

# Design and implementation of the just-in-time retrieving policy for schedule-based distributed multimedia presentations

Chun-Chuan Yang<sup>\*</sup>, Yi-Zheng Yang

*Multimedia and Communications Laboratory, Department of Computer Science and Information Engineering, National Chi-Nan University,  
1 University Rd, Puli, Nantou Hsein 545, Taiwan, ROC*

Received 29 May 2002; received in revised form 24 September 2002; accepted 27 September 2002

## Abstract

In order to provide the smooth playback of a distributed multimedia presentation, the object-retrieving engine for the player must fetch each object before its playback time. In this paper, a smart object-retrieving engine is proposed, which adopts a retrieving policy named the just-in-time policy. The policy expects the retrieval process of an object to finish right before the playback time of the object to achieve a better buffer utilization and network bandwidth efficiency. The proposed object-retrieving engine focuses on SMIL1.0-based multimedia presentations. By converting the synchronization relationship of objects in the SMIL1.0 document to Real-Time Synchronization Model, which simplifies the handling of the synchronization relationship, and considering the end-to-end bandwidth as well as the user interactions, the object-retrieving engine determines the object request time for each object. The engine issues the request to fetch each object for the ongoing presentation at the object request time, and provides the player with proper media objects. The feasibility and better performance of the proposed just-in-time retrieving policy had been proved by performance measurements of system implementation.

© 2002 Elsevier Inc. All rights reserved.

**Keywords:** Distributed multimedia presentation; User interaction; Synchronized Multimedia Integration Language (SMIL); Real-Time Synchronization Model (RTSM); Object retrieving

## 1. Introduction

With the development of high-speed networking and computer technologies, multimedia information systems like the *distributed multimedia presentation* (Shih and Davis, 1997; Moreno and Mayer, 1999; Liew et al., 1999; Bertino et al., 2000; Palacharla et al., 1997; Carneiro et al., 1999) are becoming more accessible to a variety of audiences. The distributed multimedia presentation is concerning with viewing a presentation via the network. The presentation system should provide VCR-like functions, such as *play/stop*, *pause/restart*, *fast forward/backward*, and *sliding*, to support user interactions. Some or all the media objects in the presentation could be somewhere in the network instead of the local site. Issues for the distributed multimedia presentation

include: the overall design of the system (Shih and Davis, 1997; Moreno and Mayer, 1999; Liew et al., 1999; Bertino et al., 2000), the scheduling (retrieving) of the objects and resource management under the network environment (Palacharla et al., 1997; Carneiro et al., 1999; Hwang and Prabhakaran, 2000), the mechanisms for dealing with the temporal relationship (synchronization) of objects in the presentation (Jeong et al., 1997; Huang and Wang, 1998; Cruz and Mahalley, 1999; Song et al., 1999; De Lima et al., 1999), the supporting of the user interactions (Bertino et al., 2000; Huang et al., 1998; Yoon and Berra, 1998; Jeng et al., 1999), etc.

The common objective of the issues for the distributed multimedia presentation is to provide a smooth playback for viewing the presentation. Since the objects in a multimedia presentation may reside in remote data servers, the critical problem is to guarantee that temporally related objects from different data servers and different network channels will play synchronously at the client site—even with random user interactions. The

<sup>\*</sup> Corresponding author. Tel.: +886-49-910960-4131; fax: +886-4-211-6689.

E-mail address: [ccyang@csie.nctu.edu.tw](mailto:ccyang@csie.nctu.edu.tw) (C.-C. Yang).

quality of the presentation depends on the ability of the player in dealing with the network behavior and user interactions. More specifically, when a presentation is ongoing, the *object-retrieving engine* for the player (or the player itself) should retrieve proper media objects before the object's playback time according to the synchronization relationship of the presentation. Hence, the object-retrieving engine plays an important role on the quality of the presentation.

Two extreme policies for object retrieving could be adopted. First, retrieve all objects before starting the presentation, or second, do not make the request to retrieve the object until the object's playback time. The first policy is called the *pre-loading policy*, and the second policy is called the *passive-loading policy* in the paper. The pre-loading policy guarantees the smooth playback in the cost of a long initial delay and a large buffer for all objects in the presentation. Moreover, since the viewer could activate hyperlinks or VCR-like functions in an ongoing presentation, it is improper to prefetch all media objects in advance. Minimal buffer is required for the passive-loading policy, but it introduces gaps between the request time and the finish time of object retrieval because of the network delay. It implies that the smooth playback is impossible for the passive-loading policy.

The proposed object-retrieving engine adopts a better policy that is called the *just-in-time policy*. The policy expects the retrieval process for an object to be finished right before the playback time of the object so that the player could continue the presentation smoothly. Under such policy, the object-retrieving engine only buffers necessary objects for the smooth progress of the presentation, thus it has a better buffer utilization and network bandwidth efficiency.

The main problem for the just-in-time object-retrieving policy is the computation (estimation) of the request time for object-retrieval made by the object-retrieving engine. The request time for retrieving an object is called the *object request time* in the paper. Two factors need to be considered in the computation of the object request time: (1) the playback time for the object and (2) the total time for retrieving the object via the network. The playback time for each object depends on the temporal relationship of the objects in the presentation and the user interactions. The total time for retrieving the object from the remote server depends on the size of the object and the effective network bandwidth from the server to the client. Under the just-in-time retrieving policy, the object request time could be estimated as the playback time of the object minus the total time for retrieving the object.

Before we further discuss the mechanisms for supporting the just-in-time policy, the representation of the multimedia presentation should be addressed. In order to compose a multimedia presentation, a method (e.g. a

script language) is necessary to specify the spatial relationship and the temporal (synchronization) relationship for the objects in the presentation. Different multimedia presentation systems usually have different formats of presentation. However, it is better to adopt a popular language as the format of the multimedia presentation (Chang, 1999). From the popularity point of view, SMIL (*Synchronized Multimedia Integration Language*) (W3C, 1998; W3C, 2001) seems to be a good candidate. SMIL was developed by the *WWW Consortium (W3C)* to address the lack of HTML for multimedia over WWW. It provides an easy way to compose multimedia presentations. With the efforts of W3C, SMIL is becoming a popular language in authoring multimedia presentations, and it is currently supported by the newest versions of the commercial browsers.

This paper focuses on the SMIL-based presentations and proposes proper mechanisms for the just-in-time retrieving policy. Currently, two versions of SMIL specification had been proposed by W3C: *SMIL1.0* (W3C, 1998) and *SMIL2.0* (W3C, 2001). The SMIL1.0 model is primarily a scheduling model, but with some flexibility to support continuous media with unknown duration. SMIL2.0 extends SMIL1.0 and introduces event-based activation or termination to let users define a dynamic activation path. Because of dynamic activation of elements, a SMIL2.0 presentation's entire timing might be changed at the run-time, which makes it difficult to implement the just-in-time retrieving policy. Therefore, the proposed object-retrieving engine deals with schedule-based (SMIL1.0) presentations only. Research of object retrieving for event-based (SMIL2.0) presentations is left as the future work.

The object-retrieving engine has to handle the synchronization relationship of the objects in the SMIL script. In order to deal with the synchronization relationship efficiently during the parsing process for the SMIL script, a synchronization model (Little and Ghafoor, 1989; Qazi et al., 1993; Yang and Huang, 1996; Lin et al., 1998; Prabhakaran and Raghavan, 1993; Huang et al., 1998; Yoon and Berra, 1998; Jeng et al., 1999), which provides a systematic view of the synchronization relationship, is helpful for the object-retrieving engine. In the literature, *DEFSM (Dynamic Extended Finite State Machine)* (Huang et al., 1998) is proposed to model the synchronization relationship for interactive multimedia presentations. However, the approach requires two models, an "actor" DEFSM and a "synchronizer" DEFSM, for an interactive multimedia presentation, and is hence complicated.

A more compact and Petri-net based model namely *OCPN (Object Composition Petri Net)* (Little and Ghafoor, 1989; Qazi et al., 1993) was proposed to model the synchronization relationship among media objects. Extensions of OCPN for supporting user interactions (Yoon and Berra, 1998; Jeng et al., 1999) were also

proposed. However, we had shown that OCPN or other Petri-net based models are not suitable for real-time network applications, and instead Real-Time Synchronization Model (RTSM) was proposed (Yang and Huang, 1996). Moreover, the synchronization behaviors that could be specified by SMIL1.0 make RTSM more suitable than Petri-net based models. For example, SMIL allows the author to set the explicit beginning time, duration, or end time for an element in the presentation. RTSM could easily model this kind of the synchronization behavior, but OCPN could not. Therefore, RTSM is adopted to model the synchronization relationship of the multimedia presentation scripted by SMIL.

To sum up, the proposed object-retrieving engine first parses the input SMIL document to extract and to represent the synchronization relationship of the objects in the presentation by RTSM. Next, the object request time for each object is determined by considering the user interaction, the playback time of the object, and effective network bandwidth to retrieve the object. The request for retrieving the object is then issued by the object-retrieving engine at the object request time.

The remainder of the paper is organized as follows. First, we make a brief survey for SMIL1.0 and RTSM in Section 2. The overview of the process for the object-retrieving engine is presented in Section 3. The conversion of the synchronization relationship of SMIL to RTSM during the parsing process is presented in Section 4. The calculation of the object request time is explained in Section 5. In Section 6, the implementation and some results of performance measurement for the proposed object-retrieving policy are presented. Finally, Section 7 concludes this paper.

## 2. Previous work

### 2.1. Survey of SMIL1.0

As mentioned in Section 1, SMIL was developed by W3C to address the lack of HTML for multimedia over WWW. With the introduction of SMIL, Web multimedia creators have a new tool for building time-based multimedia presentations that combine audio, video, images, and text. The proposed SMIL standard defines an XML-based language that allows control over what, where, and when of media elements in a multimedia presentation with a simple clear markup language similar to HTML.

In other words, SMIL could be used to describe both the spatial relationship and the temporal relationship of a multimedia presentation. The spatial relationship is concerning with the visual layout of media objects in the presentation, while the temporal relationship is concerning with timing control of media objects. The ele-

ments for spatial relationship in SMIL1.0 include the  $\langle layout \rangle$  element and the  $\langle region \rangle$  element. The  $\langle layout \rangle$  element determines how the elements in the document's body are positioned. The  $\langle region \rangle$  element controls the position, the size, and scaling of media object elements.

The synchronization elements in SMIL1.0 for the temporal relationship include the  $\langle seq \rangle$  element, the  $\langle par \rangle$  element, and the class of media object elements such as  $\langle img \rangle$ ,  $\langle video \rangle$ ,  $\langle audio \rangle$ , and  $\langle text \rangle$ . The  $\langle seq \rangle$  element defines a sequence of elements in which elements play one after the other. The  $\langle par \rangle$  element defines a simple parallel time grouping in which multiple elements can play back at the same time. Both  $\langle seq \rangle$  and  $\langle par \rangle$  allow the nested structure that means the children element of them could be any of the synchronization elements. The media object elements allow the inclusion of media objects into an SMIL presentation. Media objects are included by reference (using a *URI*).

Besides, some synchronization related attributes such as “*begin*”, “*dur*”, and “*end*” could be associated with these synchronization elements. The “*begin*” attribute specifies the time for the explicit begin of an element. The “*end*” attribute specifies the explicit end of an element. The “*dur*” attribute specifies the explicit duration of an element.

### 2.2. Survey of real-time synchronization model

RTSM (Yang and Huang, 1996) was proposed to address the lack of Petri-net based models for dealing with real-time synchronization. The elements in RTSM include *place*, *token*, and *transition* as in OCPN (Little and Ghafoor, 1989). However, there are two kinds of places in RTSM, *regular places* and *enforced places*. A different firing rule for enforced places is defined. The rule specifies that once an enforced place becomes unblocked (in other words, the related action with the place is completed), the transition following the place will be immediately fired regardless the states of other places feeding the same transition.

An example of RTSM is shown in Fig. 1 in which a single circle is for the regular place, a double circle is for the enforced place, and a bar is drawn for the transition.

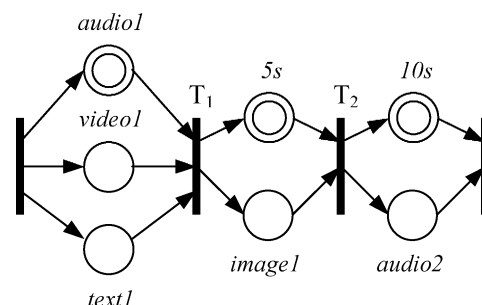


Fig. 1. An example of RTSM.

The RTSM in Fig. 1 requires that audio segment *audio1*, the video clip *video1*, and the text data *text1* be played simultaneously. Since *audio1* is an enforced placed, transition  $T_1$  is fired right after *audio1* is finished, no matter if *video1* or *text1* is finished or not. After firing transition  $T_1$ , *image1* is displayed for 5 s then transition  $T_2$  is fired. Finally, *audio2* is played for 10 s after transition  $T_2$  fires. Note that the enforced place of “5 s” in the figure is not a media object but a virtual medium that is called *Time Medium* (Yang and Huang, 1996). The time medium is used to represent time duration.

### 3. Overview of the object-retrieving process

The object-retrieving process is responsible for retrieving proper media objects and dealing with user actions for the presentation. The operation model of the proposed object-retrieving engine is illustrated in Fig. 2. The object-retrieving engine first accepts the SMIL script from the player, and converts the synchronization relationship to RTSM. By analyzing the RTSM model and collecting object related information from data servers, the playback duration for each object under normal play mode is calculated. The object-retrieving engine then uses the obtained playback duration of each object as a base to handle the user actions and determines the object request time for objects. Proper objects are retrieved and then delivered to the player by the object-retrieving engine according to the object request time.

Functions of user interactions supported by the proposed object-retrieving engine include *play*, *stop*, *pause/restart*, *fast forward*, *fast backward*, and *sliding*. To de-

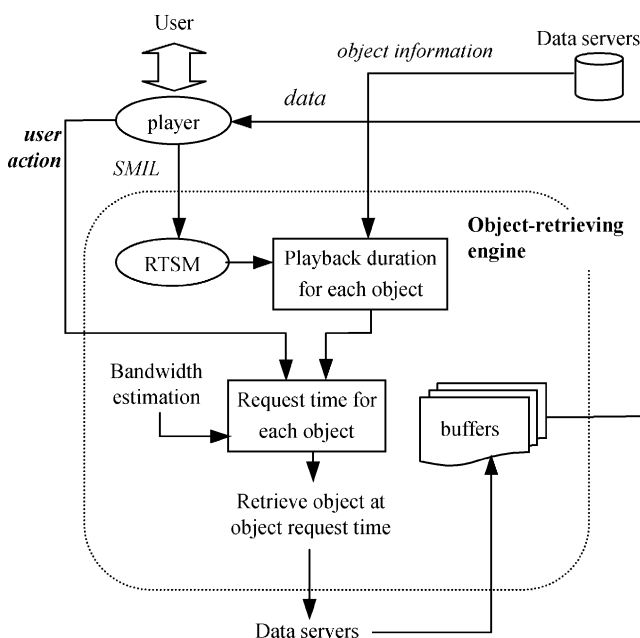


Fig. 2. Overview of the object-retrieving process.

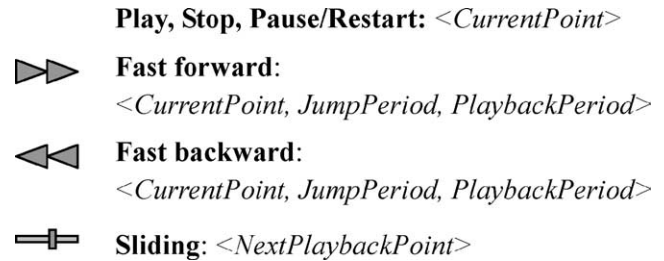


Fig. 3. User actions and corresponding parameters.

fine the user interactions precisely, some parameters are associated with the user actions as shown in Fig. 3. For example, three parameters should be passed to the object-retrieving engine for the fast forward mode: *CurrentPoint*, *JumpPeriod*, and *PlaybackPeriod*. *CurrentPoint* is used to indicate the time point within the overall presentation time at which the user action is made. It is a relative value timed from the start of the presentation, and it means fast forwarding of the presentation should be started from the point indicated by *CurrentPoint*. *JumpPeriod* and *PlaybackPeriod* are used to define how does the player achieve the fast forward/backward operation as explained in the following.

Since it is impossible to speed the presentation of a multimedia document physically as in the VCR system, we need a new way to define fast forward/backward operation. We define *JumpPeriod* as the period to be skipped in fast forward/backward operation and *PlaybackPeriod* as the period to be played. By using these two parameters, we could achieve the functionality of fast forward/backward, and the play speed of the presentation becomes  $(JumpPeriod + PlaybackPeriod) / PlaybackPeriod$ . The playback patterns for fast forward and fast backward are shown in Fig. 4 in which the only difference between fast forward and fast backward is the reverse ongoing directions. As for the sliding action, the object-retrieving engine only needs to know the next playback point, which is denoted by *NextPlaybackPoint*, so that it could retrieve proper objects.

When some user action is made, the object-retrieving engine tries to locate the objects that should be played next under the new mode. However, there are always cases that only part of an object needs to be played, such as a sub-segment of an audio object or a sub-clip of a video object. So it is assumed that data servers have the ability of sub-sampling a continuous object like audio or video, and the object-retrieving engine only retrieves necessary part of the object instead of retrieving the whole object.

### 4. Converting SMIL to RTSM

In this section, we present the algorithm for converting the input SMIL script to RTSM. Since the

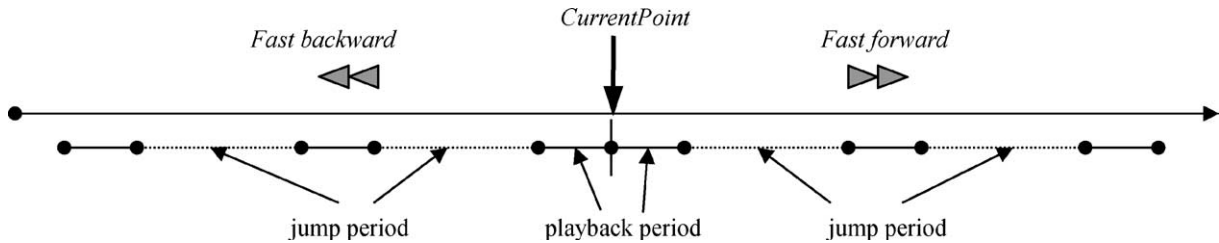


Fig. 4. Playback patterns for fast forward/backward.

temporal relationship is the main concern of RTSM, only the synchronization elements in SMIL1.0 are examined. There are three kinds of synchronization elements to be converted: the  $\langle seq \rangle$  element, the  $\langle par \rangle$  element, and the class of media object elements such as  $\langle img \rangle$ ,  $\langle video \rangle$ ,  $\langle audio \rangle$  and  $\langle text \rangle$ . Besides, some synchronization related attributes such as “begin”, “dur”, and “end” could be associated with these synchronization elements. We assume that the player has checked the syntax of the SMIL document, so the object-retrieving engine only performs necessary conversion.

#### 4.1. Converting the $\langle seq \rangle$ element

The  $\langle seq \rangle$  element defines a sequence of elements in which elements play one after the other. The children elements of the  $\langle seq \rangle$  element could be any of the synchronization elements such as  $\langle seq \rangle$ ,  $\langle par \rangle$ , or the media object elements, so the conversion is a recursive procedure. Since the children of a  $\langle seq \rangle$  element form a temporal sequence, we concatenate each child of  $\langle seq \rangle$  one by one in RTSM as illustrated in Fig. 5. Note that there are *virtual places* (denoted by the dashed circle) in the figure. They are used to maintain the consistency of RTSM, since the arc could only be the link between a transition and a place. In fact, the virtual place is a regular place that maps to the time medium with zero duration.

#### 4.2. Converting the $\langle par \rangle$ element

The  $\langle par \rangle$  element defines a simple parallel time grouping in which multiple elements can play back at the same time. Thus, all children of  $\langle par \rangle$  should be within the same pair of transition ( $T_s$ ,  $T_e$ ) as illustrated in Fig. 6. There are three variations for  $\langle par \rangle$  since a special

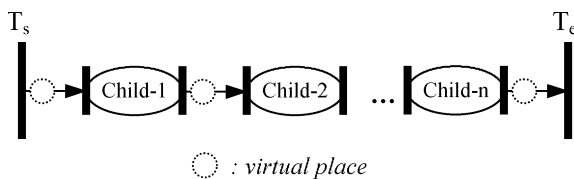


Fig. 5. Convert the  $\langle seq \rangle$  element to RTSM.

attribute, “endsync”, could be associated with  $\langle par \rangle$ . The “endsync” attribute controls the end of the  $\langle par \rangle$  element, as a function of children. Legal values for the attribute are “last”, “first”, and “id-ref”.

The value of “last” requires  $\langle par \rangle$  to end with the last end of all the child elements, and the corresponding RTSM is shown in Fig. 6(a), in which transition  $T_e$  could not be fired unless all children end. The value of “first” requires  $\langle par \rangle$  to end with the earliest end of all the child elements. Therefore, we should change the places between each child element and transition  $T_e$  to *virtual enforced places* as illustrated in Fig. 6(b) so that the child that ends first will fire transition  $T_e$ . A virtual enforced place is an enforced place that maps to the time medium with zero duration. The value of “id-ref” requires  $\langle par \rangle$  to end with the specified child. So we change the place between the specified child and transition  $T_e$  to the virtual enforced place as shown in Fig. 6(c).

Other synchronization attributes, such as “begin”, “end”, and “dur”, could also be associated with  $\langle seq \rangle$  and  $\langle par \rangle$ , but the conversion is similar to that in the media object elements that we present in the following.

#### 4.3. Converting the media object elements

The media object elements allow the inclusion of media objects into a SMIL presentation. Media objects are included by reference (using a URI). One media object element naturally represents a regular place in RTSM. However, the attributes associated with the element require some more complex conversion. We examine and convert attributes “begin”, “end”, and “dur” respectively in the following.

##### 4.3.1. Converting the “begin” attribute

This attribute specifies the time for the explicit begin of an element. Two types of values could be contained in the attribute: *delay-value* and *event-value*. A delay value is a clock-value to postpone the playback time of the element by the delay value. Therefore, one enforced place representing the delay time with the specified duration is added in front of the element as illustrated in Fig. 7(a). The event-value requires the element begin when a certain event occurs. According to the

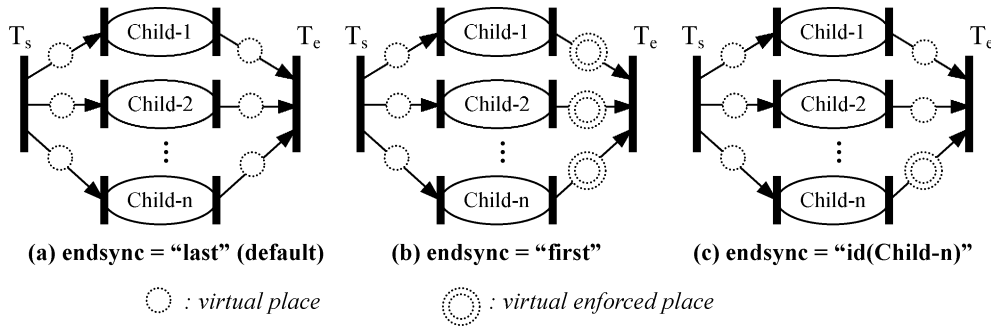


Fig. 6. Convert the  $\langle par \rangle$  element to RTSM.

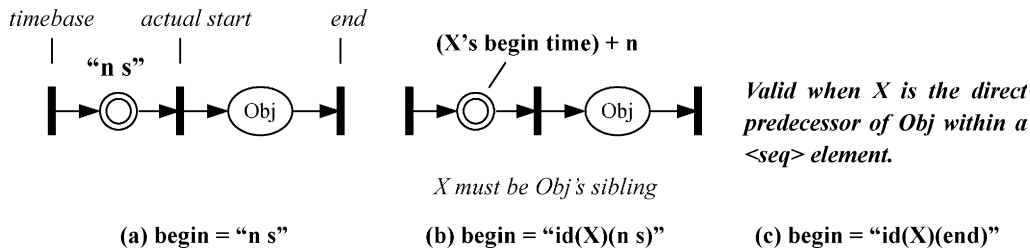


Fig. 7. Effect of "begin" attribute on RTSM.

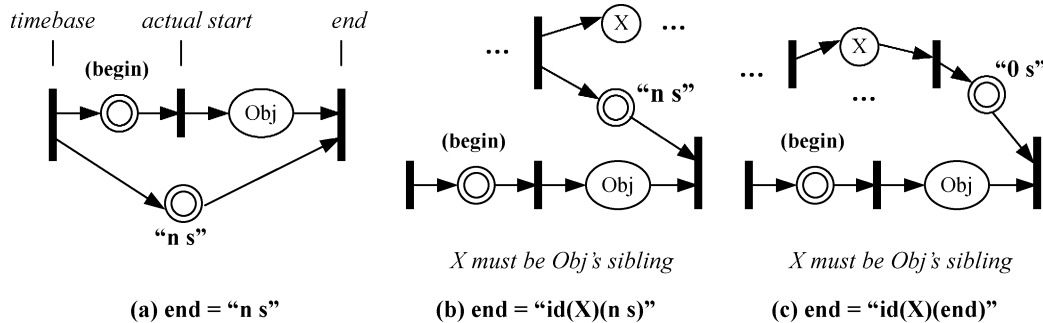


Fig. 8. Effect of "end" attribute on RTSM.

specification of SMIL1.0, the element  $X$  generating the event must be "in scope", in other words,  $X$  must be a sibling of the element that contains the "begin" attribute. There are two variations for the event-value, which is shown in Fig. 7(b) and (c) respectively.

In Fig. 7(b), the " $id(X)(n s)$ " value of "begin" attribute means that element  $Obj$  begins after its sibling  $X$  has begun for  $n$  seconds. So one enforced place representing the delay time with value of summation of " $X$ 's begin time" and " $n$  seconds" is added in front of  $Obj$  element. Actually, from the semantic point of view, the case in the Fig. 7(b) is only valid when elements  $X$  and  $Obj$  are child elements in a  $\langle par \rangle$  element. The other value of event-value for "begin" attribute is " $id(X)(end)$ ", which means the  $Obj$  element begins right after element  $X$  ends. The case is actually the function of  $\langle seq \rangle$ . Therefore, the value is only valid when  $X$  is the direct predecessor of the element  $Obj$  in a  $\langle seq \rangle$  element, and it introduces nothing to RTSM.

#### 4.3.2. Converting the "end" attribute

This attribute specifies the explicit end of an element. There are also three possible values for the attribute as in the "begin" attribute. We illustrate the conversion of them in Fig. 8. In Fig. 8(a), a clock value represents the end time from the original timebase of the element. So one enforced place with the clock value is added between the original start transition and the end transition. In Fig. 8(b), the " $id(X)(n s)$ " value of "end" attribute means that element  $Obj$  ends when its sibling  $X$  has begun for  $n$  seconds. Therefore, there is a enforced place between the actual start transition of  $X$  and the end transition of  $Obj$ . Finally, Fig. 8(c) illustrates the case of value " $id(X)(end)$ ", which means element  $Obj$  must end when element  $X$  ends.

#### 4.3.3. Converting the "dur" attribute

This attribute specifies the explicit duration of an element. Therefore, the value of the "dur" attribute, which

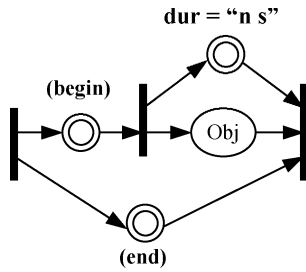


Fig. 9. Effect of “dur” attribute on RTSM.

```

<seq>
  <par  endsync = id(URI-1)>
    <audio src=URI-1 />
    <video src=URI-2 />
    <text src=URI-3 />
  </par>
  <img src=URI-4 dur= "5s" />
  <audio src=URI-5, dur= "10s" />
</seq>
    
```

Fig. 11. A SMIL code snippet.

is a clock value, forms an enforced place between the actual start transition and the end transition. We illustrate the effect of the attribute in Fig. 9.

#### 4.4. Hyperlinks in SMIL

As specified in the SMIL1.0 specification, the hyperlink element  $\langle a \rangle$  is transparent when playing the presentation until the user activates the link and starts a new presentation. Thus, only child elements of the element  $\langle a \rangle$  are converted to RTSM, which is the same as previous sections.

#### 4.5. Simplifying the obtained RTSM

As mentioned in the above section, some virtual (enforced) places are added in the RTSM during the converting process. However, there are some cases of RTSM that could be simplified by applying three rules, which are shown in Fig. 10. First, if the only input of a transition is a media place and the only output of the transition is a virtual place, we could naturally replace the case with the media place only since the virtual place is actually dummy (Fig. 10(1)). Second, if the only output of the transition in rule (1) is a virtual enforced place, it means the firing of the media place will enforce to fire the following transition of the virtual enforced place. We could replace the case by changing the media place to an enforced one as shown in Fig. 10(2). Third, if there is only one virtual place between two transitions,

the two transitions could be combined into one transition as shown in Fig. 10(3).

#### 4.6. An example for the conversion

Fig. 11 shows a sample SMIL code snippet, in which only temporal information is displayed. The initial RTSM right after the converting process is shown in Fig. 12 and the simplified RTSM for the example is shown in Fig. 13.

SMIL1.0 also introduced the “repeat” attribute, which is used to repeat a media element or an entire time container, such as  $\langle seq \rangle$  or  $\langle par \rangle$ . With the presence of the “repeat” attribute, the RTSM model for the element is copied for the number of times specified by the value of the “repeat” attribute.

### 5. Determine the object request time

The object request time indicates the time to issue the request to retrieve the object. However, the object playback time depends on the play mode, which is dynamically activated by the user. The object request time thus needs to be re-computed each time the user action is made. Before computing the object request time for the user action, the object-retrieving engine has to know the *original playback duration* for each object under normal play mode (i.e. play speed = 1). The original playback duration is then used as a base to compute the object request time as well as the new playback time for each object under specific user action. The original playback duration is computed by traversing the RTSM obtained from the converted process mentioned in Section 4.

#### 5.1. Compute the original playback duration

The playback time for an object is actually the firing time of the starting transition, and the end time of the object is the firing time of the ending transition of the object in RTSM. To compute the firing time for each

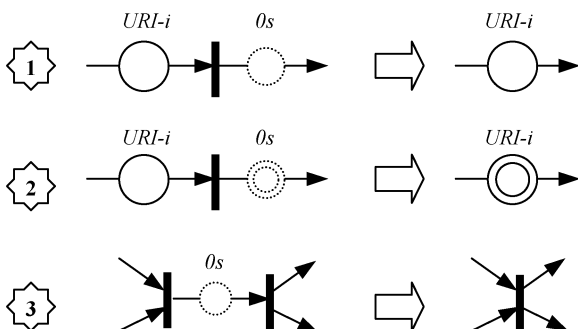


Fig. 10. Three rules to simplify RTSM.

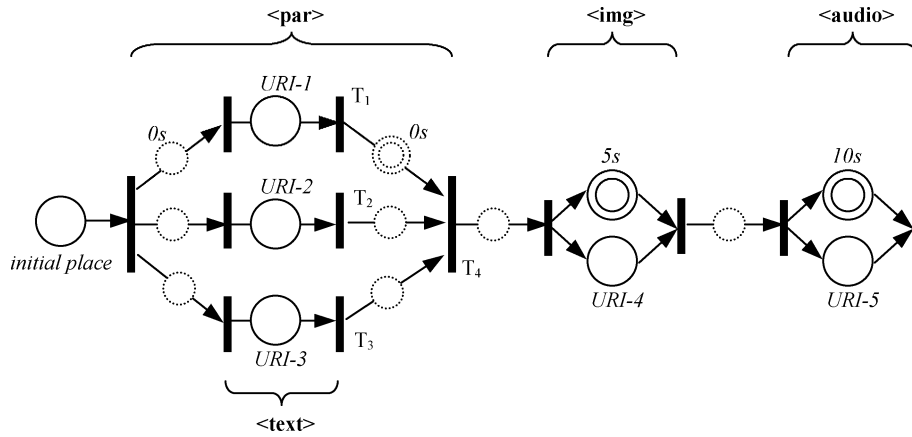


Fig. 12. RTSM for the sample SMIL snippet.

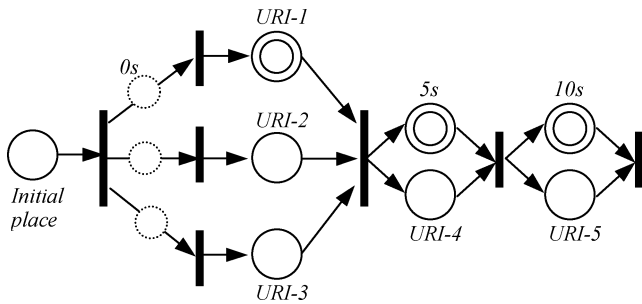


Fig. 13. Simplified RTSM for the sample SMIL snippet.

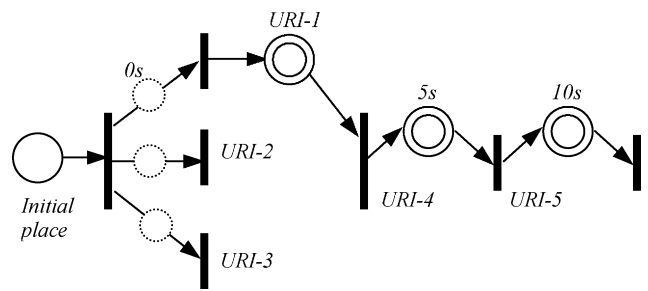


Fig. 14. Reduced RTSM for the sample SMIL snippet.

transition, we have to traverse the RTSM. Since there is usually more than one place that feeds to a transition, the behavior of a transition depends on the type of places that feed into it. If a transition is fed by some enforced places, the enforced places will dominate the behavior of the transition. In other words, if a transition is fed by some enforced places, the other regular places cannot affect the firing time of the transition at all. Therefore, we reduce the RTSM by removing the regular places that feed to a transition with enforced places. The reduced RTSM for the example in Fig. 13 is shown in Fig. 14.

The firing time for each transition is then computed by traversing the reduced RTSM. There are only two possibilities for one transition in the reduced RTSM: (1) places that feed to the transition are all enforced places, or (2) places that feed to the transition are all regular places. For case (1), the firing time of the transition is the minimal value of “the firing time of the preceding transition” + “the duration of the following place”, which is illustrated in Fig. 15(a). The firing time of the transition for case (2) is instead the maximum value of its predecessors as illustrated in Fig. 15(b). The duration of each place depends on the type of the media object. For an enforced place of time medium, the duration of the place is the value of the time duration. For static media objects, such as *<img>* and *<text>*, the duration of the

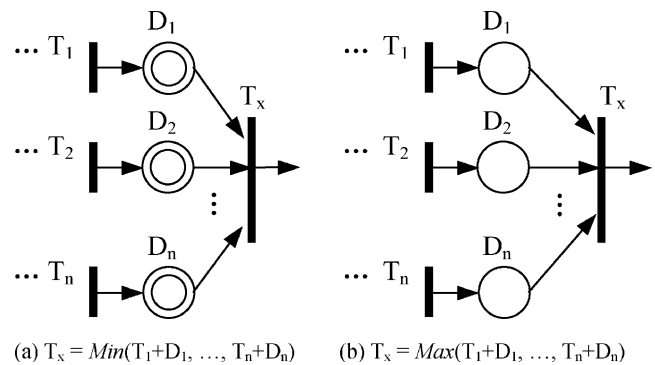


Fig. 15. Determine the firing time for transition  $T_x$ .

place is zero. For continuous media object like *<audio>* and *<video>*, the duration of the place is the implicit duration of the object that is provided by the data server. Since the objects stored in a data server are all pre-orchestrated, it is easy for the data server to obtain the implicit duration of a continuous object.

After the traversing process mentioned above, there may be some cases of inconsistency that the firing time of a transition is later than the firing time of its following transition. The reason is that removing the regular places of a transition with some enforced places in the reduction stage only affects the firing time of the following

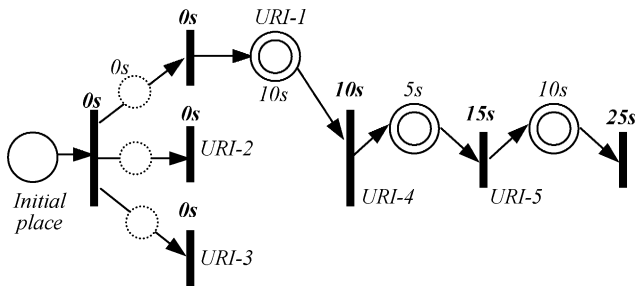


Fig. 16. The firing time for each transition for the sample SMIL snippet.

transitions. However, the enforced firing of a transition should also make all the preceding transitions fire simultaneously. Thus, the solution to remove the inconsistency is to replace the firing time of a transition with the firing time of its following transition while the firing time of the transition is later than that of its following transition. Assuming that the intrinsic duration of the audio object *URI-1* in Fig. 14 is 10 s, the firing time of each transition for the example is shown in Fig. 16.

### 5.2. Calculate the object request time for user actions

The objects that should be retrieved (played) depend on the user action since different user actions result in different playback patterns and different playback times of objects. In this section, the mechanism to determine the proper object (or part of an object) to be played as well as the object request time for each user action is presented.

As mentioned in Section 5.1, the data server provides the intrinsic duration of each object to determine the original playback duration. Moreover, in order to determine the proper objects to play and the object’s playback time as well as the object request time under possible user actions, the object-retrieving engine has to collect other information for each object from the data server. When the object-retrieving engine accepts the SMIL script, it sends probe packets to all data servers for asking the object information, which includes *object size*, *estimated bandwidth*, and *play rate*.

The data server estimates the bandwidth for the object to the object-retrieving engine. The *estimated bandwidth* is used to estimate the delay of an object from the server to the client. There are two factors that affect the item: (1) the transmission rate of the server for the requested object, and (2) the effective network bandwidth of the path from the server to the client. The transmission rate depends on the load of the server and the capacity of the outgoing link. The effective network bandwidth could be measured by some bandwidth measuring mechanisms (Bolliger and Gross, 1999; Lai and Baker, 1999; Paxson, 1999). The estimation of the end-to-end bandwidth for retrieving an object is beyond

the scope of the paper. If the server cannot provide information about the estimated bandwidth, it should inform the client to perform the estimation by itself.

The *play rate*, which is only valid for continuous objects, indicates the amount of data that is played within unit time for a continuous object and is used for locating any part of the object. We denote the object size for object *URI-i* as  $Size_{URI-i}$ , the estimated bandwidth as  $EstBW_{URI-i}$ , and the play rate as  $PlayRate_{URI-i}$ .

In addition to object related information, the object-retrieving engine also has to estimate the time for the request packet arrived to the data server. We use the round trip delay, denoted by  $RTDelay_{URI-i}$  as the estimated value for the delay of the request packet to the server. Thus, the total time to retrieve an object is the summation of the delay of the request packet and the transmission time of the object from the data server to the client site. That is, the retrieving time for object *URI-i* is estimated as  $(Size_{URI-i}/EstBW_{URI-i}) + RTDelay_{URI-i}$ . The object-retrieving engine then fills in the *object information table*, which is shown in Fig. 17, as a reference for computing the object request time.

#### 5.2.1. Play action

For each action made by the user, the player passes the action ID and associated parameters to the object-retrieving engine as presented in Section 3. *CurrentPoint* is the only parameter that is passed to the engine, and it is a time point within the total presentation time to indicate the starting point of the playback. The default action for viewing a presentation should be the play action with  $CurrentPoint = 0$  s, i.e. the beginning of the presentation.

The original playback duration in the object information table is used as a reference to identify the objects that should be played for the user action. Note that the values of the original playback duration ( $PBTime_{URI-i}$ ,  $PBEnd_{URI-i}$ ) for an object in the table are relative to the beginning ( $CurrentPoint = 0$  s) of the presentation.

| ID    | By RTSM |        | By Data Server |        |          |         |
|-------|---------|--------|----------------|--------|----------|---------|
|       | PBTime  | PBEnd  | Size           | EstBW  | PlayRate | RTDelay |
| URI-1 | 5 sec   | 10 sec | 40KB           | 10KBps | 8KBps    | 0.1 sec |
| URI-2 | ...     | ...    | ...            | ...    | ...      | ...     |
| ...   |         |        |                |        |          |         |

*(PBTime, PBEnd)*: original playback duration for the object

*EstBW*: estimated bandwidth for the object from the server to the client

*PlayRate*: only valid for continuous objects like audio or video

*RTDelay*: round-trip delay measured by the object-retrieving engine

Fig. 17. The object information table.

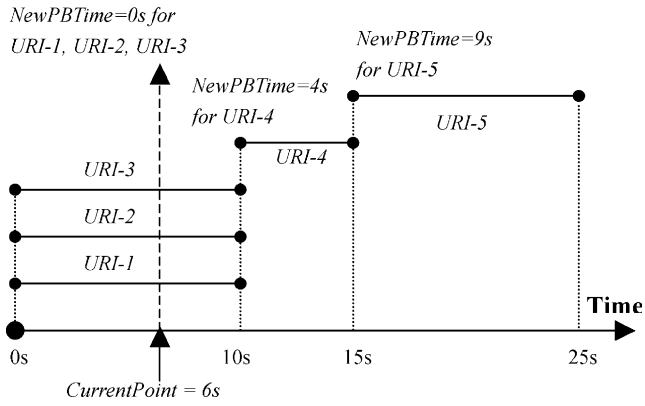


Fig. 18. An example for the play action.

Therefore, objects with original playback time later than *CurrentPoint* should be played for the play action, and the new playback time for the object is  $PBTime_{URI-i} - CurrentPoint$ , which is denoted by  $NewPBTime_{URI-i}$ . In other words, the object should be played  $NewPBTime_{URI-i}$  seconds later after starting the playback for the new user action. An example for illustration of the new playback time for the play action is shown in Fig. 18. Finally, the object request time is computed as  $NewPBTime_{URI-i} - (Size_{URI-i}/EstBW_{URI-i} + RTDelay_{URI-i})$ .

Because of the random user actions, there are cases that *CurrentPoint* does not align to the  $PBTime_{URI-i}$  of an object but within the original playback duration of the object. In such case, the new playback time for the object becomes 0 s (i.e. beginning of the new playback) and the retrieving pattern depends on the type of the object. For static objects like images and text, since they are not temporally divisible, the object-retrieving engine should still retrieve the whole object. For continuous objects like video and audio, only part of the object should be retrieved. From time point of view, the retrieved part of the continuous object, which is called *sub-object*, is related to the new playback duration for the object. The new playback duration is actually from *CurrentPoint* to  $PBEnd_{URI-i}$ . For instance, the new playback duration for *URI-1* (audio) and *URI-2* (video) in Fig. 18 is (6 s, 10 s). The size for the sub-object denoted by  $SubSize_{URI-i}$  is computed as  $(PBEnd_{URI-i} - CurrentPoint) * PlayRate_{URI-i}$ . Therefore, the retrieving time for the sub-object becomes  $(SubSize_{URI-i}/EstBW_{URI-i} + RTDelay_{URI-i})$ . An illustration for the calculation of a sub-object is shown in Fig. 19. The object request time for the sub-object is then computed as  $NewPBTime_{URI-i} - (SubSize_{URI-i}/EstBW_{URI-i} + RTDelay_{URI-i})$ .

The calculation of the object request time is based on the *NewPBTime* for the object. Since the value of *NewPBTime* is relative to the starting time of the new playback for the user action, the object request time is also relative to the starting time of the playback. A negative value of the object request time implies that the request for the object should be issued before the play-

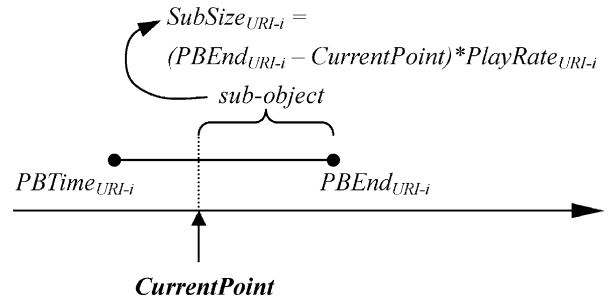


Fig. 19. Illustration for locating a sub-object.

back of the presentation for the user action. Therefore, when the user action is made, the playback of the presentation should be delayed by the maximum negative value of all the object request times, which is the initial delay for the user action.

### 5.2.2. Pause/restart

When the user makes the pause action, the object-retrieving engine stops all the pre-fetching process and waits for the next user action. The restart action is assumed to follow the pause action to continue the presentation in play mode. Therefore, the *CurrentPoint* parameter associated with the restart action indicates the next playback point of the presentation. The operation of the object-retrieving engine is thus the same as that of the play action.

### 5.2.3. Fast forward

Two parameters, *JumpPeriod* and *PlaybackPeriod*, define the forwarding pattern and *CurrentPoint* indicates the starting point of the presentation as mentioned in Section 3. In order to identify the objects or sub-objects that should be retrieved under this action, we have to scan the original playback duration in the object information table from *CurrentPoint* to the end of the presentation. The summation of *JumpPeriod* and *PlaybackPeriod* is used as the cycle time to scan the original playback durations. The overlapping duration of an object with *PlaybackPeriod* is the sub-object to be retrieved and to be played.

We use the example in Fig. 20 to illustrate the iterative process. The playback duration for each object, which is derived from the sample SMIL document, is shown in the upper part of the figure. It is assumed that the user made the fast forward action while the playback of the presentation reaches the time point of 6th second, and *JumpPeriod* and *PlaybackPeriod* are 5 and 2 s respectively. Therefore, the duration from 6th to 8th second is the first part of the presentation to be played, 13th to 15th second is the next part to be played, and then 20th to 22nd second, etc., as shown in the figure. The above time periods are then used to locate the part of an object to be retrieved, and the pre-fetch timetable for the fast forward action is set up as shown in Fig. 21.

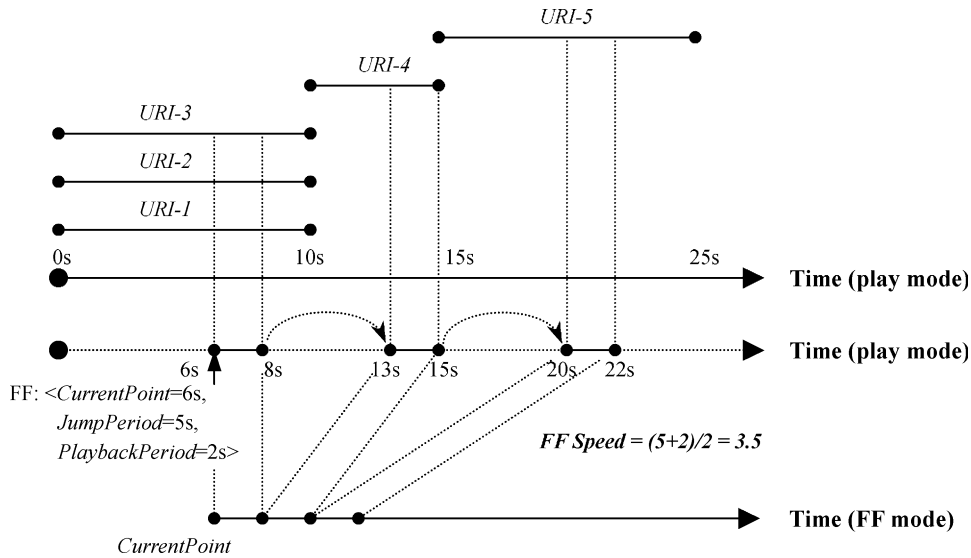


Fig. 20. An example of fast forward operation.

Note that in the pre-fetch table in Fig. 21, the sub-object fields for *URI-3 (text)* and *URI-4 (image)* are invalid since they are static objects, which means the whole object for *URI-3* and *URI-4* should be retrieved.

The retrieving time for sub-object *URI-i* with duration *PlaybackPeriod* is calculated as  $(PlaybackPeriod * PlayRate_{URI-i}) / EstBW_{URI-i} + RTDelay_{URI-i}$ . The new playback time for each sub-object depends on the count of iteration in which the object is scanned, and it is calculated as  $PlaybackPeriod * IterationCount$ . The pre-fetch time (object request time) is then computed as the new playback time minus the retrieving time of the sub-object. After setting up the pre-fetch timetable, the object-retrieving engine makes requests to retrieve objects (or sub-objects) according to the pre-fetch time for each

*CurrentPoint*: 6s

**Action**: fast forward, *JumpPeriod*=5s, *PlaybackPeriod*=2s

Relative to *CurrentPoint*

| Obj ID | sub-object | NewPBTime | end time | pre-fetch time                |
|--------|------------|-----------|----------|-------------------------------|
| URI-1  | 6s ~ 8s    | 0s        | 2s       | $NewPBTime - retrieval\ time$ |
| URI-2  | 6s ~ 8s    | 0s        | 2s       | $NewPBTime - retrieval\ time$ |
| URI-3  | N/A        | 0s        | 2s       | $NewPBTime - retrieval\ time$ |
| URI-4  | N/A        | 2s        | 4s       | $NewPBTime - retrieval\ time$ |
| URI-5  | 5s ~ 7s    | 2s + 2s   | 6s       | $NewPBTime - retrieval\ time$ |

Fig. 21. Pre-fetch timetable.

object. Note that when a new user action is made, the object-retrieving engine updates the pre-fetch timetable with the new computed pre-fetch time for each object.

#### 5.2.4. Fast backward

The operation of the object-retrieving engine for the fast backward action is similar to that of the fast forward action, and the only difference is that the direction of the scanning process is reverse as shown in Fig. 4. Computation of the retrieving time and the pre-fetch time is the same as in the fast forward action. Note that the object-retrieving engine is only responsible for retrieving proper data for the player, so the ability of reversing the playback of continuous objects depends on the player. If the player cannot provide the reverse playback of the object data, it should just play the retrieved data in the normal (forward) direction.

#### 5.2.5. Sliding

When the user uses the slider to change the playback point of the presentation, the player must determine the next playback point and passes the value (*NextPlaybackPoint*) to the object-retrieving engine. Thus the value of *NextPlaybackPoint* indicates the time point of the presentation to be played. It is assumed that the sliding action results in the normal playing mode from the new playback point, so the operation of the object-retrieving engine is the same as that of the play action.

#### 5.2.6. Discussion

If the data server supports the streaming mechanism for continuous objects, it is not necessary to retrieve the whole content of the object before its playback time. Only the amount of data to support the streaming operation is required. Thus, the transmission time of

the object is  $BufferSize_{URI-i}/EstBW_{URI-i}$ , in which  $BufferSize_{URI-i}$  is the amount of data to buffer. The value of  $BufferSize_{URI-i}$  depends on the streaming operation and is not addressed in this paper.

As presented in Sections 5.2.1 and 5.2.3,  $PlayRate_{URI-i}$  for continuous objects provided by the data server is used to calculate the size ( $SubSize_{URI-i}$ ) of the retrieved sub-object. However, it was assumed that the encoding scheme for continuous objects (audio/video) generates constant-bit-rate (CBR) data. For variable-bit-rate (VBR) encoding schemes, the calculation of the size for sub-objects depends on the encoding scheme and is thus more complicated, which is beyond the scope of the paper.

The object-retrieving engine has to re-compute the pre-fetch time of objects for each time a new user action is made, so the computation time of the algorithm to calculate the pre-fetch time is critical for the performance of the object-retrieving engine. The scanning process for determining the objects to be retrieved and for computing the pre-fetch time is only one pass for all objects in the presentation, so the algorithm only takes linear computation time.

The accuracy of end-to-end bandwidth estimation affects the performance of the object-retrieving engine as well as the quality of the presentation. Since the network behavior is very dynamic, it is impossible to exactly estimate the time required to finish the retrieving process for a media object. Thus, we discuss the impact of the accuracy of estimated time to finish the retrieving process for a media object on the performance of the object-retrieving engine.

If the estimated time is more pessimistic (bandwidth is underestimated) than the actual status, the object will be buffered for some time before its playback time. On the other hand, if the estimated time is more optimistic (bandwidth is overestimated) than the actual status (e.g. network is congested), the presentation will probably be paused to wait for the object. Furthermore, if the network bandwidth could be reserved in advance by some booking method, the estimated object request time will be more precise. Hence, the quality of the presentation and the buffer utilization will also be improved.

## 6. Implementation and performance measurement

We have implemented an experimental system for the feasibility and performance evaluation of the proposed object-retrieving engine. The network environment for the experimental system is shown in Fig. 22. There are two data servers (free web sites) in the system: one is for providing the audio data, and the other is for providing Image/HTML data. All the three object-retrieving policies, *pre-loading*, *passive-loading*, and *just-in-time* are included in the system for performance comparison. The

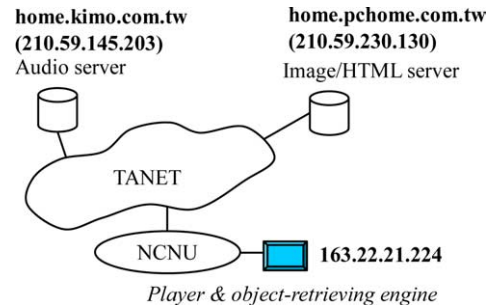


Fig. 22. Network environment for the experimental system.

performance criterion is the jitters between the playback time and the arrival time of the object (i.e.  $jitter = actual\ playback\ time - arrival\ time$ ). A positive value of the jitters implies the buffering time for the object before playback, while a negative value of the jitters indicates the pause time of the playback to wait for the object to arrive. Thus, the value of the jitters closer to zero implies a better performance.

The test SMIL document for performance measurement includes six child elements in the root  $\langle seq \rangle$  element and each child element is composed of three media elements (audio, image, and text) with duration 10 s. That is, there are six audio objects, six image objects, and six text objects in the 60-s presentation. The size of the audio objects is 80 K bytes, the size of the image objects is within the range of 20–40 K bytes, and the size of the text objects is within the range of 0.5–13 K bytes.

Since the data servers are free web sites that cannot provide the advanced service like bandwidth estimation, the object-retrieving engine in the system has to estimate the bandwidth by itself. For the just-in-time retrieving policy, the engine measures the average bandwidth from the data server to the client by requesting some test files from the server. The  $EstBW$  obtained is then used in the calculation of the object request time as presented in Section 5. Some of the performance measurements are displayed in Figs. 23–25 in which the object-retrieving engine assumed that the *play* action is enabled after loading the test SMIL document ( $CurrentPoint = 0$  s). Figs. 26–28 show the relationship between the estimated bandwidth and the actual end-to-end bandwidth experienced by each object under the just-in-time policy.

The pre-loading policy retrieves all the objects before starting the playback of the presentation, so the later the playback time of an object (which has a larger ID in the test SMIL file), the larger jitters (more buffering time) the object will experience. Therefore, the jitters for the pre-loading policy form a monotonic ascending curve in the figures. On the other hand, the jitters for the passive-loading policy are always negative as shown in the figures, and the value of the jitters depends on the traffic condition and the size of the requested object.

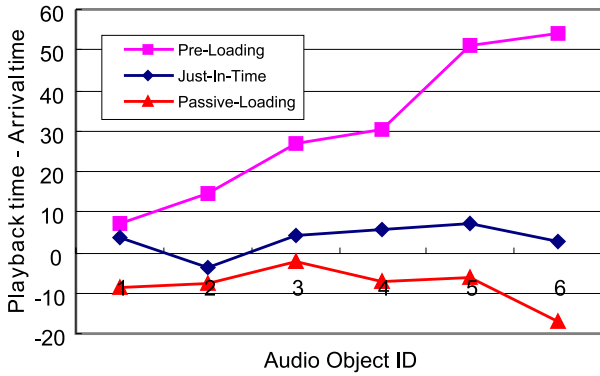


Fig. 23. Jitters of audio objects.

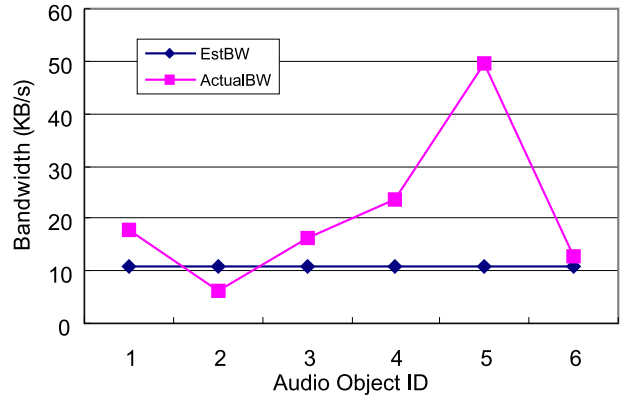


Fig. 26. Estimated BW vs. actual BW for audio.

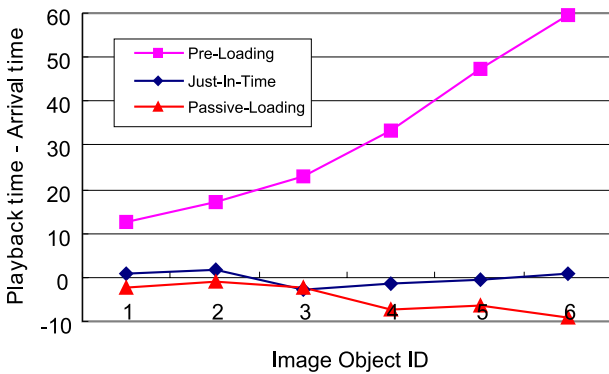


Fig. 24. Jitters of image objects.

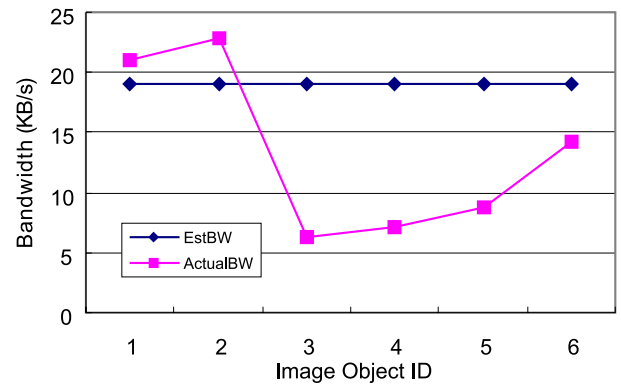


Fig. 27. Estimated BW vs. actual BW for image.

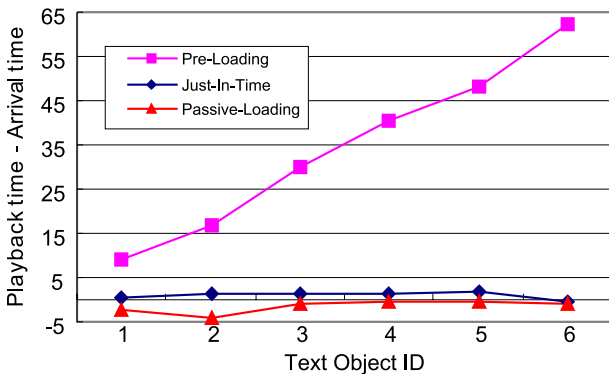


Fig. 25. Jitters of text objects.

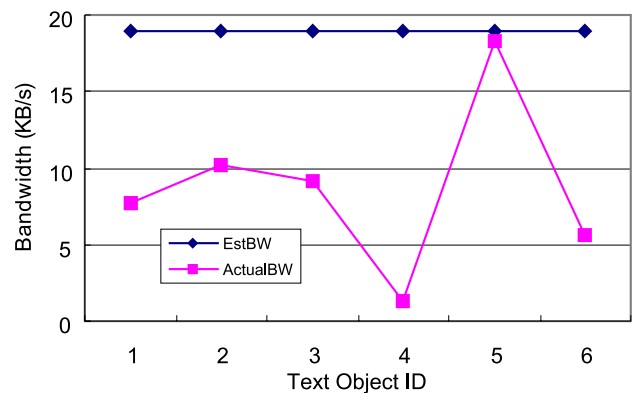


Fig. 28. Estimated BW vs. actual BW for text.

Figs. 26–28 show that the actual end-to-end bandwidth fluctuates when the object-retrieving engine retrieves media objects from the servers, so the measured average bandwidth for computing the object request time is imprecise. However, the performance of the just-in-time policy is still better than that of the other two policies, which justifies the idea of the just-in-time policy. Actually, under the same network traffic condition, the jitters of the just-in-time policy should always be smaller than those of the other two extreme policies,

which means the better performance for the just-in-time policy could always be achieved. Some observations for the impact of the bandwidth estimation on the performance of the just-in-time policy are presented in the following.

The end-to-end bandwidth for audio objects ID 1 and 3–6 in Fig. 26 was underestimated, i.e. the estimated bandwidth was less than the actual bandwidth, so all the jitters for these objects in Fig. 23 were above the *X-axis*

(i.e. positive values of jitters). On the other hand, the overestimation of the end-to-end bandwidth for audio object ID 2 resulted in a negative value of the jitter. However, no matter how the bandwidth is under- or overestimated, the jitters tend to be near the *X-axis* (i.e. close to zero) in comparing with the other policies. It is the inherency of the just-in-time policy.

It is worth mentioning that the end-to-end bandwidth for image objects ID 3–6 were overestimated as displayed in Fig. 27, but the jitter for image object ID 6 was positive. The reason is because the calculation of the object request time for the just-in-time policy also considers the latency for the request packet to arrive the server as presented in Section 5.2. The amount for this latency is estimated as the measured round-trip delay. Thus, the conservative value of the latency compensates the effect of the overestimated end-to-end bandwidth. Similar situations happened in the case of text objects (Figs. 25 and 28).

## 7. Conclusion

Distributed multimedia presentation enables the users to view a multimedia presentation in the distributed manner, in which the objects of the presentation are located at remote sites. In order to provide the smooth playback of the presentation, the object-retrieving engine for the player must fetch each object before its playback time. In this paper, the just-in-time retrieving policy for distributed multimedia presentations was proposed. The policy expects the object-retrieving process to finish right before the playback time of the objects. Mechanisms, which include the converting algorithm of the synchronization relationship to RTSM, calculation of the original playback duration for each object, and the estimation of the object request time, to support the just-in-time policy were proposed.

The contributions of the paper are listed as follows:

- (1) The features of the distributed multimedia presentation were investigated, and the just-in-time policy with corresponding mechanisms to support object retrieving was proposed.
- (2) The converting algorithm of the synchronization relationship for the SMIL1.0-based multimedia presentation to RTSM was proposed. The conversion provides an easier and systematic way to deal with the temporal relationship of the objects in the presentation.
- (3) Modeling of the user interactions such as fast forward, fast backward, and sliding, etc., for distributed multimedia presentations was proposed. Procedures of the object retrieving for handling the random user actions during the playback of the presentation were also presented in the paper.

- (4) The feasibility and the better performance of the proposed just-in-time policy had been proved by system implementation and performance measurements.
- (5) Although this paper focuses on the SMIL-based multimedia presentations, the proposed methodology could be also applied to non-SMIL presentations.

## References

- Bertino, E., Ferrari, E., Stolf, M., 2000. MPGS: an interactive tool for the specification and generation of multimedia presentations. *IEEE Transactions on Knowledge and Data Engineering* 12, 102–125.
- Bolliger, J., Gross, Th., 1999. Bandwidth Modelling for Network-Aware Applications. In: *Proceedings of IEEE INFOCOM*, pp. 1300–1309.
- Carneiro da Cunha, E., Rust da Costa Carmo, L.F., Pirmez, L., 1999. A stream relationship monitor for adaptive multimedia document retrieval. In: *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM '99)*, pp. 2071–2075.
- Chang, S.-K., 1999. Perspectives in multimedia software engineering. In: *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pp. 74–78.
- Cruz, I.F., Mahalley, P.S., 1999. Temporal synchronization in multimedia presentations. In: *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pp. 851–856.
- De Lima, R.M. et al., 1999. SAMM: An integrated Environment to Support Multimedia Synchronization of Pre-orchestrated Data. In: *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pp. 929–933.
- Huang, C.-M., Wang, C., 1998. Synchronization for interactive multimedia presentations. *IEEE Multimedia*, 44–62.
- Huang, C.-M., Lin, C.-H., Wang, C., 1998. An EFSM-based formal model for providing VCR-like functions in multimedia systems. In: *Proceedings of International Workshop on Multimedia Software Engineering*, pp. 10–17.
- Hwang, E., Prabhakaran, B., 2000. Protocol for collaborative multimedia presentations. In: *Proceedings of IEEE International Conference on Multimedia and Expo (ICME 2000)*, pp. 45–48.
- Jeng, M.D., Wen, Y.L., Chen, S.L., 1999. Synchronization models for pre-orchestrated interactive multimedia presentations using timed Petri nets. In: *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC '99)*, pp. 191–196.
- Jeong, T., Ham, J.W., Kim, S.J., 1997. A pre-scheduling mechanism for multimedia presentation synchronization. In: *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pp. 379–386.
- Lai, K., Baker, M., 1999. Measuring Bandwidth. In: *Proceedings of IEEE INFOCOM*, pp. 235–245.
- Liew, S.C., Lau, T., Lau, E., Fung, W., 1999. INTELLECT: a system for authoring, distributing, and presenting multimedia contents over the Internet. In: *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pp. 62–66.
- Lin, I.H., Wu, C.-C., Lee, B.-H., 1998. A synchronization model or multimedia presentation with critical overlap avoidance. In: *Proceedings of International Workshop on Multimedia Software Engineering*, pp. 36–43.
- Little, T.D.C., Ghafoor, A., 1989. Synchronization and storage models for multimedia objects. *IEEE Journal of Selected Area in Communications* 8, 413–427.
- Moreno, R., Mayer, R.E., 1999. Deriving instructional design principles from multimedia presentations with animations. In: *Proceed-*

- ings of IEEE International Conference on Multimedia Computing and Systems, pp. 720–725.
- Palacharla, S., Karmouch, A., Mahmoud, S.A., 1997. Design and implementation of a real-time multimedia presentation system using RTP. In: Proceedings of the Twenty-First Annual International Computer Software and Applications Conference (COMP-SAC '97), pp. 376–381.
- Paxson, V., 1999. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking* 7, 277–292.
- Prabhakaran, B., Raghavan, S.V., 1993. Synchronization models for multimedia presentation with user participation. In: Proceedings of ACM Multimedia, pp. 157–166.
- Qazi, N.U., Woo, M., Ghafoor, A., 1993. A Synchronization and Communication Model for Distributed Multimedia Objects. In: Proceedings of ACM Multimedia, pp. 147–155.
- Shih, T.K., Davis, R.E., 1997. IMMPS: a multimedia presentation design system. *IEEE Multimedia* 4, 67–78.
- Song, Y., Mielke, M., Zhang, A., 1999. NetMedia: Synchronized Streaming of Multimedia Presentations in Distributed Environments. In: Proceedings of IEEE International Conference on Multimedia Computing and Systems, pp. 585–590.
- W3C Recommendation, 1998. Synchronized Multimedia Integration Language (SMIL) 1.0 Specification, <http://www.w3c.org/TR/REC-smil>.
- W3C Recommendation, 2001. Synchronized Multimedia Integration Language (SMIL 2.0) Specification, <http://www.w3.org/TR/smil20>.
- Yang, C.C., Huang, J.H., 1996. A multimedia synchronization model and its implementation in transport protocols. *IEEE Journal of Selected Area in Communications* 14, 212–225.
- Yoon, K., Berra, P.B., 1998. TOCPN: interactive temporal model for interactive multimedia documents. In: Proceedings of International Workshop on Multi-Media Database Management Systems, pp. 136–144.