# User-Interaction Supported Data-Retrieving Engine for Distributed Multimedia Presentations

Chun-Chuan Yang
Multimedia and Communications Laboratory
Department of Computer Science and Information Engineering
National Chi-Nan University, Taiwan, R.O.C.
ccyang@csie.ncnu.edu.tw

***Abstract*— Distributed multimedia presentation enables users to view a multimedia presentation in the distributed manner, in which the objects of the presentation are located in remote sites. User-interaction provides the viewer with convenient viewing styles like fast forward and fast backward for a presentation, but it also introduces complexity to the data-retrieving engine. A smart data-retrieving engine that supports user interactions for the SMIL-based multimedia presentation is proposed in the paper. The SMIL script of the presentation is first converted to Real-Time Synchronization Model (RTSM) in order to provide a systematic view of the synchronization relationship. The algorithm for determining the proper objects to be retrieved as well as the pre-fetch time of the object under user actions is proposed for the data-retrieving engine.**

***Index Terms*— Distributed Multimedia Presentation, SMIL, Synchronization Model, User-Interaction**

## I. INTRODUCTION

With the development of high-speed network and computer technologies, multimedia information system like *distributed multimedia presentation* [1-3] is no longer unreachable. Distributed multimedia presentation involves various multimedia objects with some temporal constraints, which is also called *synchronization relationship*. In order to compose a multimedia presentation, a mechanism (language) is required for the presentation author. The World Wide Web Consortium (W3C) developed the *Synchronization Multimedia Integration Language (SMIL)* [4-7] that allows the use of a text editor to write multimedia presentations with both spatial relationship and temporal relationship. With the efforts of W3C, SMIL is becoming the most popular language in authoring multimedia presentations. This paper is thus focus on the SMIL-based presentations.

Since objects in a multimedia presentation may reside in remote data servers as shown in Fig. 1, 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 [8, 9] such as *play*, *stop*, *pause/restart*, *fast forward*, *fast backward* and *sliding (by a slider)*. The quality of the presentation depends on the ability of the player in dealing with the network behavior and user interactions. When a presentation is ongoing, the *data-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

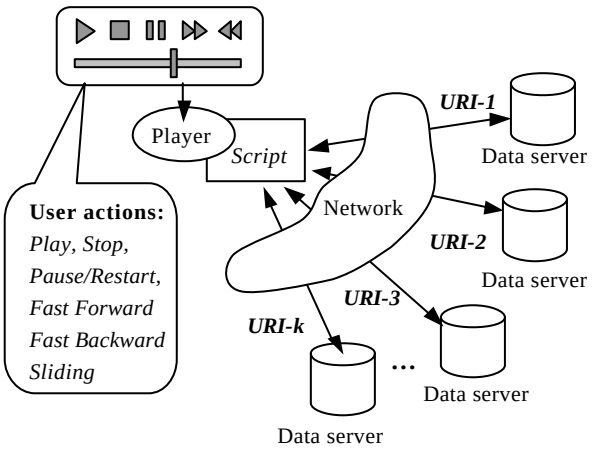



Fig. 1. Distributed multimedia presentation with user interactions

of the presentation. Therefore, the data-retrieving engine plays an important role on the quality of the presentation.

Two extreme policies for data retrieving could be adopted. First, retrieve all objects before starting the presentation, or second, never make the request to retrieve the object until the object's playback time. The first policy guarantees the smooth playback and deals with user interactions well in the cost of a long initial delay and a large buffer for all objects. Moreover, since the viewer could activate hyperlinks in an ongoing presentation as in SMIL, it is improper to pre-fetch all media objects in advance. Minimal buffer is required for the second 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 second policy and the random user interactions make the situation worse.

The proposed data-retrieving engine in the paper adopts a better policy that is called the *just-in-time policy*. The policy requires 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 data-retrieving engine only buffers necessary objects for the smooth progress of the presentation, thus it has a better buffer utilization and network bandwidth efficiency. For user interactions, the player provides users with VCR-like control functions, and passes the user action with corresponding parameters to the data-retrieving engine. By analyzing the synchronization

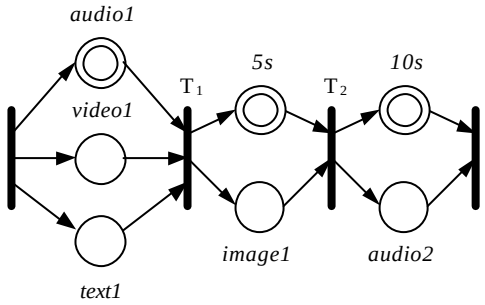



Fig. 2. An example of RTSM

relationship of the presentation and the user action, the data-retrieving engine determines the proper objects which should be played next and retrieve them just in time to meet the user action.

To deal with the synchronization relationship efficiently, a synchronization model [8-13], that provides a systematic view of the synchronization relationship specified in the SMIL script, is necessary for the data-retrieving engine. In the literature, *DEFSM (Dynamic Extended Finite State Machine)* [8] 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 presentation, and is hence complicated. A more compact and Petri-net based model namely *OCPN (Object Composition Petri Net)* was proposed to model the synchronization relationship among media objects [9-10]. However, we had shown that OCPN is not suitable for real-time network applications, and instead the *Real-Time Synchronization Model (RTSM)* was proposed [13]. Moreover, some synchronization behaviors that could be specified in SMIL make RTSM more suitable than OCPN. For example, SMIL allows the author to set the explicit beginning time, duration, or end time for a media object. RTSM could easily model the synchronization behavior [14], but OCPN could not. Therefore, RT SM is adopted in the proposed data-retrieving engine. We make a brief survey for RTSM in the following.

The elements in RTSM include *place*, *token*, and *transition* as in OCPN. 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 it will be immediately fired regardless the states of other places feeding the same transition. An example of RTSM is shown in Fig. 2 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. The RTSM in the figure requires that the audio segment *audio1*, the video clip *video1* and the text data *text1* be played simultaneously. Since *audio1* is an enforced place, transition *T1* is fired right after *audio1* is finished, regardless of whether *video1* or *text1* has finished or not. After firing *T1*, *image1* is displayed for 5 seconds then transition *T2* is fired. Finally, *audio2* is played for 10 seconds after *T2* fires. Note that the enforced place of "5s" in the figure is not a media object but a virtual medium that is called *Time Medium* [13]. The time medium is used to represent time duration.

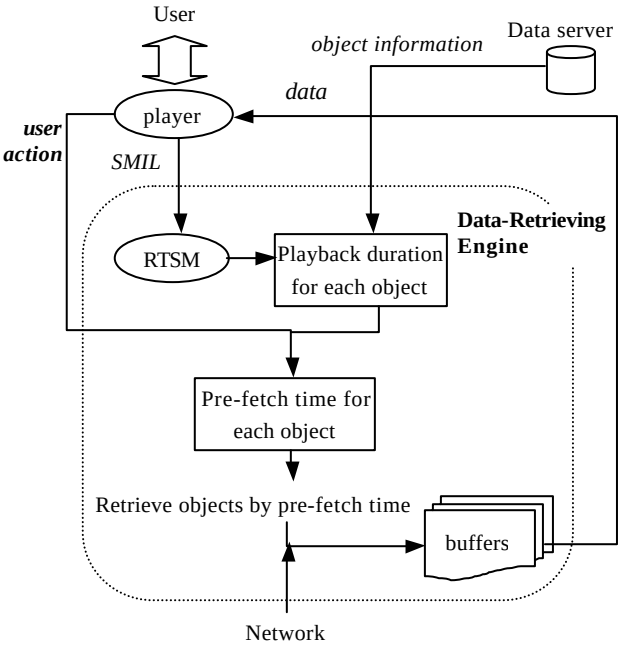



Fig. 3. Overview of the user-interaction supported data-retrieving engine

The remainder of the paper is organized as follows. The architecture of the proposed data-retrieving engine is explained in section II, in which the interface between the player and the data-retrieving engine for user interactions is also presented. In section III, a brief introduction for converting the SMIL temporal relationship to RTSM is presented. In section IV the method of determining the playback duration for each object under normal play mode is explained. The algorithm determining the pre-fetch time of each object for user actions is presented in section V. Finally, section VI concludes this paper.

## II. ARCHITECTURE OF THE DATA-RETRIEVING ENGINE

User-interaction supported data-retrieving engine is responsible for retrieving proper media objects and dealing with user actions for the presentation. The operation model of the proposed data-retrieving engine is shown in Fig. 3. The data-retrieving engine first accepts the SMIL script from the player, and converts the synchronization relationship to RTSM. By analyzing the RTSM model and by collecting object related information from data servers, the playback duration for each object under normal play mode is calculated. The data-retrieving engine then uses the obtained playback duration of each object as a base to handle the user actions and determines the pre-fetch time for objects. Proper objects are retrieved and delivered to the player by the data-retrieving engine according to the object's pre-fetch time.

User actions supported by the proposed data-retrieving engine include *play, stop, pause/restart, fast forward, fast backward, and sliding*. To define the user interactions more precisely, some parameters are required to be associated with the user actions as shown in Fig. 4. For example, in

**Play, Stop, Pause/Restart:** <*CurrentPoint*>

**Fast forward**:
<*CurrentPoint, JumpPeriod, PlaybackPeriod*>

**Fast backward**:
<*CurrentPoint, JumpPeriod, PlaybackPeriod*>

**Sliding**: <*NextPlaybackPoint*>

Fig. 4. User actions and corresponding parameters.

the fast forward mode, three parameters should be also passed to the data-retrieving engine: *CurrentPoint, JumpPeriod, and PlaybackPeriod*. *CurrentPoint* is used to indicate the time point within the overall presentation time at which the user action is made. *JumpPeriod* and *PlaybackPeriod* are used to define how does the player achieve the fast forward/backward operation. 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 apparently becomes *(JumpPeriod + PlaybackPeriod) / PlaybackPeriod*. The playback patterns for fast forward/backward are shown in Fig. 5 in which the only difference between fast forward and fast backward is the reverse ongoing directions. As for the sliding action, the data-retrieving engine only needs to know the next playback point, which is denoted by *NextPlaybackPoint*, so that it could retrieve proper objects.

When the user action is made, the data-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 one media 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 data-retrieving engine only retrieves necessary part of the object instead of retrieving the whole object. In the following, we present each step of data-retrieving engine.

```
<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. 6. Sample SMIL Document

## III. CONVERSION OF SMIL TO RTSM

Converting SMIL to RTSM requires examining the synchronization elements in SMIL like <*par*>, <*seq*>, and media object elements with related attributes. The algorithm of the conversion was proposed in the previous work [14], so we only illustrate the conversion by the sample SMIL script shown in Fig. 6.

The SMIL script in Fig. 6 requires the player to play the audio object *URI-1*, the video object *URI-2* and text object *URI-3* synchronously since these three objects are contained in a <par> element. The value of the “*endsync*” attribute in the <par> element requires <par> to end with the end of the audio object *URI-1*. In other words, once the audio object *URI-1* finishes playing, the video object *URI-2* and the text object *URI-3* must also stop playing at the same time. After the <par> element, the player has to display the image object *URI-4* for 5 seconds, and then play the audio object *URI-5* for 10 seconds. It is easy to be aware that the synchronization relationship of the SMIL script is similar to that of the RTSM in Fig. 2.

The obtained RTSM for the sample SMIL document after the converting process is shown in Fig. 7. Note that there are some *virtual places* and *virtual enforced places* (denoted by dashed circle and dashed double circle respectively) in the figure. They are used to maintain the consistency of the model, since the arc (arrows in the figure) 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, and the virtual enforced place is an enforced place that maps to time medium with

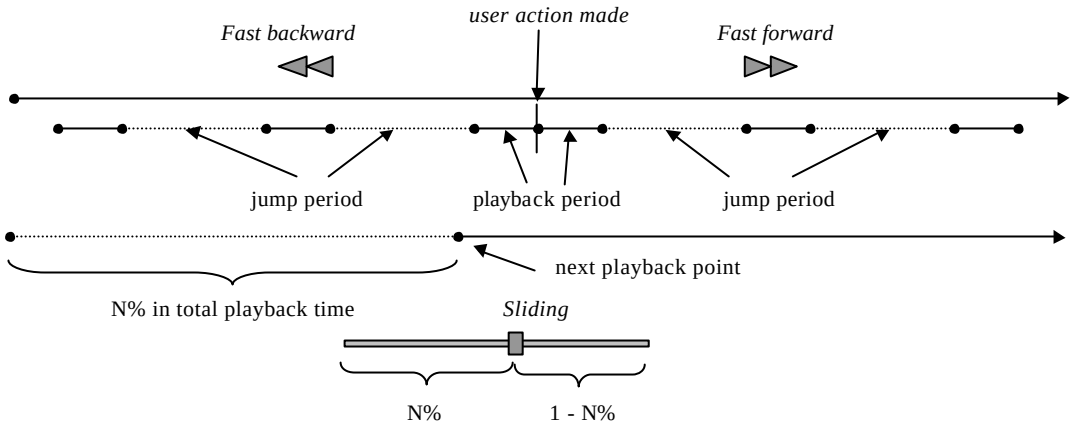



Fig. 5. Playback patterns for Fast forward/backward and Sliding operations.

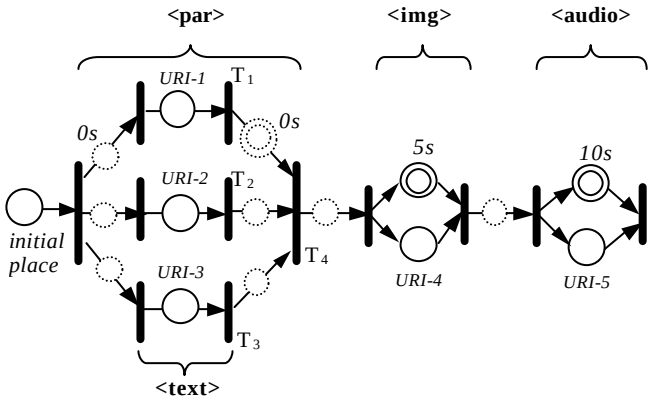



Fig. 7. RTSM for the sample SMIL document

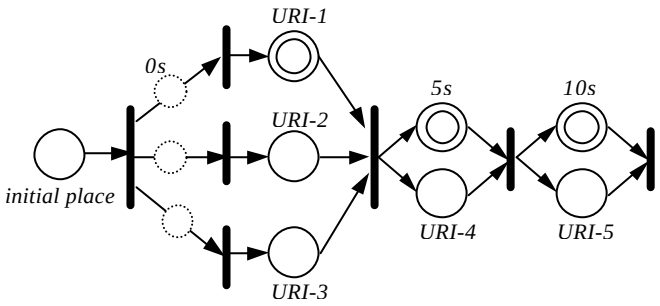



Fig. 8. Simplified RTSM for the sample SMIL document

zero duration. The virtual (enforced) places introduce dummy parts to the obtained RTSM. We apply three simple rules to simplify the model [14], and the simplified RTSM for the sample SMIL document are shown in Fig. 8.

## IV. COMPUTE THE PLAYBACK DURATION

In order to compute the pre-fetch time for each object, the data-retrieving engine first has to know the playback duration for each object under normal play mode (i.e. play speed = 1). The obtained playback duration is then used as a base to compute the pre-fetch time for objects under specific user action. 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 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 can not 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. 8 is shown in Fig. 9.

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.

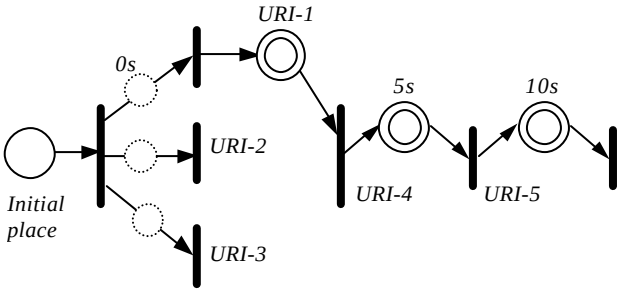
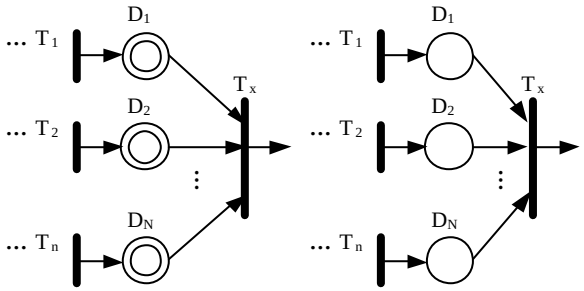



Fig. 9. Reduced RTSM for the sample SMIL document




(a) $T_x = Min(T_1+D_1, \ldots, T_n+D_n)$ (b) $T_x = Max(T_1+D_1, \ldots, T_n+D_n)$

Fig. 10. Determine the active time for transition $T_x$

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. 10-(a). The firing time of the transition for case (2) is instead the maximum value of its predecessors as illustrated in Fig. 10-(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 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 transitions. However, the enforced firing of a transition should also make all the preceding transitions fire simultaneously [13]. 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. 9 is 10 seconds, the firing time of each transition for the example is shown in Fig. 11, and the playback duration for each object in the example of Fig. 8 is shown in Fig. 12.

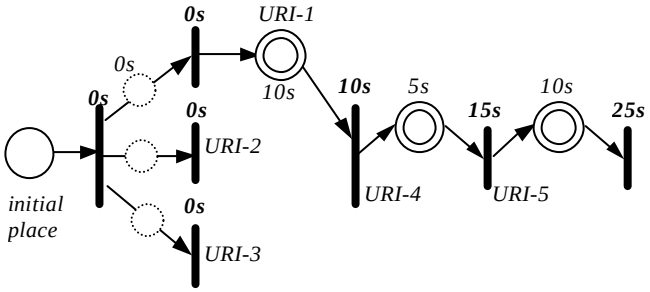



Fig. 11. The firing time for each transition for the example

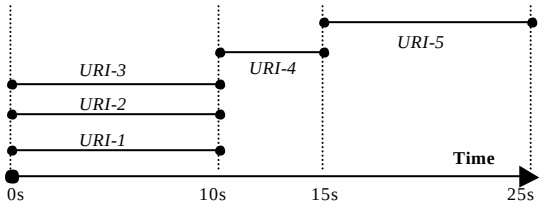



Fig 12. Object playback duration for the sample SMIL document

## V. DETERMINE THE PRE-FETCH TIME

In this section, the mechanism to determine the proper object (or part of an object) to retrieve as well as the object's pre-fetch time is presented. The objects that should be retrieved depend on the user action since different user actions result in different playback patterns and different playback time of objects. The pre-fetch time for an object depends on both the object's playback time under the user action and the network condition.

As mentioned in section IV, the data server provides the intrinsic duration for continuous objects to determine the playback duration for each object in the presentation under normal play mode. Besides, the data-retrieving engine also has to collect other information for each object from the data servers to determine the pre-fetch time. When the data-retrieving engine accepts the SMIL script, it sends probe packets to all the data servers for asking the object information, which includes *object size*, *estimated bandwidth*, and *play rate*. The data server has to estimate the bandwidth for the object to the data-retrieving engine by some bandwidth measuring mechanism [15, 16]. 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 $Size_{URI\text{-}i}$, the estimated bandwidth as $EstBW_{URI\text{-}i}$, and the play rate as $PlayRate_{URI\text{-}i}$.

In addition to object related information, the data-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 $RTDelay_{URI\text{-}i}$ 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 $(Size_{URI\text{-}i} / EstBW_{URI\text{-}i}) + RTDelay_{URI\text{-}i}$. The data-retrieving engine then

*By RTSM* *By Data Server*

| Obj ID | playback time | end time | size | EstBW | play rate | measured RTT |
|---|---|---|---|---|---|---|
| URI-1 | 5 sec | 10 sec | 40KB | 10KBps | 8KBps | 0.1 sec |
| URI-2 | … | … | … | … | … | … |
| … | | | | | | |

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

**play rate:** provided by the server, only valid for continuous objects

Fig. 13. Object information table for the sample SMIL document

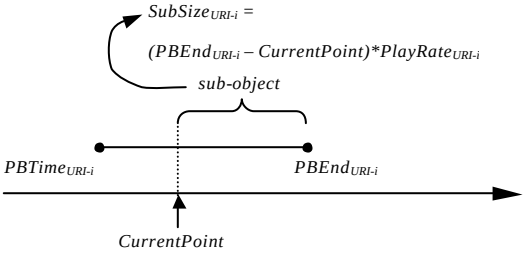



Fig. 14. Illustration for locating a sub-object.

fills in the object information table, which is shown in Fig. 13, as a reference for computing the pre-fetch time of each object.

### A. Play action

For each action made by the user, the player passes the action and associated parameters to the data-retrieving engine as presented in section II. *CurrentPoint* is the only parameter that is passed to the data-retrieving engine, and it is a time point within the total presentation time under normal play mode to indicate the time when the user action is made. Initially, the default action is the play action and *CurrentPoint* is the starting of the presentation (i.e. *0s*).

The playback duration in the object information table is used as a reference to identify the objects that should be played (retrieved) under the user action. Note that the values of the playback time (denoted by $PBTime_{URI\text{-}i}$) and end time (denoted by $PBEnd_{URI\text{-}i}$) for an object in the table are relative to the starting of the presentation. Therefore, objects with playback time later than *CurrentPoint* should be played under the play mode. The new playback time for object *URI-i* is $PBTime_{URI\text{-}i} - CurrentPoint$, which is denoted by $NewPBTime_{URI\text{-}i}$. In other words, the object should be played $NewPBTime_{URI\text{-}i}$ seconds later after the user action is made. The pre-fetch time for the object is computed as $NewPBTime_{URI\text{-}i} - (Size_{URI\text{-}i} / EstBW_{URI\text{-}i} + RTDelay_{URI\text{-}i})$.

Because of the random user actions, there are cases that *CurrentPoint* does not align to the $PBTime_{URI\text{-}i}$ of an object but within its playback duration. In such case, the new playback time for the object becomes *0s* (i.e. the action time) and the retrieving pattern depends on the types of the object. For static objects like images, since they are not temporally divisible, the data-retrieving engine should still retrieve the whole object. For continuous objects like audio,

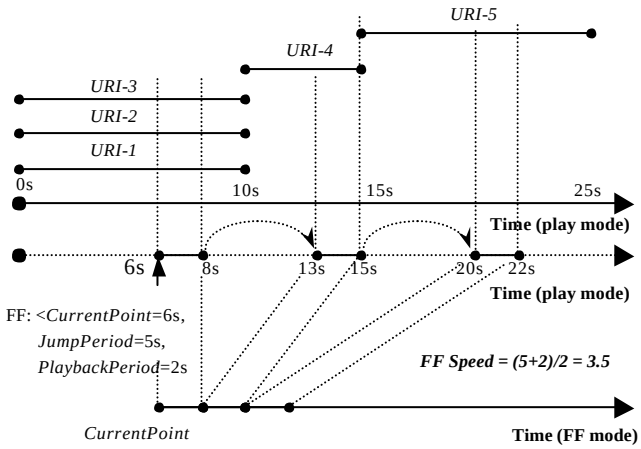



Fig. 15. E.g. Fast forward operation

only part of the object should be retrieved. The retrieved part of the object, which is called *sub-object*, relates to the new playback duration for the object, which is from *CurrentPoint* to $PBEnd_{URI\text{-}i}$. The size for the sub-object denoted by $SubSize_{URI\text{-}i}$ is computed as $(PBEnd_{URI\text{-}i} - CurrentPoint)*PlayRate_{URI\text{-}i}$. Thus, the retrieving time for the sub-part becomes $(SubSize_{URI\text{-}i} / EstBW_{URI\text{-}i} + RTDelay_{URI\text{-}i})$. Fig. 14 shows an illustration for the case of a continuous object. The pre-fetch time for the sub-object is then $NewPBTime_{URI\text{-}i} - (SubSize_{URI\text{-}i} / EstBW_{URI\text{-}i} + RTDelay_{URI\text{-}i})$.

### *B. Pause/Restart*

Once the user makes the pause action, the data-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 data-retrieving engine is thus the same as that of the play action.

### *C. Fast Forward*

Two parameters, *JumpPeriod* and *PlaybackPeriod*, define the forwarding pattern and *CurrentPoint* indicates the action time as mentioned in section II. In order to identify the objects or sub-objects that should be retrieved under this action, we have to scan the 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 playback duration in the information table. In each iteration, the part of the duration of an object, which *PlaybackPeriod* is located, is the sub-object to be retrieved and to be played.

We use the example in Fig. 15 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 at the 6th second in the presentation time, and *JumpPeriod* and *PlaybackPeriod* are 5 seconds and 2 seconds respectively. Therefore, the duration from the 6th to the 8th second is the first part of the presentation to be played, the 13th to the 15th second is the next part to

**Time base**: when the user action is made (e.g. 6s in Figure 14), i.e. *CurrentPoint*

**User action**: **fast forward** with *JumpPeriod*=5s and *PlaybackPeriod*=2s

*relative value to the time base*

| 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 | 2s | *NewPBTime – retrieval time* |
| URI-5 | 5s ~ 7s | 2s + 2s | 2s | *NewPBTime – retrieval time* |

Fig. 16. Pre-fetch timetable

be played, and then the 20th to the 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. 16. Note that in the pre-fetch table in Fig. 16, the sub-object fields for *URI-3* (text) and *URI-4* (image) are invalid since they are static objects.

The retrieving time for sub-object *URI-i* with duration *PlaybackPeriod* is calculated as $(PlaybackPeriod* PlayRate_{URI\text{-}i}) / EstBW_{URI\text{-}i} + RTDelay_{URI\text{-}i}$. The new playback time for each sub-object depends on the time of iteration in which the object is scanned, and it' s in the form of *PlaybackPeriod*IterationTime*. The per-fetch 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 data-retrieving engine makes requests to retrieve objects (or sub-objects) according to the pre-fetch time for objects. Note that when a new user action is made, the data-retrieving engine updates the pre-fetch timetable with the new computed pre-fetch time for each object.

### *D. Fast Backward*

The operation of the data-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. 5. Computation of the retrieving time and the pre-fetch time is the same as in the fast forward action. Note that the data-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.

### *E. 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 data-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 time point, so the operation of the data-retrieving engine is the same as that of the play action.

### *F. Discussion*

Since the value of the pre-fetch time is relative to the action time indicated by *CurrentPoint*, a negative value of the pre-fetch time implies that the object should be

retrieved before playing the presentation for the action just made. The method to deal with such case is to make the request to retrieve the object right after the user action is made.

The data-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 data-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 performance of the data-retrieving engine also depends on the accuracy of the estimated retrieving time. Since the network is dynamic, it is impossible to exactly estimate the time required to retrieve the object. However, if there is some way of booking to reserve the bandwidth in advance from the data server to the client, the estimated pre-fetch time will be more precise, and the quality of the presentation will also be improved. The simulation results in the previous work [14] show that the data-retrieving engine with the just-in-time policy could achieve better performance than the other policies.

## VI. CONCLUSION

User-interaction provides users with flexible viewing styles such as fast forward, fast backward and sliding operation for a multimedia presentation, but it also introduces more complexity to the data-retrieving process for distributed objects. In this paper, a data-retrieving engine is proposed for the SMIL-based distributed multimedia presentation with user interactions. The proposed data-retrieving engine adopts the just-in-time policy to efficiently make use of the data buffers and network bandwidth. The policy requires the data-retrieving engine to finish retrieving the object right before the object's playback time to provide the smooth progress of the presentation.

The data-retrieving engine first converts the SMIL script to the RTSM model (Real-Time Synchronization Model) for systematic analysis. To deal with random user interactions, the mechanism of determining the objects that should be retrieved under specific user action is proposed. By considering both the user action and the network condition, the pre-fetch time for objects could be determined, and the calculation of the pre-fetch time is presented in the paper. Although the data-retrieving engine has to re-compute the pre-fetch time of objects for each time a new user action is made, it is explained that the computation time of the algorithm is linear, and it could be finished within a short time.

## REFERENCES


[1] B. Prabhakaran and S. V. Raghavan, "Synchronization models for multimedia presentation with user participation," Proc. ACM Multimedia 1993, pp.157-166.

[2] M. Mielke and A. Zhang, "Optimally Ensured Interactive Service in Distributed Multimedia Presentation Systems," Proceedings, IEEE International Conference on Multimedia Computing and Systems, 1999, pp. 661–666.

[3] Y. Song, M. Mielke, and A. Zhang, "NetMedia: Synchronized Streaming of Multimedia Presentations in Distributed Environments," Proceedings of IEEE International Conference on Multimedia Computing and Systems, 1999, pp. 585-590.

[4] *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*, W3C Recommendation, June 1998, http://www.w3c.org/TR/REC-smil.

[5] *Synchronized Multimedia Integration Language (SMIL) Boston Specification*, W3C Working Draft 20-August-1999, http://www.w3.org/TR/smil-boston.

[6] Larry Bouthillier, "Synchronized Multimedia on the Web- A New W3C Format is All Smiles," Web Techniques Magazine, September 1998, Vol. 3 Issue 9.

[7] Philipp Hoschka, "An introduction to the Synchronized Multimedia Integration Language," IEEE Multimedia, Oct.-Dec. 1998, pp. 84-88.

[8] Chung-Ming Huang and Chian Wang, "Synchronization for interactive multimedia presentations," IEEE Multimedia, Oct.-Dec. 1998, pp. 44–62.

[9] K. Yoon and P. B. Berra, "TOCPN: interactive temporal model for interactive multimedia documents," Proceedings of International Workshop on Multi-Media Database Management Systems, 1998 pp. 136-144.

[10] T. D. C. Little, and A. Ghafoor, "Synchronization and storage models for multimedia objects," IEEE Journal of Selected Area in Communications, vol. 8, no. 3, pp. 413-427, Apr. 1989.

[11] N. U. Qazi, M. Woo, and A. Ghafoor, "A Synchronization and Communication Model for Distributed Multimedia Objects," Proc. ACM Multimedia 93, pp. 147-155.

[12] InHo Lin, Chwan-Chia Wu, and Bih-Hwang Lee, "A synchronization model for multimedia presentation with critical overlap avoidance," Proceedings of International Workshop on Multimedia Software Engineering, 1998, pp. 36-43.

[13] C. C. Yang and J. H. Huang, "A Multimedia Synchronization Model and its Implementation in Transport protocols," IEEE Journal of Selected Area in Communications, vol. 14, No. 1, pp. 212-225, Jan. 1996.

[14] C. C. Yang, "On the Design of the Data-Retrieving Engine for Distributed Multimedia Presentations," accepted by ICC 2001.

[15] J. Bolliger and Th. Gross, "Bandwidth Modelling for Network-Aware Applications," Proceedings., IEEE INFOCOM' 99, pp. 1300-1309, 1999.

[16] K. Lai and M. Baker, "Measuring Bandwidth," Proceedings., IEEE INFOCOM' 99, pp. 235-245, 1999.