130 Lytton Ave., Palo Alto, CA 94301
jinyu@pa.dec.com

Abstract

This paper presents a simple and intuitive hypermedia synchronization model – the Media Relation Graph (MRG), and an alternative implementation of the Hypermedia Presentation and Authoring System (HPAS), which is the testbed for MRG. HPAS is a system for presenting, integrating, and managing time-based hypermedia documents on the Web. The underlying temporal model of HPAS combines the power of both interval-based and point-based synchronization mechanisms. The new Java-based implementation exploits many rich features of commercial Web browsers and reuses existing browser components, such as plugins and Java applets. (An overview of HPAS and its original Unix/C implementation can be found elsewhere [18].)

1 Introduction

The exponential growth in the number and variety of Web-oriented products and services is driven by the use of rich media types such as image, audio, and video. The combination and integration of these monomedia, or multimedia, is widely used for representing and exchanging information. Used together with both content-based and time-based navigation, the result is the merging of multimedia and hyperlinks, or hypermedia.

Many important multimedia applications cannot be implemented using today's Web technology such as HTML and HTTP. Scenarios like "show image 1 for 10 seconds; after image 1 has started for 5 seconds, play audio 1 and show image 2 in parallel for 20 seconds" cannot be easily expressed in current Web technology. However, this kind of scenarios is typical in many areas, including:

- TV-like content on the Web
- Integrated streaming audio/video on the Internet
- Interactive/on-demand television/video
- Canned CD-ROM multimedia presentation

Time-based hypermedia documents are well-suited for applications in the above areas.

Within a time-based hypermedia framework, a distinction should be made between hypermedia documents and hypermedia objects (or simply documents and objects). Objects usually represent monomedia data, such as MPEG videos and GIF images, which are identified by Uniform Resource Locators (URLs) [2] and MIME types [4]. Documents function as containers for objects. A document describes the meta information about the enclosed objects. The information includes the temporal, spatial, and content relationships among a number of related objects, and the attributes of individual objects. Examples of documents are HSL (a format implemented by our system) and SMIL [19]. HTML files are also hypermedia documents, except that they are not time-based. Documents can also be treated like objects; e.g. an HTML file embedded into an HSL document is treated as an HTML object.

The presentation of a time-based hypermedia document can be modeled by a directed acyclic graph (DAG), as in Figure 1. Vertices in the graph represent media objects, and the directed edges represent the flow of time. Note that the edges point downward, so the time flows top to bottom in the graph. Intuitively, an edge from vertices v_1 to v_2 means that v_1 to v_2 are played in serial, with v_1 before v_2; ie. they do not overlap in time. At any point of a presentation, there is a (possibly empty) set of media objects playing on the screen, which can be modelled as a set of multiple concurrent threads, with each thread presenting an active object. The number of concurrent threads changes dynamically throughout a presentation. For example, in Figure 1, there are three concurrent threads (and thus three active objects) at some time instant, represented by the three blackened vertices.

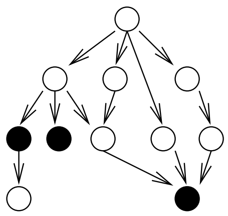

Figure 1: Presentation graph

Our temporal synchronization model, the Media Relation Graph (MRG), is based on some further refinement of the temporal DAG described above. To represent the information in MRG, we have designed the Hypermedia Synchronization Language (HSL, a SGML-conforming format [16]). HSL documents can be interpreted and presented by the Hypermedia Presentation and Authoring System (HPAS), which we have developed as a testbed for MRG and HSL. In this paper, we will describe the Java-based implementation of HPAS, which supports a rich subset of the features in the original Unix/C implementation [18].

The next section describes our temporal model in detail. Sections 3 and 4 discuss the validation and presentation of documents produced by our synchronization model, respectively. Section 5 outlines the format of HSL documents. Hyperlinking is briefly explained in section 6. Finally, section 7 describes the new browser/Java-based implementation.

2 Synchronization model

There are two levels of multimedia synchronization, namely intra-object synchronization and inter-object synchronization [3]. The former is concerned with the time relations within one media object, such as an MPEG video, while the latter is concerned with time relations between two or more media objects. There are yet two more subtypes of synchronization within the inter-object category: low-level "lip" synchronization and high-level endpoints-based synchronization. In the HPAS project, we are mainly interested in high-level endpoints-based synchronization.

There have been numerous approaches in specifying high-level media synchronizations. Most of them bear two characteristics. First, the syntax is declarative. Declarative syntax is easier to author and easier to convert. On the other hand, scripting-based systems require proficiency in programming, which severely limits the range of authors. Scripts are also less portable and reusable. Second, the specifications are relation-based; that is, each object is described in terms of other temporally related objects. Timeline-based (non-relational) systems require the start/end times of objects to be fixed on the time axis; therefore, document parts cannot be efficiently reused (requires readjusting all the start/end times of the objects to be reused); furthermore, timeline-based specifications cannot model nondeterminism (objects with unknown durations). It should also be pointed out that both the scripting and timeline approaches do not scale well.

The relation-based specifications can be further divided into two major flavors: interval-based vs. point-based [17]. In interval-based models, each media object is associated with a temporal interval, which is characterized as a nonzero duration of time. According to Allen, given any two temporal intervals, there are 13 mutually exclusive relationships [1]. The 13 temporal relations can be represented as Figure 2a [13]. The figure shows only seven of the thirteen relations since the remaining ones are inverse relations, by simply swapping the labels. For instance, *after* is the inverse relation of *before*. In point-based approaches, relations are based on time instants. Given two time instants, there are 3 mutually exclusive relationships, namely *before* ($<$), *simultaneous to* ($=$), and *after* ($>$) [17]. Few existing multimedia systems are solely based on point-based specifications; Madeus [9] is purely based on Allen's interval relations; most other systems, such as CMIF [7], ISIS [11], OCPN [13], Firefly [5], and CHIMP [6], are based on a hybrid of the two approaches.

Our temporal synchronization model, MRG, is also based on a hybrid of the interval-based and point-based approaches. Media objects are modeled as temporal intervals, and the start/end times of the objects are treated as time instants. The merit of our approach is that the specification is particularly intuitive; this makes the authoring process much easier. In the following sections, the semantics of MRG will be described in greater detail.

2.1 Endpoints-based relations

Allen's 13 interval relations cover all the possible relationships between two temporal intervals. The 13 interval relations can efficiently describe what happened between two temporal intervals in history (i.e. after play-out of the two objects corresponding to the two intervals); however, Allen's relations are not well-suited for specifying what should happen between two intervals in the future [10]. For example, in the relation p_α *overlaps* p_β, p_β cannot be started during p_α if the end time of p_α is unknown.

Unlike Allen's purely interval-based model, our approach takes into account not only time intervals, but also time instants. Our model is based on the observa-

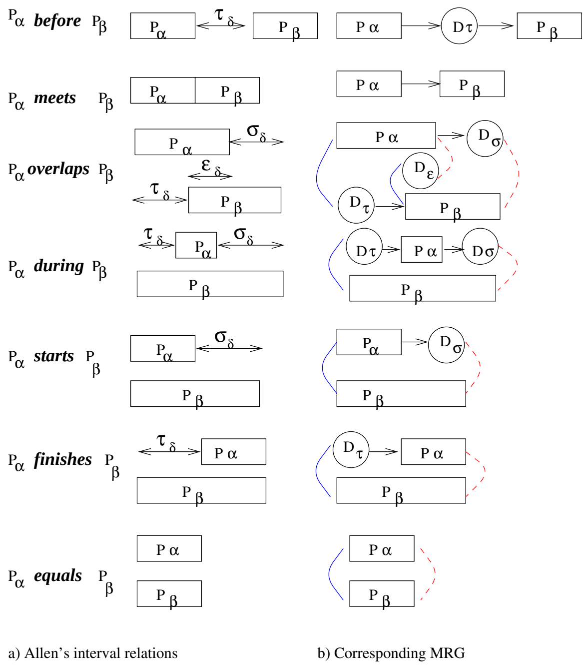

Figure 2: Allen's relations and MRG

tion that there are 12 relations between the 4 endpoints of 2 temporal intervals. The 12 relations are listed in Table 1. Note that there are two implicit relations that always hold between the endpoints, namely "a.start$<$$a$.end" and "$b$.start$<$$b$.end".

For the purpose of defining multimedia synchronization operators, it is useful to collapse the relations "a.end$<$$b$.start" and "$a$.end$=$$b$.start" into one relation "a.end$\leq$$b$.start", and the relations "a.start$=$$b$.end" and "$a$.start$>$$b$.end" into one relation "a.start$\geq$$b$.end". Therefore, we have reduced the 12 relations into 10 relations, which are listed in Table 2.

Note that the 10 relations are not mutually exclusive (Allen's relations are). The interrelations of the 10 relations are shown in the implication table (Table 3). The Venn diagram (Figure 3) illustrates the relationships graphically.

From the Venn diagram, we can see that some relations are disjoint, some of them have subset relationship, and yet others intersect but do not form subset relationship.

2.2 Media Relation Graph

Obviously, not all the 10 relations are needed to specify time relations in multimedia. Therefore, we define the most useful and intuitive 3 out of the 10 relations as MRG operators. The 3 relations are "a.end$\leq$$b$.start",

a.end$<$$b$.start
a.end$=$$b$.start
a.end$>$$b$.start
a.start$<$$b$.end
a.start$=$$b$.end
a.start$>$$b$.end
a.start$<$$b$.start
a.start$=$$b$.start
a.start$>$$b$.start
a.end$<$$b$.end
a.end$=$$b$.end
a.end$>$$b$.end

Table 1: 12 endpoints-based relations

a.end$\leq$$b$.start
a.end$>$$b$.start
a.start$<$$b$.end
a.start$\geq$$b$.end
a.start$<$$b$.start
a.start$=$$b$.start
a.start$>$$b$.start
a.end$<$$b$.end
a.end$=$$b$.end
a.end$>$$b$.end

Table 2: 10 reduced endpoints-based relations

"a.start$=$$b$.start", and "$a$.end$=$$b$.end"; they are named ***SerialLink***, ***StartSync***, and ***EndSync***, respectively. For (a *SerialLink* b), we call a the parent and b the child; for (a *StartSync* b) and (a *EndSync* b), we call a and b peers. Each of the 3 operators forms a temporal constraint between its operands.

The combination of the 3 MRG operators can express all the 10 relations. With the help of an intermediate interval i, we can express the remaining 7 relations, as illustrated below.

- (a *EndSync* i *StartSync* b) $\Rightarrow$ a.end$>$$b$.start
- (a *StartSync* i *EndSync* b) $\Rightarrow$ a.start$<$$b$.end
- (b *SerialLink* a) $\Rightarrow$ a.start$\geq$$b$.end
- (a *StartSync* i *SerialLink* b) $\Rightarrow$ a.start$<$$b$.start
- (b *StartSync* i *SerialLink* a) $\Rightarrow$ a.start$>$$b$.start
- (a *SerialLink* i *EndSync* b) $\Rightarrow$ a.end$<$$b$.end
- (b *SerialLink* i *EndSync* a) $\Rightarrow$ a.end$>$$b$.end

a.end≤b.start	⇒	a.start<b.end, a.start<b.start, a.end<b.end
a.end>b.start	⇒	no info
a.start<b.end	⇒	no info
a.start≥b.end	⇒	a.end>b.start, a.start>b.start, a.end>b.end
a.start<b.start	⇒	a.start<b.end
a.start=b.start	⇒	a.end>b.start, a.start<b.end
a.start>b.start	⇒	a.end>b.start
a.end<b.end	⇒	a.start<b.end
a.end=b.end	⇒	a.end>b.start, a.start<b.end
a.end>b.end	⇒	a.end>b.start

Table 3: Implication table of the 10 relations

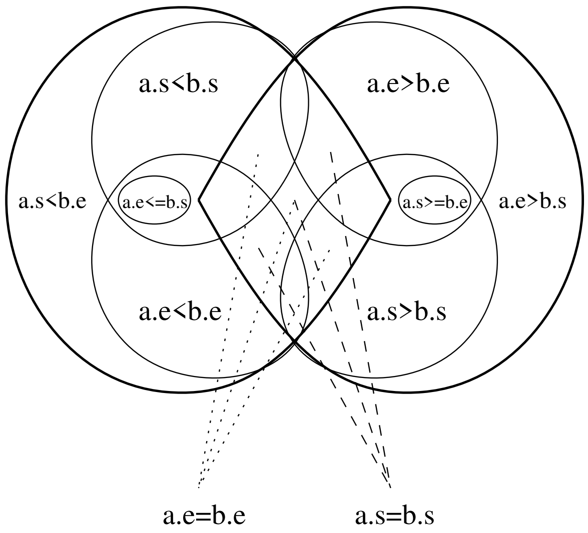

Figure 3: Venn diagram of the 10 relations

Intuitively speaking, (*a SerialLink b*) means objects *a* and *b* occur in sequence; and (*a StartSync b*) and (*a EndSync b*) mean objects *a* and *b* start and end at the same time, respectively.

In graph context, the three operators are represented by three kinds of edges in MRG. As shown in Figure 4a, a one-way arrow denotes the *SerialLink* operator, where the left hand side operand is the vertex at the starting end of the arrow and the right hand side operand is the vertex being pointed to by the arrow. Similarly, the *StartSync* operator is denoted by a solid line segment, and the *EndSync* operator is represented by a dashed line segment. Note that the DAG we introduced in section 1 (Figure 1) is a simplified version of MRG; the DAG lacks the constraints *StartSync* and *EndSync*.

There are two kinds of vertices in MRG. A rectangular vertex represents a regular media object and a round vertex represents a delay object, which does not have media type, content, or spatial layout. Finally, for an MRG

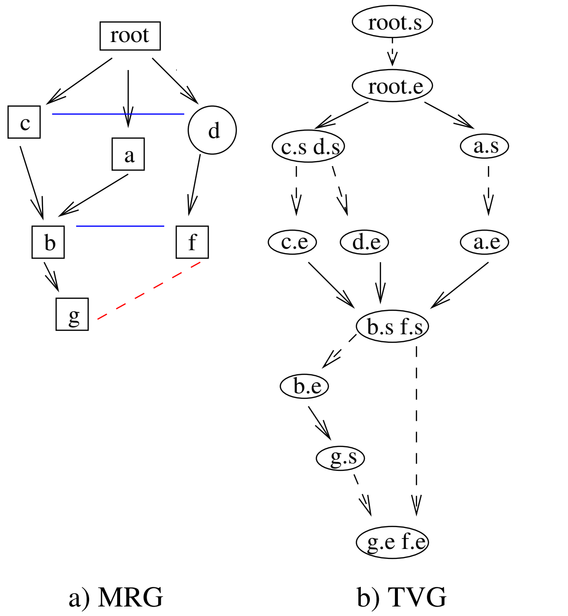

Figure 4: MRG vs. TVG

to model a complete HSL document, we also need a *root* vertex, which denotes the starting point of the hypermedia presentation defined by the HSL document. Since an object in an HSL document has an associated temporal interval and is represented by a vertex in MRG, the operands of the three MRG operators can be "object", "interval", or "vertex", depending on the context.

SerialLink is a best-effort operator. It tries its best to make the transition between its two operands instantaneous (the "=" part of *SerialLink*). The "<" part of *SerialLink* models nondeterministic delay, which is always minimized. For (*a SerialLink b*), what can cause b.start to be delayed? First, if we have (*c SerialLink b*) and a.end<c.end, then the start time of b will be delayed to the end time of c. We call this "V"-shape, as a, b, c, and the two *SerialLink* edges form a "V". This scenario can be extended to "W"-shape or "M"-shape, involving any number of parents and children. In general, the start time of an object is the latest end time (which is nondeterministic) of its parents. Second, if we have (*f StartSync b*) and a.end<f.start, then the start time of b will be delayed from the end time of a to the start time of f. Both situations are shown in Figure 4a. Note that the quantitative aspect (e.g. a.end<c.end and a.end<f.start) is not captured in MRG. On the whole, if an MRG is a tree (no multiple parents) and contains no *StartSync*s, all *SerialLink*s are instantaneous; i.e. "≤" becomes "=". *SerialLink* is transitive.

Both the $StartSync$ and $EndSync$ operators behave like rendezvous points. That is, if (*a StartSync b*), the start time of a and b is the greater of $starta$ and $startb$, where $starta$ and $startb$ are the start times of a and b without the $StartSync$ constraint. The same rule applies to $EndSync$. Note that $EndSync$ never affects the start times of its operands. $StartSync$ and $EndSync$ are transitive and symmetric.

In addition to the visible components (vertices and edges) of MRG, which are qualitative, each object may optionally define a quantitative attribute *ttl* (time to live), which specifies the lifetime of the object. For a text or image object, *ttl* specifies how long the object will be displayed; for an audio or video object, it enforces how long the object will be played, regardless of the object's natural content length; for a delay object, it specifies the amount of delay time introduced by the object. If *ttl* is unspecified, a text or image object will be displayed forever, and an audio or video object will be played until the end of its natural content.

With the definitions of $SerialLink$, $StartSync$, $EndSync$, *ttl*, and delay object, we can now use MRG to express Allen's 13 interval relations. This is illustrated in Figure 2b.

The MRG operators $SerialLink$, $StartSync$, and $EndSync$ are generally called synchronization arcs, or simply sync-arcs. The generalized sync-arc relates two time instants, called source and destination. In CMIF [14], the sync-arc modifies the destination; in the W3C multimedia standard SMIL [19], the sync-arc modifies the source. Both of them allow a delay to be specified on the arc. The former delays the destination, and the latter delays the source. The $SerialLink$ operator is a sync-arc from the end point of the first object to the start point of the second object, with a minimized nondeterministic delay applied to the destination. The above three types of sync-arcs are all binary operators. $StartSync$ and $EndSync$ operators are n-ary sync-arcs; i.e. they can be applied to a group of objects. This significantly reduces the amount of specification effort required. Although delays cannot be directly specified on the arcs, they can be emulated by a combination of delay objects and $SerialLink$s. For example, to specify f starts 5 seconds after c starts, we add a delay object d with *ttl* equal to 5 seconds, and relate c, f, and d by (*c StartSync d*) and (*d SerialLink f*) (Figure 4a).

3 Validation of MRG specifications

Temporal inconsistencies are easily introduced when authoring complicated multimedia documents. There are two major categories of inconsistencies, namely qualitative and quantitative. Qualitative inconsistencies are caused by conflicting temporal relations, while quantitative inconsistencies are caused by incompatible durations [12]. Due to the simplicity of MRG and its rendezvous-based operators, quantitative inconsistencies do not exist in our model; i.e. quantitative consistency is guaranteed by construction. Therefore, we only need to check for qualitative inconsistencies.

To facilitate the detection of qualitative inconsistencies, we first transform an MRG into a Temporal Validation Graph (TVG), which contains two types of vertices. A TVG "start" vertex contains one or more start points of the vertices in MRG; a TVG "end" vertex contains one or more end points of the vertices in MRG. The transformation satisfies the following rules:

1. For each vertex a in the MRG, there are two TVG vertices as and ae containing a.start and a.end, respectively; there is also a directed dashed edge from as to ae.
2. If (*a StartSync b*), a.start and b.start are in one TVG "start" vertex.
3. If (*a EndSync b*), a.end and b.end are in one TVG "end" vertex.
4. If (*a SerialLink b*), there is a directed solid edge from ae to bs.

Figure 4b shows the TVG corresponding to the MRG in Figure 4a. Note that the vertices along a path in a TVG alternate between "start" and "end" (edges from "start" vertices to "end" vertices are dashed, while edges from "end" vertices to "start" vertices are solid). TVG has the following important properties:

- If there is a path from as to bs, then a.start$<b$.start.
- If there is a path from ae to be, then a.end$<b$.end.
- If there is a path from as to be, then a.start$<b$.end.
- If there is a path from ae to bs, then a.end$\leq b$.start.

Therefore, to ensure the validity of a temporal specification represented by an MRG, we need to follow the following procedure:

- To add (*a SerialLink b*), there must be no path from bs to ae.
- To add (*a StartSync b*), there must be no path from as to bs, and no path from bs to as.

- To add (*a EndSync b*), there must be no path from ae to be, and no path from be to ae.

The above procedure can be implemented using standard reachability analysis (depth first search), therefore, the running time of each addition of MRG edges is linear (in terms of number of vertices and edges in the TVG); hence the validation of the whole MRG is a quadratic problem.

The consistency checking procedure is applied incrementally in the authoring stage (via our authoring tool). Since we allow the creation of HSL documents using text editors, we also need to apply the validation algorithm before presenting an HSL document. The validation procedure is applied to every temporal constraint (specified by one of the three MRG operators) in the document. If an inconsistent constraint is detected, the user is warned and the constraint is simply ignored.

If there is no path from either endpoint of one object to either endpoint of another object in a TVG, then there is no temporal relationship between the two objects. This usually means that the author does not care about the relationship between the two – which one starts first, and so on. If the author does care, he/she will add a constraint between the two objects in the corresponding MRG, whether implicitly (through transitivity) or explicitly.

Finally, from the properties of TVG, we can further derive two temporal overlapping rules:

- If $as = bs$ or $ae = be$, then a and b overlap in time.
- If there is a path from ae to bs, or from be to as, then a and b do not overlap in time.

4 Presentation scheduling

In order to ensure that media objects are presented in the specified order, a presentation scheduler needs to be developed. There are two types of hypermedia schedulers, namely compile-time scheduler and run-time scheduler [5]. A compile-time scheduler is static. It fixes the start and end times of objects, according to the temporal information specified in the hypermedia document; an optimum schedule may be generated by some form of quantitative analysis. The compile-time scheduler may also help prefetching, resource allocation and detection. A sophisticated compile-time scheduler may use heuristics and statistics to pre-arm hyperlinks [14]. On the other hand, a run-time scheduler is dynamic, and well-adapted to handling unpredictable behaviors (such as user interactions). It also constantly adjusts itself to match the changes in its execution environment.

Before proceeding further with presentation scheduling, let us make a distinction between various kinds of media objects in hypermedia systems, based on their start and end behavior:

- Bounded object
  The start and end times (or the start time and duration) of the object are known. For example, text and images with ttl specified, pre-recorded (stored) audio/video clips, etc. We call audio/video continuous objects and text/image discrete objects.
- Unterminated object
  The start time of the object is known, but the end time is unknown. For example, a live feed without a scheduled end time, a program execution (such as simulations and CGI scripts), etc.
- Unpredictable object
  The object may be started by a hyperlink and terminated by another hyperlink.

For most multimedia documents, the durations (ttl) of stored audio/video objects are not specified, so we cannot obtain their end times at the document level. First, if an audio/video object is remote, we could try to retrieve the meta information through a network protocol – using a special purpose video server which implements a protocol call that returns the intrinsic duration of a media object. However, we cannot rely on special protocols, as the Web is built on top of generic protocols like HTTP. We could also try to read the header of the media object to determine its timing information, but this is highly media-dependent (such as how many bytes we need to read). Moreover, there are media types whose headers do not have the necessary meta information (e.g. AVI). Even if we managed to obtain the intrinsic duration of a stored media object, the real play-out duration will likely vary under environmental conditions, such as slow or bursty network access and lack of client processing power. Second, if the audio/video object is local, we must obtain the timing information by reading the header of the media file. As described above, this is not always achievable. Furthermore, environmental constraints such as the speed of the client CPU also make the duration of the stored object a variable.

Because of the volatile nature of the Internet and the large variety of media types, all bounded objects that do not have ttl explicitly specified become unterminated objects. Hence, multimedia presentations on the Web are inherently nondeterministic.

Our conclusion is that the compile-time scheduler is only useful in a closed environment, such as where media objects are all stored locally and media types are all well understood by the system. Since our target environment is the Web, we choose to implement a run-time only scheduler, which handles unterminated and unpredictable objects, as well as bounded continuous and discrete objects.

Now let us proceed with the presentation algorithm. First, we need to describe the states of hypermedia objects in HPAS:

- Activated
  A visual object has appeared on the screen; an aural object has occupied an audio resource (such as an audio channel).
- Playing
  A continuous object is in progress.
- Paused
  A continuous object is temporarily paused.
- Content end
  A continuous object has reached the end of its natural content.
- *ttl* expired
  The author-specified lifetime of an object has been reached.
- Finished
  If *ttl* is defined, this state is the same as "*ttl* expired"; otherwise, it is the same as "content end".
- Deactivated
  A visual object has disappeared from the screen; an aural object has released its audio resource.

If the lifetime (*ttl*) of an object has expired before the end of its natural content, the object is immediately cut off from playing. What is the behavior of an object between the finished and deactivated states? For a discrete object, it stays on the screen until entering the deactivated state; for a video object, its last frame stays on the screen; for an audio object, it keeps its visual components (such as volume controls) visible, if any.

The activation of object a is governed by the following rules:

1. a's parents and the parents' $EndSync$ peers have all entered the deactivated state; and for each b, where b is one of a's $StartSync$ peers, b's parents and the parents' $EndSync$ peers have all entered the deactivated state.
2. a and all of its $StartSync$ peers enter the activated state at the same time, once Rule 1 is satisfied.

The deactivation of object a is governed by the following rules:

1. a and all of its $EndSync$ peers have entered the finished state.
2. a and all of its $EndSync$ peers enter the deactivated state at the same time, once Rule 1 is satisfied.

The object activation/deactivation policies translate to the following event-driven presentation algorithm, which is the core of our run-time scheduler.

```
onContentEnd() {
  if ttl unspecified
    onFinished()
}

onTTLExpired() {
  onFinished()
}

onFinished() {
  set this object's state to finished

  for p in (EndSync peers of this object)
    if p is not in the finished state
      return
  // now all EndSync peers are
  // in the finished state

  // deactivate this object
  // and all of its EndSync peers,
  // and activate their children if appropriate
  for o in (this object and its EndSync peers) {
    deactivate o

    for c in (children of o)
      if c satisfies the activation policy {
        // implies that c's StartSync peers
        // also satisfy the activation policy
        activate c and c's StartSync peers
        play c and c's StartSync peers
      }
  }
}
```

Either one of the "onContentEnd" and "onTTLExpired" event handlers may invoke "onFinished", depending on whether *ttl* is defined. The event handler "onFinished" tries to satisfy the deactivation policy in the first "for" loop; it then deactivates the object and its $EndSync$ peers, and activates their children if they satisfy the activation policy. Essentially, the "onFinished" event handler traverses the MRG in a stepwise, breadth-first fashion.

Besides common functions like "play" and "pause", our presentation scheduler also implements the "skip" and "jump" operations. The "skip" operation allows a user to deactivate all currently activated objects, and activate their children, if the children satisfy the activation policy. With "skip", the user can step through a presentation at a faster pace. The user may also use hyperlinks to "jump" to a future or past object in the presentation. When jumping to a future object, the scheduler first traverses the MRG until it finds the object, then it activates the object and its *StartSync* peers. For a past object, the whole presentation is first reset to its startup state, then the scheduler treats the past object as a future object and advances to it. The "jump" operation can also be used to start a presentation from the middle of a document. In that case, the starting point of the presentation is addressed by an object ID. The scheduler simply advances to the object and starts the presentation from there.

5 HSL document

HSL (Hypermedia Synchronization Language) is a SGML-conforming descriptive format. Its declarative syntax provides a simple yet powerful way to describe hypermedia presentations.

Let's consider the following sample code from an HSL document:

```
...
<comp> ... </comp>

<obj .../>

<comp id="Bird">
  <obj id="BirdShow" src="bshow.mpg"/>
  <obj id="BirdWalk" src="bwalk.mpg"/>
  <obj id="delay1" ttl="4s"
       seriallink='BirdIntro'/>
  <obj id="BirdIntro" src="bintro.html"
       ttl="20"/>

  <comp id="Seagull" seriallink='BirdSong'/>
    ...
  </comp>
  <obj id="BirdSong" src="bsong.wav"/>

  <startsync>
    BirdShow BirdWalk delay1
  </startsync>
  <endsync>
    BirdShow BirdWalk
  </endsync>
</comp>
...
```

The <obj> element represents a (atomic) media object; it cannot be nested. The <comp> element represents a composite object; it groups a set of objects (atomic or composite) together. <comp> may be nested to an arbitrary depth; therefore an HSL document has a tree-like document structure. Upon the activation of a <comp>, the first lexical element (<obj> or <comp>) within the <comp> will be activated. The top level construct of an HSL document (<body>) is modelled as an implicit composite object, so the presentation of the HSL document starts with the first lexical <obj> or <comp> in the document.

Each composite object represents exactly one MRG, with the vertices corresponding to the immediate children of the <comp>. The MRG may be disconnected. In that case, it is the union of two or more connected subgraphs. One of them is the subgraph containing the first lexical element of the <comp>; this subgraph represents the default or main presentation; all other subgraphs represent *atemporal* [8] presentations. The atemporal presentations can only be activated by hyperlinks (see section 6). Essentially, atemporal presentations allow users to choose among different paths (alternative presentations) within a document. In the HSL code segment above, "BirdShow", "BirdWalk", "delay1", and "BirdIntro" form the default or main presentation, while "Seagull" and "BirdSong" form an atemporal presentation.

6 Hyperlink

A hypermedia system must support extensive user interactions through *hyperlinks*. In HPAS, a hyperlink defines a relationship between two entities, namely the source anchor and the destination anchor. The source anchor is denoted by a hypermedia object within an HSL document; the destination anchor is much more flexible – it can be any entity addressable by a URL [2]. An author can specify the effect on the source when a hyperlink is followed. The default behavior is "overlay", which means the presentation containing the source is terminated immediately and replaced with the destination presentation. Alternatively, "spawn" means the system should start another window to present the destination, while keeping the source intact; "coexist" means the system should present the destination and the source side by side, if possible.

The destination of a hyperlink may be a future or past object in the current presentation, as we have described in the "jump" operation from section 4. In addition, the destination may represent an object in an atemporal presentation. The activation of such a hyperlink starts the atemporal presentation from the object represented by the destination anchor.

What we have described so far is the document level hyperlinks, which are defined by HSL. There is another level of hyperlinks, namely object level links. Examples include hyperlinks within a video stream, and the <a> element within an HTML object which has been embedded into an HSL document. Object level links require the knowledge of specific media types, so their behaviors are solely controlled by media handlers [18]; object level hyperlinks are not visible in the document layer.

7 Implementation

Advances in browser technology allow many new interesting applications to be written. Plugins and ActiveX controls extend browsers' capabilities seamlessly, and Java applets allow rapid development of platform independent applications. By exploiting features in LiveConnect [23] and Dynamic HTML [22], HPAS is able to reuse existing software components such as plugins and applets as media handlers. Continuous objects (audio/video) are played by the Java Media Player [21] (wrapped in an applet) and the RealPlayer [24] plugin; discrete objects (text/image) are rendered directly in the HTML browser.

To control the browser, applets, and plugins, HPAS uses the Java class "netscape.javascript.JSObject", which provides a handle to the JavaScript interpreter. Dynamic HTML allows HTML elements to be created, deleted, and modified by JavaScript on the fly. Therefore, to activate a media object, HPAS simply directs the JavaScript interpreter to output the appropriate HTML element, and the browser will either render the HTML element directly, or launch the appropriate plugin or applet to render it. Here are some example HTML elements emitted by HPAS:

```
<!-- text object -->
<object id="gold"
style="position:absolute;left:4;top:200;
       width:320;height:240"
data="fish.html">
</object>

<!-- image object -->
<img id="silver"
style="position:absolute;..."
src="flower.jpg">

<!-- audio/video object -->
<object id="copper"
style="position:absolute;..."
classid="java:hpas.mhandler.jmf">
<param name="MediaFile" value="run.mpg">
</object>

<!-- RealAudio/RealVideo -->
<object id="iron"
style="position:absolute;..."
data="realworld.rpm">
</object>
```

To deactivate a media object, HPAS asks the JavaScript interpreter to delete the HTML element with the corresponding ID. To control a media handler (applet or plugin), HPAS gets the Java object representing the media handler from JavaScript, and then calls whatever public methods are available (such as "play" and "pause") from that Java object.

HPAS is implemented as a Java applet with a set of Java classes, which can be either stored locally or downloaded on the fly before presenting an HSL document. Running as a Java applet also allows multiple HSL documents to be played in multiple browser windows at the same time. Because HPAS runs inside HTML browsers, to present an HSL document, an HTML wrapper is needed:

```
<html><head>...</head><body>
<applet name="hpas" code="hpas.AppletMain"
mayscript width="0" height="0">
  <param name="src"
   value="http://www.goldfish.com/demo.hsl">
</applet>

<div id="layout"
style="position:absolute;
       left:100;top:200;
       width:640;height:480"></div>

<!-- other HTML goes here -->
</body></html>
```

The <applet> element contains the invisible HPAS applet. It is invisible because HPAS controls are displayed as popup windows, thus leaving all the space in the browser window for media rendering. The <div> element defines the area in which an HSL presentation will be displayed. Typically the <div> occupies the whole browser window, but in the above example, it starts from (100, 200) and has the size 640×480. The <div> element must have the ID "layout", so that the HPAS applet may access it from JavaScript. An author can also define other static HTML elements in the HTML wrapper, which will have nothing to do with HPAS.

The HTML wrapper approach also facilitates the use of external layout. Instead of the single <div> element with ID "layout", the author defines a series of <div> elements, each corresponds to an HSL object (with the same ID). Those <div> elements define the geometry of the corresponding HSL objects; therefore, no layout information is needed in the HSL document itself. This external layout mechanism allows a single HSL document to be reused with different spatial layouts. The idea is analogous to applying different stylesheets to an HTML document.

Since HPAS is in the form of a Java applet, it can be controlled by JavaScript code embedded in HTML documents. For example, to start playing an HSL presentation, the following JavaScript statement can be used:

```
document.applets['hpas'].play();
```

Figure 5: A scene from an HSL presentation

As a consequence, an author may choose to create his/her own user interface (using a combination of HTML and JavaScript) instead of HPAS' default Java AWT interface.

Figure 5 shows a running HSL presentation (the surrounding Netscape browser window frames and borders are cropped out).

The W3C multimedia standard SMIL [19] addresses many similar issues in hypermedia synchronization; therefore, a converter has been implemented (within HPAS applet) to present SMIL documents in the HPAS environment. Since SMIL and HSL use different temporal models, a few SMIL features will be missing after the conversion.

The current browser/Java-based implementation consists of less than 10,000 lines of Java code, while the original Unix/C-based implementation has around 30,000 lines of C/C++ code. Why is there such a big difference? First, we are now reusing existing software as media handlers; second, Java provides many useful utilities, such as Vector and Hashtable, which saved us from rewriting them from scratch.

The Java version of HPAS is available at

`http://www.research.digital.com/SRC/HPAS/.`

8 Conclusions and future work

In the past two and a half years we have been working on the HPAS project to support the presentation and composition of time-based hypermedia documents. The current implementation provides services for integrating and reusing pluggable components such as Java applets and browser plugins. Hypermedia objects rendered by those software components are synchronized both temporally and spatially during the presentation stage of HSL documents.

The system is well suited for presenting dynamic and interactive information on the Web, such as product/service advertisements, distance learning and self-guided course work, entertainment information, etc.

Currently, the authoring tool is still based on Unix/C, so we are planning to rewrite it as a standalone Java application. In the near future, we would like to implement Document Object Model (DOM) [20] for HSL/SMIL. Applying DOM on top of HSL/SMIL will allow authors to create highly interactive time-based hypermedia documents.

9 Acknowledgment

I am indebted to Monika Henzinger, for her suggestion on graph transformation (from MRG to TVG), and to Yuan Yu, for the tireless discussion on the temporal model. Finally, I would like to give special thanks to the reviewers Paul McJones, Marc Najork, and Krishna Bharat for their timely advice on the content and structure of the paper.

References

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

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

[3] G. Blakowski and R. Steinmetz. A Media Synchronization Survey: Reference Model, Specification, and Case Studies. *IEEE Journal on Selected Areas in Communications*, vol.14, no.1, pp. 5-35, January 1996.

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

[5] M.C. Buchanan and P.T. Zellweger. Automatic Temporal Layout Mechanisms. *Proceedings of ACM Multimedia'93*, pp. 341-350, August 1993.

[6] K.S Candan, B. Prabhakaran, and V.S. Subrahmanian. CHIMP: A Framework for Supporting Distributed Multimedia Document Authoring and Presentation. *Proceedings of ACM Multimedia'96*, pp. 329-340, November 1996.

[7] L. Hardman, G. van Rossum, and D.C.A. Bulterman. Structured Multimedia Authoring. *Proceedings of ACM Multimedia'93*, pp. 283-289, August 1993.

[8] L. Hardman, D.C.A. Bulterman, and G. van Rossum. The Amsterdam Hypermedia Model. *Communications of the ACM*, vol.37. no.2, pp. 50-62, February 1994.

[9] M. Jourdan, N. Layaida, and L. Sabry-Ismail. Time Representation and Management in MADEUS: an Authoring Environment for Multimedia Documents. *Proceedings of Multimedia Computing and Networking 1997*, pp. 68-79, February 1997.

[10] C. Keramane and A. Duda. Interval Expressions - a Functional Model for Interactive Dynamic Multimedia Presentations. *Proceedings of IEEE ICMCS'96*, pp. 283-286, June 1996.

[11] M.Y. Kim and J. Song. Multimedia Documents with Elastic Time. *Proceedings of ACM Multimedia'93*, pp. 143-154, August 1993.

[12] N. Layaida and L. Sabry-Ismail. Maintaining Temporal Consistency of Multimedia Documents Using Constraint Networks. *Proceedings of Multimedia Computing and Networking 1996*, pp. 124-135, January 1996.

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

[14] G. van Rossum, J. Jansen, K.S. Mullender, D.C.A. Bulterman. CMIFed: A Presentation Environment for Portable Hypermedia Documents. *Proceedings of ACM Multimedia'93*, pp. 183-188, August 1993.

[15] J. Schnepf, J.A. Konstan, and D.H.C. Du. Doing FLIPS: FLexible Interactive Presentation Synchronization. *IEEE Journal on Selected Areas in Communications*, vol.14, no.1, pp. 114-125, January 1996.

[16] B. Travis and D. Waldt. *The SGML Implementation Guide*. Springer-Verlag, 1995.

[17] T. Wahl and K. Rothermel. Representing Time in Multimedia Systems. *Proceedings of IEEE ICMCS'94*, pp. 538-543, May 1994.

[18] J. Yu and Y. Xiang. Hypermedia Presentation and Authoring System. *Proceedings of the 6th International WWW Conference*, pp. 153-164, April 1997.

[19] W3C SYMM Working Group. Synchronized Multimedia Integration Language (SMIL). http://www.w3.org/TR/REC-smil/.

[20] W3C DOM Working Group. Document Object Model Specification.
http://www.w3.org/TR/WD-DOM/.

[21] Intel Media for Java.
http://www.intel.com/ial/jmedia/.

[22] Dynamic HTML in Netscape Communicator.
http://developer.netscape.com/library/documentation/communicator/dynhtml/.

[23] LiveConnect, in Netscape JavaScript Guide, chapter 5.
http://developer.netscape.com/library/documentation/communicator/jsguide4/livecon.htm.

[24] RealPlayer 4.0.
http://www.real.com/products/player/.