Interactive Multimedia Presentation Management in Distributed Multimedia Systems

Ramazan Savas Aygun and Aidong Zhang
Dept. of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, NY 14260
{aygun,azhang}@cse.buffalo.edu

Abstract

Multimedia presentations are difficult to handle in existence of user interactions and complex synchronization requirements. In this paper, we present a synchronization model which can handle both time-based and event-based actions. The model can cope with the VCR-type user interactions. We also give a simple rule-based synchronization specification language which does not complicate as user interactions are allowed.

1. Introduction

There has been tremendous effort on the management of multimedia presentations for more than a decade. Applications such as video-on-demand, educational learning and tutoring, asynchronous distant and collaborative engineering need sophisticated models to store, access, query and present the multimedia data. These applications require a sophisticated synchronization model which can handle complex synchronization requirements. The model should be supplemented with a language which does not complicate as user interactions are allowed.

Composition of media objects play an important role in today's multimedia databases. A multimedia presentation must synchronize the media objects that participate in the presentation. The delay or loss of data over the network should not corrupt the integrity of the synchronization among individual media streams. In a multimedia database, the user queries may require to retrieve a multimedia presentation rather than a single stream. The users should be able to perform play, pause, resume, (fast-slow) forward, (fast-slow) backward and skip operations which provide great flexibility over the presentation. If the query presentation manager cannot support these functionalities in a consistent manner, the multimedia database will not be satisfiable to users.

We provide a model and a language which can handle both event-based and time-based actions having complex synchronization requirements with less complexity. The language contains the specification of media objects that participate in the presentation and the synchronization rules. The synchronization rules are based on the ECA (Event-Condition-Action) rules. The event expression, the condition statement and the actions that should be executed are parts of a rule. The synchronization model determines when these events are signaled, when conditions are met and when actions should be executed. The model is responsible for the integrity of the presentations after user interactions such as play, pause, resume, (fast or slow) forward, (fast or slow) backward and skip. These user interactions provide flexible access and scan on the results of a query. They do not complicate the specification of a presentation, since our model is an event-based model and the synchronization rules for skip and backward operations are deduced directly from the other rules. Synchronization rules separate events, conditions and actions. Our model does not have the disadvantages which the previous models have, such as limitation with either time-based or event-based actions, allowing only subset of the user interactions, limitation on the synchronization specification and application, and the complexity of the declaration if user interactions are supported.

In this paper, we will first describe our model in the next Section. We will discuss user interactions in Section 3. In Section 4, we will explain the language by giving examples. The last section concludes the paper.

2. The synchronization model for multimedia presentations

The synchronization model layouts media objects in a presentation. The synchronization model should have the following properties:

- It can compose media objects in various ways.
- It should support user interactions of VCR type like skip or backward which are helpful in browsing of the query results.
- It should have a simple way of specification of multimedia presentations. The user interactions usually complicate specifications.
- It should support both time-based and event-based operations.

The model has four major components, receivers, controllers, actors and the timeline. Events, conditions and actions are handled by these components.

2.1 Events, conditions and actions

When events are received, the corresponding conditions are checked. If a condition is satisfied, the corresponding actions are executed.

Event. There are 3 types of stream events, *start* event, *end* event and *realization* event. The *start* and *end* events are signaled when the first and last points (e.g. frames in a video stream) are processed by the stream, respectively. The *realization* event is signaled if a specific point of the stream is encountered. For example, this point is a frame number in a video stream. The realization event is important, because it enables to start new streams within a previously started stream. Each event has an event source, event data, event type and event destination. Event source can be the user or a stream. Composition of events may be required to trigger actions instead of a single event. Composite events can be created by boolean operators AND and OR:

$AND(e_1,e_2,...,e_n)$: All events(e_1 to e_n) should be signaled to trigger an action.

OR($e_1,e_2,...,e$): At least one of the events should be signaled to trigger an action.

For example, an image will be shown when the corresponding parts in both *audio AND video* data are realized. Composite events can also be generated by any combination of composite events.

Condition. A condition indicates the status of the presentation and its media objects. The most important condition is the direction of the presentation. The receipt of the events matter when the direction is forward or backward. Other types of conditions include the states of the media objects. For example, an audio stream may be turned off during fast-forward presentation and events that are expected by this stream may not be received. This kind of cases must be handled carefully.

Action. An action indicates what to execute when conditions are satisfied. *Starting* and *closing* a stream, and *displaying* or *hiding* images, slides and text are sample actions. There are two kinds of actions: *Immediate* Action and *Deferred* Action. *Immediate* action is an action that should be applied as soon as the conditions are satisfied. Deferred action is time-based and it can be started after enough time passing.

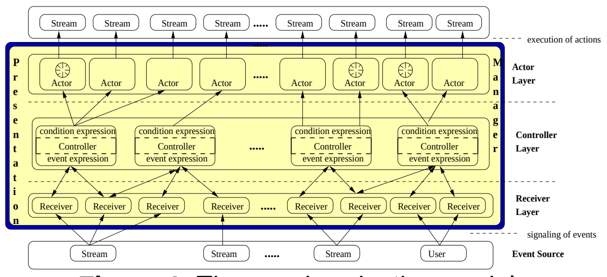

Figure 1. The synchronization model.

2.2 Receivers, controllers and actors

The synchronization model is composed of three layers, the receiver layer, the controller layer and the actor layer. Receivers are objects to receive events. Controllers check composite events and conditions about the presentation such as the direction. Actors execute the actions once their conditions are satisfied. The relationships between these elements are depicted in Figure 1.

Definition 1. A receiver is a triple $R=(e, t, C)$ where e is the event that will be received; t is the timer; and C is a set of controller objects.

Receiver R can question the event source through its event e. When e is signaled, receiver R will receive e. Receiver R also knows whether the event source can send the events, if not it takes necessary precautions. These precautions may assume that the event e will be signaled after some amount of time, or may ignore that e will be signaled. So, receiver R has a timer object. When receiver R receives event e, it sends information of the receipt of e to all its controllers in C. A receiver object is depicted in Figure 2. There is a receiver for each single event.

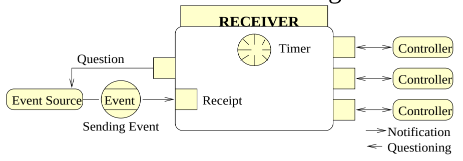

Figure 2. A receiver object.

Definition 2. A controller is a 4-tuple $C = (R, ee, ce, A)$ where R is a set of receivers; ee is an event expression; *ce* is a condition expression; and *A* is a set of actors.

Controller *C* has two components to verify, composite events *ee* and conditions *ce* about the presentation. When the controller *C* is notified, it first checks whether the event composition condition *ee* is satisfied by questioning the receiver of the event. Once the event composition condition *ee* is satisfied, it verifies the conditions *ce* about the states of media objects or the presentation. After the conditions *ce* are satisfied, the controller notifies its actors in *A*. A controller object is depicted in Figure 3.

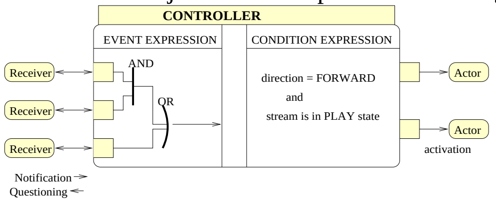

Figure 3. A controller object.

Definition 3. An actor is a pair *A = (a, t)* where *a* is an action that will be executed after time *t* passed.

Once actor *A* is informed, it checks whether it has some sleeping time *t* to wait for. If *t* is greater than 0, actor *A* sleeps for that amount and starts action *a*. If *t* is *0*, action *a* is an immediate action. If *t>0*, action *a* is a deferred action. An actor object is depicted in Figure 4.

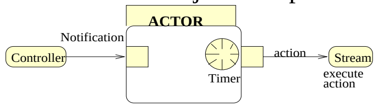

Figure 4. An actor object

2.3 Timeline

If skip and backward operations are allowed, alive actions, received or not-received events, sleeping actors and satisfied controllers should be known for any point in the presentation. The timeline object provides this kind of information. The existing work uses timeline to signal events at specific points or to layout streams.

Definition 4. A timeline object is a 4-tuple *T = (receiverT, controllerT, actorT, actionT)* where *receiverT*, *controllerT*, *actorT* and *actionT* are timelines for receivers, controllers, actors and actions, respectively. The timelines *receiverT*, *controllerT*, *actorT* and *actionT* keep the expected times of the receipt of events by receivers, the expected times of the satisfaction of the controllers, the expected times of the activation of the actors and the expected times of the start of the actions, respectively.

The information that is needed to create the timeline is the duration of streams and the relationships among the streams. The expected time for the receipt of *realization*, *start* and *end* events of streams only depend on duration of the stream and the start time of the action that starts the stream. Since the duration of a stream is already known, the start of the action depends on the activation of its actor. The activation of the actor depends on the satisfaction of the controller. The expected time when the controller will be satisfied depends on the expected time when its event composition condition is satisfied. The expected time for the satisfaction of an event composition condition is handled in the following way: In our model, events can be composed using AND and OR operators. Assume that ev_1 and ev_2 are two event expressions where $time(ev_1)$ and $time(ev_2)$ give the expected times of satisfaction of ev_1 and ev_2, respectively. Then, the expected time for composite events is found in the following way:

$$time(ev_1ANDev_2)=maximum(time(ev_1), time(ev_2))$$
$$time(ev_1ORev_2)=minimum(time(ev_1), time(ev_2))$$

where the *maximum* and *minimum* functions return the maximum and minimum of the two values, respectively.

3. User interactions

The low-level user interactions are interactions like VCR functions such as play, pause, resume, forward (fast or slow), backward (fast or slow) and skip. The high-level user interactions are based on the low-level operations. In our education database, each presentation has metadata in the form of table of contents which enables browsing of the presentation. If the user likes to watch a presentation, the presentation is supported with this table of contents. The table of contents contains the headlines, chapters, sections and subsections of the presentation.

The event-based models can handle play, pause, resume, speed-up and slow-down operations easier than the time-based models. The time-based models need to update the duration and start time of all the objects that participate in the presentation for the pre-specified operations. In the event-based models, these interactions only need to inform active streams. In our case, the time is connected to actors. When an actor is notified, it only needs to sleep for *(sleeping Time)/(|speed Of Presentation|)*. Speeding up or slowing down does not add any complexity to the presentation and only require the update of the speed of the presentation.

The user interactions like skip or changing direction (backwarding when playing forward or vice versa) need to be handled carefully. When skip-forward is performed, some events may be skipped which may cause ignorance of future streams which depend on the receipt of these events. The problem is solved by using the timeline of the presentation. In the timeline object, the expected time of the receipt of each event and the satisfaction of each controller is known. Therefore, it is known when events should have been received and when the controllers should have been satisfied by tracing the timeline. It is not always reasonable to start the actors whose controllers are satisfied, since their actions might have already finished. So, only the actions which will be active at the skip point are started from their corresponding points. The actors whose *sleep time* has not expired are allowed to sleep for the remaining time. If the direction of the presentation is modified, then receiver conditions, controllers and actors still need to be updated.

Presentations are usually specified in terms of constraints or conditions. Operations such as skip and backward increase the complexity of the specification since constraints should be specified for all the segments of the presentation. FLIPS [4] only provides event-based actions and does not enable backward. Nsync [1] does not enable backward and specification is complex since skip operation is allowed.

In our system, the event composition and other conditions for the backward presentation are automatically derived from the declaration of the rules of the forward presentation. So, the author does not have to consider the backward presentation or skipping, and this alleviates the declaration of the presentation substantially. The following logic is used for the generation of the backward presentation:

- **End-start relationship**. If the end of stream$_A$ participates in starting stream$_B$, when stream$_B$ reaches its beginning in the backward presentation, it will participate in backwarding stream$_{A.}$
- **End-end relationship.** If the end of stream$_A$ participates in ending stream$_B$, when stream$_A$ starts in the backward direction, it will participate in starting stream$_B$ in the backward direction.
- **Start-start(end) relationship.** If the start of stream$_A$ participates in starting (ending) stream$_B$, when stream$_A$ ends in the backward direction, it will participate in ending (starting) stream$_B$ in the backward direction.
- **Realization-start(end) relationship.** The realization happens when a segment of a stream is realized. Assume that the realization point is *P* and such points are monotonically increasing within a stream. If the realization of *P* in stream$_A$ participates in starting (ending) stream$_B$, then the realization of *(P-1)* will participate in ending stream$_B$ in the backward presentation.
- **Realization events in composite events.** Let stream$_A$.realization(P$_1$) and stream$_B$.realization(P$_2$) be realization events for streams stream$_A$ and stream$_B$. If (stream$_A$.realization(P$_1$) AND stream$_B$.realization(P$_2$)) cause some actions in the forward presentation, (stream$_A$.realization(P$_1$-1) OR stream$_B$.realization(P$_2$-1)) will cause actions in the backward presentation. Because the actions become active when both of the events are realized in the forward direction, the actions should be active as soon as one of the events are realized in the backward direction. If (stream$_A$.realization(P$_1$) OR stream$_B$.realization(P$_2$)) cause some actions in the forward presentation, (stream$_A$.realization(P$_1$-1) AND stream$_B$.realization(P$_2$-1)) will cause actions in the backward presentation.

4. The synchronization specification language

The specification of a presentation is mainly composed of two parts: the declaration of streams and the declaration of synchronization rules.

4.1 Declaration of streams

The declaration of a stream is composed of a source file name, an identifier to be used in rules, and start and end points in the source stream. Each stream declaration starts with its stream type, i.e. image, text, audio, slide, and video.

Example. <*video* src="examplesLecturer.mpg" *id*=LecturerVideo *StartPoint*=2000 *EndPoint*=60000> describes a video stream whose source is "examplesLecturer.mpg'' and identifier is LecturerVideo. Start and ending points are optional. Points indicate frame numbers for video streams.

4.2 Declaration of synchronization rules

Rules are ECA(event-condition-action)-type rules. A rule is composed of an event expression, a condition expression and a group of action expressions. Once the event expression is satisfied, the condition expression is checked. If the condition expression is true, then the actions are performed. A synchronization rule which has a single action has the following format:

```
<event eventExpression
        <condition conditionExpression
                <action actionExpression>>>
```

If a rule has more than a single action, actions are listed consecutively.

Start rule. The start rule has start as the event expression. This rule indicates what to perform when the presentation starts.

```
<event start <condition direction=forward
                <action ClassVideo.start>
                <action ClassAudio.start>
                <action ObjectSlides.display(1)>>>
```

The previous rule has the *direction=forward* as the condition expression. The rule has three actions. It indicates to start the video ClassVideo and the audio ClassAudio and to display the first slide of ObjectSlides. The condition expression is optional for any rule and if the condition is not specified, the default condition is *(direction=forward)*.

Complex synchronization rule. The events can be composed in an event expression to handle complex synchronization requirements. For example, the rule,

<*event* ((LecturerVideo.realization(N_1) and StuVideo.realization(N_2)) or ExAudio.realization(M))
    <*action* InheritanceText.display >>,

indicates the display of the InheritanceText when LecturerVideo and StuVideo realize the frame numbers N_1 and N_2, respectively, or ExAudio progresses M seconds. This kind of rules cannot be handled by PREMO [3]. PREMO enables only rules created by AND compositions. The event expression cannot be composite in [2].

Integration of time. The time can be associated with the actions. The rule below is about displaying slides of ObjectSlides. The following rule shows how to integrate time with actions.

<*event* ObjectSlides.realization(1)
    <*action* ObjectSlides.display(2) begin=20s>>,

The rule indicates to display 2nd slide 20 seconds after the display of the 1st slide. The following rules,

<*event* Audio.realization(M)
    <*action* Slides.display(4)>>

and

<event ClassAudio.start
    <*action* ObjectSlides.display(4) begin=M>>,

have different synchronization requirements. The first rule insists to display the slide after the audio stream has progressed M seconds. If the stream cannot be played properly, the display will be delayed. In the second one, the slide will be displayed *M* seconds after the start of the audio whether that audio is played properly or not. In SMIL [6], there is no way to distinguish these rules.

Pure time-based presentation. In fact, a presentation can be specified purely time-based depending on the user's start event. In that case, there will be only one rule in the presentation and will have the following format:

<*event* start
    <action ..... begin= .. >
    <action ..... begin= .. > ... >.

This rule shows how easy it is to declare a presentation depending on their start times. But this kind of rules is not enough to handle the integrity and consistency over the network because no constraint is specified among the streams.

Backward presentation. The rules for backward presentation are automatically generated. The *backward* event is used to determine which streams to start in backward presentation. The *backward* action prompts to start the stream in the backward direction.

In the following rule, when both ClassVideo and ClassAudio end, the ObjectVideo and the ObjectAudio are started.

<event (ClassVideo.end and ClassAudio.end)
        <*action* ObjectVideo.start>
        <*action* ObjectAudio.start>>

This rule has only *end-start* relationships. The actions will move to the event expression and the events will move to the action part. The streams will be started in the backward direction. The rule will be as follows:

<*event* (ObjectAudio.start AND ObjectVideo.start)
      <*condition* direction=backward
            <*action* ClassVideo.backward>
            <*action* ClassAudio.backward>>>

5. Conclusion

In this paper, we presented a multimedia presentation model and a language to satisfy complex synchronization requirements with less complexity. The model has the power of dealing with the event-based and time-based actions while enabling low-level user-interactions. The specification does not complicate as user interactions are allowed. We use this model in NetMedia[5] system.

References

[1] B. Bailey, J. Konstan, R. Cooley, and M. Dejong, "Nsync- A Toolkit For Building Interactive Multimedia Presentations, *Proc. of ACM Multimedia*, Minnesota, Minneapolis, September 1998, pp. 257-266

[2] R. Hamakawa and J. Rekimoto, "Object Composition and Playback Models for Handling Multimedia Data", *Multimedia Systems,* 1994, 2:26-35

[3] I. Herman, N. Corriera, D.A. Duce, D.J. Duke, G.J. Reynolds, and J.V. Loo, "A Standard Model for Multimedia Synchronization: Premo Synchronization Objects", *Multimedia systems,* 1998, 6(2):88-101.

[4] J. Schnepf, J. Konstan, and D. Du, "FLIPS: Flexible Interactive Presentation Synchronization", *IEEE Selected Areas of Communication,* 1996, 14(1):114-125.

[5] Y.Song, M. Mielke, and A. Zhang, "NetMedia: Synchronized Streaming of Multimedia Presentations in Distributed Environment", *IEEE Int. Conf. On Multimedia Comp. & Systems,* Italy, Florence, June 1999, pp. 585-590.

[6] The cwi smil page. http://www.cwi.nl/smil.