SynchRuler: A Rule-based Flexible Synchronization Model with Model Checking

Ramazan Savaş Aygün and Aidong Zhang

This research was funded by NSF grant IIS-9733730.

R. Aygun is with Computer Science Department, University of Alabama in Huntsville, Technology Hall N360, Huntsville, AL 35899. E-mail: raygun@cs.uah.edu

A. Zhang is with Department of Computer Science and Engineering, State University of New York at Buffalo, 201 Bell Hall, Box 602000, Buffalo, NY 14260-2000. E-mail: azhang@cse.buffalo.edu

Abstract

Flexible synchronization models cannot provide a proper way of managing user interactions that change the course of a presentation. In this paper, we present a flexible synchronization model, termed *SynchRuler*, which allows such user interactions including backward and skip. The synchronization rules, which are based on Event-Condition-Action (ECA) rules, are maintained to handle relationships among streams in *SynchRuler*. The synchronization rules are manipulated by Receiver-Controller-Actor (RCA) scheme where receivers, controllers, and actors are objects to receive events, to check conditions, and to execute actions, respectively. The verification of a multimedia presentation specification is performed with the synchronization model. The correctness of the model and the presentation is controlled with a technique called *model checking*. Model checker PROMELA/SPIN tool is used for automatic verification of the correctness of LTL (Linear Temporal Logic) formulas.

Index Terms

Multimedia Synchronization, Model Checking, Synchronization Rules, Multimedia Presentations

I. Introduction

Multimedia presentation management has drawn great attention in the last decade due to new emerging applications like video teleconferencing, collaborative engineering, asynchronous learning, and video-on-demand. There have been challenging problems confronted when multimedia presentations enable user interactions and are transmitted over networks shared by many users. The loss and delay of the data over the networks require a comprehensive specification of the synchronization requirements. The reduction of human effort in the automation of multimedia authoring is one of the challenging problems [24].

Multimedia presentation management research started with the organization of streams that participate in a presentation. VCR-based user interactions are incorporated at different levels at the later research. Flexible models do not enforce timing constraints and temporal organization is rather performed by relating events in the presentation. For example, *stream A starts after (meets) stream B.* Skip and backward interactions are able to be incorporated in time-based models. There are only few flexible models considering skip operation but with restrictions. Nevertheless, to our knowledge, there is no implemented flexible model that explicitly supports the backward functionality. The main difficulty behind backward and skip interactions is the change in the course of a presentation. Although conceptual (or graph-based) models have been proposed to handle backward and skip [22], the implemented systems could not use these models since they require the authors to spend enormous time on modeling due to enforcing the incorporations of the transitions for backward and skip. The author is challenged with the identification of the source and destination of transitions and the number of transitions.

When the traditional methods check the correctness of a specification, they ignore the capabilities and the interpretation of the synchronization model. To ensure that the synchronization model plays the presentation as expected is as important as the correctness of the specification. The synchronization models are likely to produce different multimedia presentations for the same specification. Hence, the synchronization model must be also incorporated in the verification.

A. Related Work

Allen [1] introduced 13 primitive temporal relationships for time intervals. Little et al. [18] extended these with n-ary and reverse relationships. The reverse relationships can declare relationships for backward presentations. The verification of a multimedia presentation usually consists of the verification of these relationships. One important factor in the modeling of presentations is whether the model is time-based or event-based. Timed Petri Nets are first introduced for multimedia presentations in OCPN [17] and extended with user interactions in [28]. The modeling of user interactions using Petri-Nets has been covered in [22]. The backward and skip interactions have been considered but a Petri-Net must be constructed for each possible skip and backward operation (including the current position of presentation and where the skip is performed). Gibbs [9] proposed a way of composing objects using their start times on a timeline. Hamakawa et al. [10] introduced an object composition and a playback model where the constraints can be defined only as pair-wise. A time-based synchronization model that considers master-slave streams having at least one master stream is explained in [15]. NSync [6] is a toolkit where the synchronization requirements are specified by synchronization expressions having syntax *When* $\{$*expression*$\}$ $\{$*action*$\}$. The synchronization *expression* semantically corresponds to "whenever the expression becomes true, invoke the corresponding *action*". NSync can only allow backward and skip with some limitations after user also specifies the operations for backward and each interval of skip. Time-based models usually keep the start time and duration of each stream.

In an event-based model, the start of a stream depends on an event signal. Events for multimedia applications are discussed in [27], and a model that includes temporal and spatial events is given in [21]. SSTS [21] is a combination of Firefly's [7] graph notation and transition rules of OCPN [17]. SSTS does not cover any user interactions. DAMSEL [23] has an event-based model that considers activation of two events such that "occurrence of an event will cause the occurrence of another event t time units later". Temporal constraints of Madeus [16] are based on Allen's temporal relations [1]. FLIPS [25] is an event-based model that has barriers and enablers to satisfy the synchronization requirements. It does not support fast-forward and fast-backward but it has a limited skip operation that moves to the beginning of another object. A timeline approach with event-based modeling that does not support fast-forward, fast-backward, and skip is proposed in [12]. In PREMO [11], synchronization points may be AND synchronization points to relate several events. The modeling of synchronization requirements using Event-Condition-Action (ECA) rules have been introduced in [2], [3], and [4], and forms the basis of ECA rule modeling in this paper. A hierarchical synchronization model that has events and constraints is given in [8]. SMIL [26] is a mark-up language for publishing synchronized multimedia presentations.

B. Our Approach

Most of the previous models are based on event-action relationships. The conditions of the presentation and participating streams also influence the actions to be executed. Thus event-condition-action (ECA) rules [19], which have been successfully employed in active database systems, are applied to multimedia presentations. Since these rules are used for synchronization, they are termed as *synchronization rules*. The synchronization model uses Receiver-Controller-Actor (RCA) scheme to execute the rules. In RCA scheme, receivers, controllers, and actors are objects to receive events, to check conditions, and to execute actions, respectively. The information for backward and skip operations are deduced by the model, and corrected by the author if necessary. To verify the correctness of the specification with the model, we use model checking technique, which automatically detects all the states that a model can enter and checks the truthness of well-formed formulas. PROMELA/SPIN [13], [14] tool has been used for model checking, which checks LTL (Linear Temporal Logic) formulas.

Contributions of our paper may be summarized as follows:

- developing a new interactive flexible synchronization model, termed *SynchRuler*,
- upgrading from event-action concepts to synchronization rules for multimedia presentations,
- incorporating backward and skip into *SynchRuler* without additional specification, and

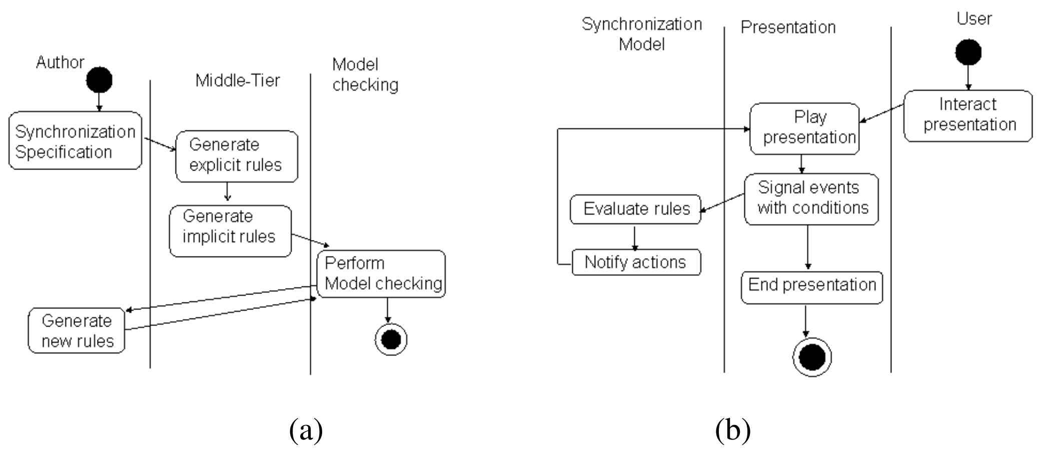

Fig. 1. (a) Authoring: rule generation and model checking, (b) multimedia presentation based on the synchronization model.

- verification of the specification with the synchronization model using model checking.

There are two phases to create a correct multimedia presentation (Fig. 1): 1) Authoring: correct rule generation and 2) multimedia presentation based on the synchronization rules. Our focus in this paper is on the authoring part. This paper is organized as follows. The following section explains the synchronization model. The user interactions are covered in Section III. The modeling and specification are explained in Section IV. Evaluation of rules is discussed in Section V. Our experiments are discussed in Section VI. The last section concludes the paper.

II. The Synchronization Model

In this section, we explain synchronization rules, elements of a multimedia presentation with SMIL, RCA scheme, and the presentation timeline object.

A. Synchronization Rules

Synchronization rules form the basis of the management of relationships among the multimedia elements. Each synchronization rule is based on an Event-Condition-Action (ECA) rule.

Definition 1: A *synchronization rule* is composed of an event expression, condition expression, and action expression. It can be formulated as: **on** *event expression* **if** *condition expression* **do** *action expression*.

A synchronization rule can be read as: When the *event expression* is satisfied if the *condition expression* is valid, then the actions in the *action expression* are executed. The *event expression* is the composition of events using boolean operators $\&\&$ (AND) and $||$ (OR). The AND composition requires signaling of all events in the composition but not necessarily at the same time. The *condition expression* is the composition of conditions using boolean operators AND and OR and determines the set of conditions to be validated when the event expression is satisfied. The action expression is the list of actions to be executed when the condition expression is satisfied.

Definition 2: An *event* is represented with $source(eventType[, eventData])$ where $source$ points the source of the event, $eventType$ represents the type of the event and $eventData$ contains information about the event.

In a multimedia system, the events may be triggered by a stream, user, or the system. Each stream is associated with events along with its data and knows when to signal events. The hierarchy of multimedia events is given in [3]. The user must specify information related to the stream events. Allen [1] specifies 13 temporal relationships. Relationships *meets, starts*, and *equals* require the *InitPoint* event for a stream. Relationships *finishes* and *equals* require the *EndPoint* event for a stream. Relationships *overlaps* and *during* require *Realization* event to start (end) another stream in the mid of a stream. The relationships *before* and *after* require temporal events since the gap between two streams can only be determined by time. Optional event data in the definition contains information like a realization point. Event type indicates whether the event is *InitPoint*, *EndPoint*, or *Realization* if it is a stream event.

Definition 3: A *condition* in a synchronization rule is represented with $(t_1 \; \theta \; t_2)$ where θ is a relation from the set $\{=, \neq, <, \leq, >, \geq\}$ and t_i is either a state variable that determines the state of a stream or presentation or a constant. The most important condition is whether the direction of the presentation is forward. The receipts of events matter when the direction is forward or backward.

Definition 4: An *action* is represented with $actionType(stream[, actionData], sleepingTime)$ where $actionType$ needs to be executed for $stream$ using $actionData$ as parameters after waiting for $sleepingTime$. *ActionData* is an optional parameter and used for fast-forward, fast-backward, and skip operations. *Starting* and *ending* streams are sample actions. *Backward* and *backend* actions are used to backward and to terminate a stream in the backward presentation.

B. Elements of a Multimedia Presentation and SMIL

The basic component of a multimedia presentation is a stream. In our model, a multimedia presentation may have a *container* consisting of streams or other containers. This allows grouping of streams and creation of subpresentations. Explicit rules are generated by middle-tier from a synchronization specification. In SMIL 1.0, there are two kinds of grouping: parallel and sequential. The extraction of synchronization rules from SMIL expressions is explained in [3].

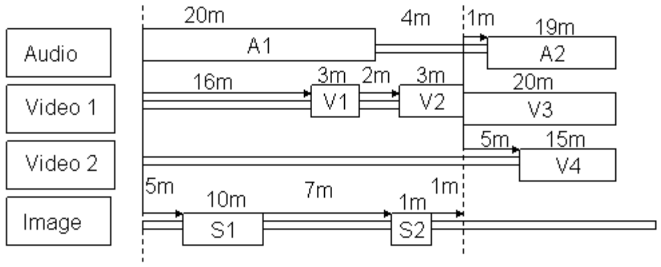

Fig. 2. Sample presentation.

Let us consider a simplified distance learning example that is depicted in Fig. 2. A multimedia lecture on shot transitions has two parts: teaching and discussion. In the first part, the instructor talks about different types of shot transitions in a video. The presentation starts with the audio (A1) of instructor associated with a slide (S1) describing different types of shot transitions. The instructor gives an example of two videos: original video (V1) and video with shot transitions (V2). V2 is associated with a fade-out slide (S2) that is shown at the time of transition. In the second part, there are two video channels: one for the instructor (V3) and another for the students (V4). The audio (A2) is common in the second part. The audio (A2) starts 1 minute after the instructor video since the beginning of the instructor video (V3) includes preparation for the discussion. The video for students (V4) is shown when students are involved in a discussion.

Assume that the presentation is grouped according to the SMIL expression given in Fig. 3. There are four containers in the presentation: the sequential presentation (SEQ1) of V1 and V2, the parallel presentation (PAR1) of A1, S1, S2, and SEQ1, the parallel presentation (PAR2) of A2, V3, and V4, and the sequential presentation (MAIN) of PAR1 and PAR2. We obtain the synchronization rules given in Fig. 4. The user event *USER(Start)* determines the beginning of a presentation. The event-action relationships for PAR1 container is depicted in Fig. 5.

```
<seq>
 <par>
     <audio id="A1" src="Instructor.au"/>
     <seq>
         <video id="V1" src="SampleVideo.mpg" begin ="16min"/ >
         <video id="V2" src="SampleTransitions.mpg" begin ="2min"/>
     </seq>
     <img id="S1" src="Transitions.gif" begin ="id(A1)(5min)" dur="10min"/>
     <img id="S2" src="FadeOut.gif" begin ="id(S1)(17min)" dur="1min"/>
 </par>
 <par>
     <video id="V3" src="Instructor.mpg" dur="20min"/>
     <video id="V4" src="Student.mpg" begin ="id(V3)(5min)" dur="15min"/>
     <audio id="A2" src="Classroom.au" begin ="id(V3)(1min)" dur="19min"/>
 </par>
</seq>
```

Fig. 3. The SMIL expression.

C. 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 of the presentation. Actors execute the actions once their conditions are satisfied.

Definition 5: A receiver is a pair $R = (e, C)$, where e is the event that will be received 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 receives e. When receiver R receives event e, it sends the information of the receipt of e to all its controllers in C. There is a receiver for each single event.

Definition 6: A controller is a 3-tuple $C = (ee, ce, A)$ where ee is an event expression; ce is a condition expression; and A is a set of actors. Controller C has two components to verify,

```
(F1)  on USER(Start)              if direction=FORWARD do start(MAIN)
(F2)  on MAIN(InitPoint)          if direction=FORWARD do start(PAR1)
(F3)  on PAR1(InitPoint)          if direction=FORWARD do start(A1)
                                                          start(SEQ1)
(F4)  on A1(InitPoint)            if direction=FORWARD do start(S1,5min)
(F5)  on SEQ1(InitPoint)          if direction=FORWARD do start(V1,16min)
(F6)  on S1(InitPoint)            if direction=FORWARD do start(S2,17min)
(F7)  on V1(EndPoint)             if direction=FORWARD do start(V2,2min)
(F8)  on V2(EndPoint)             if direction=FORWARD do end(SEQ1)
(F9)  on (SEQ1(EndPoint) && A1(EndPoint) && S1(EndPoint)
         && S2(EndPoint))         if direction=FORWARD do end(PAR1)
(F10) on PAR1(EndPoint)           if direction=FORWARD do start(PAR2)
(F11) on PAR2(InitPoint)          if direction=FORWARD do start(V3)
(F12) on V3(InitPoint)            if direction=FORWARD do start(A2,1min)
                                                          start(V4,5min)
(F13) on (V3(EndPoint) && A2(EndPoint)
         && V4(EndPoint))         if direction=FORWARD do end(PAR2)
(F14) on PAR2(EndPoint)           if direction=FORWARD do end(MAIN)
```

Fig. 4. Forward synchronization rules.

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 asking the receivers of the events. Once the event composition condition ee is satisfied, it verifies the conditions ce about the states of media objects or the presentation. After the condition expression ce is satisfied, the controller notifies its actors in A.

Definition 7: 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 t and then starts action a.

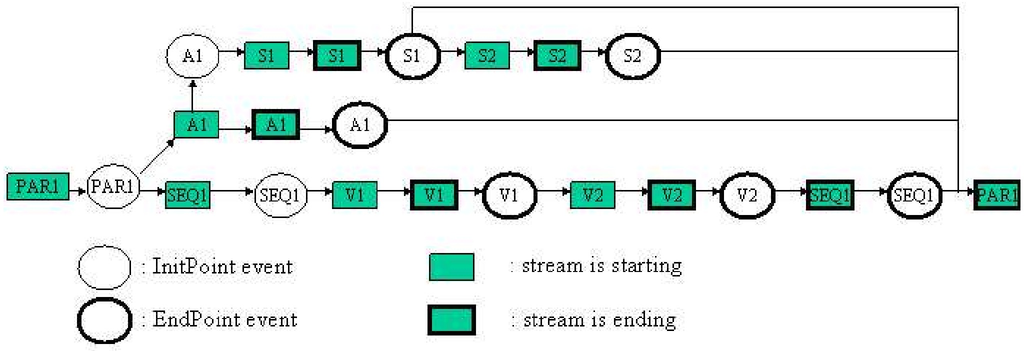

Fig. 5. The event action relationships for PAR1.

Receivers and controllers can be set or reset by the system anytime. Let F denote the direction condition $(direction = FORWARD)$. According to the synchronization rules given in Fig. 4, there are 19 receivers for events specified in the event expressions (Table I), 14 controllers for synchronization rules (Table II) and 16 actors for actions (Table III).

D. Timeline

Definition 8: A presentation timeline object is a 4-tuple $T = (receiverT, controllerT, actorT, actionT)$ where $receiverT$, $controllerT$, $actorT$, and $actionT$ are timelines to 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 expected time for finding the satisfaction of a controller is the expected time of the satisfaction of its event expression. The expected time for the satisfaction of an event composition condition is calculated using the composition type. 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 computed according to the predictive logic for WBT (will become true) in [6]:

$$time(ev_1 \ \&\& \ ev_2) = maximum(time(ev_1), time(ev_2))$$
$$time(ev_1 \ || \ ev_2) = minimum(time(ev_1), time(ev_2))$$

where $maximum$ and $minimum$ functions return the maximum and minimum of the two values, respectively. The timelines for receivers, controllers, and actors for synchronization rules given

R1	USER(Start)
R2	MAIN(InitPoint)
R3	PAR1(InitPoint)
R4	A1(InitPoint)
R5	SEQ1(InitPoint)
R6	S1(InitPoint)
R7	S1(EndPoint)
R8	V1(EndPoint)
R9	A1(EndPoint)
R10	S2(EndPoint)
R11	V2(EndPoint)
R12	SEQ1(EndPoint)
R13	PAR1(EndPoint)
R14	PAR2(InitPoint)
R15	V3(InitPoint)
R16	V3(EndPoint)
R17	A2(EndPoint)
R18	V4(EndPoint)
R19	PAR2(EndPoint)

TABLE I

RECEIVERS.

C1	R1 && F
C2	R2 && F
C3	R3 && F
C4	R4 && F
C5	R5 && F
C6	R6 && F
C7	R8 && F
C8	R11 && F
C9	R7 && R9&& R10&& R12&& F
C10	R13 && F
C11	R14 && F
C12	R15 && F
C13	R16 && R17 && R18 && F
C14	R19 && F

TABLE II

CONTROLLERS.

A1	start(MAIN)
A2	start(PAR1)
A3	start(A1)
A4	start(SEQ1)
A5	start(S1,5min)
A6	start(V1,16min)
A7	start(V2,2min)
A8	start(S2,17min)
A9	end(SEQ1)
A10	end(PAR1)
A11	start(PAR2)
A12	start(V3)
A13	start(A2,1min)
A14	start(V4,5min)
A15	end(PAR2)
A16	end(MAIN)

TABLE III

ACTORS.

in Fig. 3 are listed in Fig. 6. The receivers and controllers are ordered according to their expected satisfaction time. Only actors that have a sleeping time greater than 0 are displayed. The name of an actor shows its activation (sleeping time) and an underlined actor shows the end of its sleeping time. The actions are displayed in a similar fashion. The name of a container or a stream shows its starting time and if it is underlined, it shows its ending time. At a time instant, if a stream or a container has the same starting time as its container, the main container is shown in the timeline.

III. USER INTERACTIONS

The support of VCR functions such as play, pause, resume, forward (fast or slow), backward (fast or slow), and skip strengthens the browsing capabilities of multimedia presentations. Event-based models can handle play, pause, resume, speed-up, and slow-down operations easier than

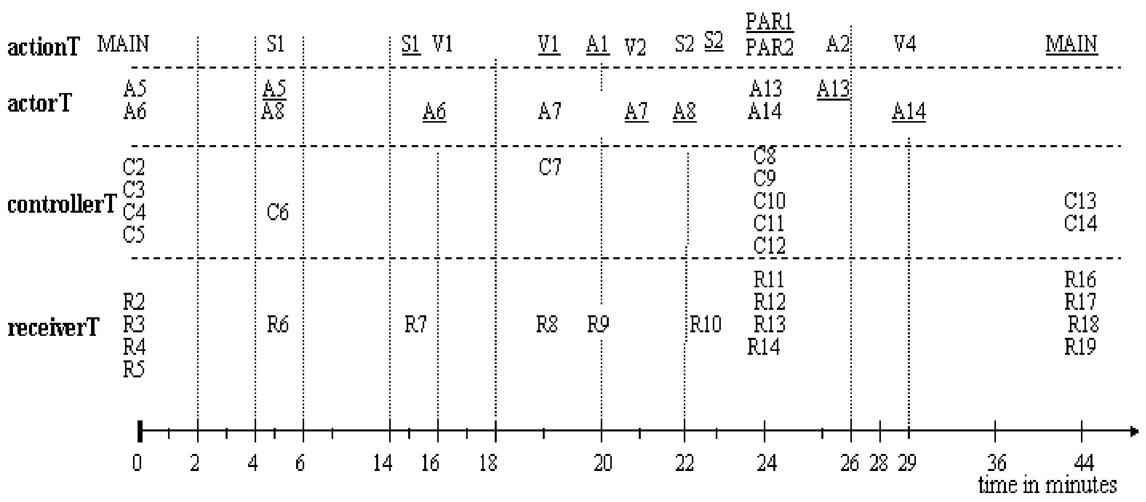
actionT
actorT
controllerT
receiverT
MAIN S1 S1 V1 V1 A1 V2 S2 S2 PAR1 PAR2 A2 V4 MAIN
A5 A6 A5 A8 A6 A7 A7 A8 A13 A14 A13 A14
C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14
R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R16 R17 R18 R19
0 2 4 6 14 16 18 20 22 24 26 28 29 36 44
time in minutes

Fig. 6. The presentation timeline.

time-based models, since these interactions only need to inform active streams. The speed of the presentation is 1 in nominal presentation. If the speed is greater than 1, it is a fast-forward operation. If the speed is less than 1 but greater than 0, it is a slow-forward operation. If the speed is negative, it is a backward operation. When an actor is notified, it only needs to sleep for $(sleepingTime)/(|speedOfPresentation|)$. Speeding up or slowing down only requires the update of the speed of the presentation.

The management of skip and backward operations is not straightforward in constraint-based or event-based models. NSync is one of the most comprehensive models that support VCR functions. However, the details of backwarding and skipping are not provided by NSync. Although it is not explicitly stated, the author must specify the synchronization expressions for the backward presentation. If backward expressions are not specified, nothing will happen when a stream reaches its beginning. Even when backward expressions are specified, the management of backwarding with skip operations cannot be handled properly. Let us consider the virtual mystery example from NSync where at time (value) 25 a dialog appears to choose a role, and the stream does not play after time $Mark$ until a role is chosen. The corresponding expressions are as follows:

```
When {I.value ≥ 25}                    { display role_dialog }
When {I.value < 25 }                   { undisplay role_dialog }
When {I.value ≥ Mark && role_chosen}   { I.speed= 1; undisplay role_dialog }
When {I.value ≥ Mark && !role_chosen}  { I.value = Mark; I.speed = 0;
                                       display role_dialog }
When {I.value = I.end}                 { role_chosen.speed = 1 }
```

Let $Mark = 40$, $I.value = 45$, and $I.end = 50$. Assume that the role is chosen and the user wants to skip to 32. The user cannot see the dialog if the role is already chosen (note that the first expression will only be satisfied when the expression is false and then becomes true; in this case there is no change of the truth of the expression). To see the dialog, the user must skip before 25 and then skip to 32 (to change the truth value of the expression). If the user skips to 20 from 45, the system will try to close an undisplayed dialog. To get rid of these issues, the author must provide a comprehensive specification so that the skip could be performed.

A. Skip Operation

In *SynchRuler*, the author of a presentation does not have to worry about skip operations. All the information for skip operations is extracted from the synchronization specification. The system is responsible for maintaining correct presentations after skips. A multimedia presentation has a lifetime. When skip-forward is performed, some events may be skipped that may cause ignorance of future streams that depend on the receipt of the skipped events. The problem can be solved by using the presentation timeline. It is not always reasonable to start the actors whose controllers are satisfied, since their actions might already have finished (to avoid restart of actions). So, only the actions that will be active at the skip point are started from their corresponding points. The actors whose *sleeping time*s have not expired are allowed to sleep for the remaining time. For example, assume that student is watching discussion session (PAR2) of the presentation and then first he decides to come to a point that he watched moments ago by backwarding the presentation and then decides to skip to the point when fade-out transition slide (S2) is displayed. Since the user initiated the backward presentation, the direction of the presentation is backward when skip is requested. This skip point corresponds to the 22^{nd} minute of the presentation. If a skip is requested to the 22^{nd} minute in the backward presentation, R10, R11, R12, R13, R14, R15, R16, R17, R18, and R19 are set and the rest of the receivers are reset. Controllers C8, C9, C10, C11, C12, C13, and C14 are set and the rest of the controllers are reset. Only containers MAIN and PAR1 and streams V2 and S2 will be active. The actor to backward A1 is a sleeping actor.

B. Backward Presentation

If the direction of the presentation is modified, then receiver conditions, controllers, and actors need to be updated. For example, if the direction is converted from forward to backward, the events that have been received are assumed to have not been received and the events that would have been received later should be set so that the earlier actions (in nominal presentation) can start again. 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, and this alleviates the specification of the presentation substantially. Authors usually specify the relationships among streams for some specific reasons. We call these reasons as *author properties*. Assume A and B are streams; C is a container; and ev_1 and ev_2 are events in a presentation. The *author properties* can be listed as follows:

Author Property 1: **Dependency** If A participates in starting B, B can be used for backwarding A. In this case, there is a dependency between A and B. If streams A and B are not overlapping, dependency property is used.

Author Property 2: **Master-Slave** If A influences B and causes B to start or end, the author considered A as a master stream over B. A should be master at this point in the backward presentation. If A and B are overlapping, master-slave property is used.

Author Property 3: **Hierarchy** If C starts its elements, the end of its elements participates in ending C in the backward presentation. A container ends when all elements are played.

Author Property 4: **Co-occurrence** If (ev_1 && ev_2) influences $action$, their co-occurrence is effective in the forward presentation. That is, the action will take place after both events are signaled. The action should be terminated when one of the events is signaled in the backward presentation. This corresponds to self-occurrence in the backward presentation.

Author Property 5: **Self-occurrence** If ($ev_1 \mid\mid ev_2$) influences $action$, their self-occurrence is effective in the forward presentation. That is, the action will take place after one of the events is signaled. The action should be terminated when both events are signaled in the backward presentation. This corresponds to co-occurrence in the backward presentation.

Author Property 6: **Realization** $A(Realization, P)$ corresponds to the realization event of P. P is an ascending number for a stream during forward presentation. In a video, it may correspond to when frame P is displayed. If $A(Realization, P)$ influences B in forward presentation, then realization of $P - 1$ is important for B in the backward presentation. If P represents time, $A(Realization, P)$ is used in the backward presentation.

The author property *realization* can be considered as an explicit declaration of master-slave property. The author properties are used to convert forward synchronization rules based on relations between events and actions in synchronization rules [4]. There is a conflict to backward stream A if stream A participates as a slave in a relationship and a stream B is dependent on stream A in another relationship. This conflict is resolved according to the order of precedence of the author properties: 1)Realization, 2) Master-Slave, 3) Dependency, 4) Hierarchy, and 5) Self-occurrence, Co-occurrence.

Master-slave rule is about simultaneous play of streams, and dependency is about in order presentation of streams. Since dependency property is about the sequential presentation of streams, it has a lower precedence than realization and master-slave properties. For example, assume that shot transitions slide (S1) should be terminated by instructor audio (A1) according to the master-slave property or by fade-out transition slide (S2) according to the dependency property. This conflict is resolved by using master-slave rule over dependency rule. Hierarchy property has lower precedence than master-slave and dependency properties. If a stream does not participate in any master-slave or dependency properties, the hierarchy property needs to be used. If the precedence of the properties is the same, one of the rules is chosen randomly to backward the stream.

The use of time for overlapping streams causes ambiguity when backward presentation is considered. For backward presentation, a time expression from the beginning of a presentation must be converted to a time expression from the end of a presentation. If two streams are overlapping, this ambiguity is resolved by using realization events. A more detailed explanation of time management is covered in [4].

Event	Dual Action
InitPoint	backward
EndPoint	backend

TABLE IV

DUAL ACTIONS FOR EVENTS.

Action	Dual Event
start	InitPoint
end	EndPoint

TABLE V

DUAL EVENTS FOR ACTIONS.

Action	Dual Action
start	backend
end	backward

TABLE VI

DUAL ACTIONS FOR ACTIONS.

```
(B1)   on  USER(Backward)           if  direction=BACKWARD do backward(MAIN)
(B2)   on   PAR1(InitPoint)         if  direction=BACKWARD do backend(MAIN)
(B3)   on  (SEQ1(InitPoint) && A1(InitPoint) && S1(InitPoint)
           && S2(InitPoint))        if  direction=BACKWARD do backend(PAR1)
(B4)   on  A1(Realization,5min)     if  direction=BACKWARD do backend(S1)
(B5)   on  V1(InitPoint)            if  direction=BACKWARD do backend(SEQ1,16min)
(B7)   on  V2(InitPoint)            if  direction=BACKWARD do backward(V1,2min)
(B8)   on  SEQ1(EndPoint)           if  direction=BACKWARD do backward(V2)
(B9)   on  PAR1(EndPoint)           if  direction=BACKWARD do backward(A1,4min)
                                                              backward(SEQ1)
                                                              backward(S1,9min)
                                                              backward(S2,1min)
(B10)  on   PAR2(InitPoint)         if  direction=BACKWARD do backward(PAR1)
(B11)  on  (V3(InitPoint) && A2(InitPoint)
           && V4(InitPoint)         if  direction=BACKWARD do backend(PAR2)
(B12a) on  V3(Realization,5min)     if  direction=BACKWARD do backend(V4)
(B12b) on  V3(Realization,1min)     if  direction=BACKWARD do backend(A2)
(B13)  on  PAR2(EndPoint)           if  direction=BACKWARD do backward(A2)
                                                              backward(V3)
                                                              backward(V4)
(B14)  on  MAIN(EndPoint)           if  direction=BACKWARD do backward(PAR2)
```

Fig. 7. Backward synchronization rules.

C. Synchronization Rules for Backward Presentation

In *SynchRuler*, all the backward rules are unique since the basis of rule generation is the forward rules and forward rules are not duplicates. The key factor in the number of rules is to start and to end streams. The relationships among streams are used to start and end streams in the backward presentation. However, if a stream does not interact with any other stream, that stream is manipulated by its container. If the start of a stream occurs by a combination of events, the backward rule is generated by using occurrence properties and order of precedence of author properties. It is guaranteed that there is a rule to backward every stream.

In our model, events have dual actions (Table IV), actions have dual events (Table V), actions have dual actions (Table VI), and conditions have dual conditions for the backward presentation. In addition, realization events have dual realization events. The duality is required to convert a forward synchronization rule into a backward synchronization rule. The dual actions for InitPoint and EndPoint is *backend* and *backward*, respectively. In the backward presentation, InitPoint and EndPoint events are signaled when *backend* and *backward* actions are performed, respectively. The dual events for *start* and *end* actions are InitPoint and EndPoint, respectively. The actions *start* and *end* have dual actions *backend* and *backward*, respectively. The dual event for $stream(Realization, P)$ is $stream(Realization, P-1)$ if P is specified in terms of stream components (e.g. frame) rather than in terms of time. The condition (direction=BACKWARD) is the dual condition for (direction=FORWARD).

Algorithm 1 Backward rule generation for simple rules

```
generateBackward(F,B)
IN: $F$ is a forward rule
OUT: $B$ is a backward rule
if $F$="on realization if condition do action" then
    $B$= "on dual(realization) if dual(condition) do dual(action)" // Realization
else if $F$="on master if condition do slave" then
    $B$="on master if dual(condition) do dual(slave)" // Master-Slave
else if $F$="on dependee if condition do dependent" then
    $B$="on dual(dependent) if dual(condition) do dual(dependee)" // Dependency or Hierarchy
end if
```

The algorithm to generate backward rules from forward rules is given in Algorithm 1. The *dual* function takes the dual of its input and returns event, condition, or action depending on where it exists in the synchronization rule. Fig. 7 depicts the backward synchronization rules that are generated for the synchronization rules given in Fig. 4. The synchronization rule F1 declares what to do when the user starts the presentation. The corresponding rule for the backward presentation, B1, determines what to do when the user backwards the presentation from the end. For USER(Start) event in F1, there is USER(Backward) event in B1. The action is backward(MAIN) in B1 for start(MAIN) action in F1.

The synchronization rule F2 has an InitPoint event and a start action. MAIN is the container of PAR1, and the backward rule is generated using the hierarchy rule. The dual event for *start* action is InitPoint. The event expression becomes PAR1(InitPoint). The dual action expression is backend(MAIN) for MAIN(InitPoint) event. All the condition expressions are *direction=BACKWARD*. The corresponding backward rule for F2 is B2. The synchronization rule F5 also contains a similar relationship but with time. Since V1 starts 16 minutes after the beginning of SEQ1 in F5, the time is included in the action expression of B4 as backend(SEQ1,16min). We do not provide more examples due to the space limitations.

IV. Modeling and Model Checking

The most common methods for verification of finite-state concurrent systems are simulation, testing, and deductive reasoning. It is not possible to consider all the cases in simulation and testing. If there is a severe problem in the model, it may even be costly for the system to verify by testing and simulation. The major advantage of model checking is that it is automatic and usually fast. The counter examples are produced by the model checking tools. We believe the verification of the specification must be performed with the synchronization model. Therefore, the author can know whether the specification is proper to play as expected with the synchronization model. We use PROMELA [13] as the specification language and SPIN [14] as the verification tool. These tools are publicly available, and Linear Temporal Logic (LTL) formulas can be verified. The representation of the synchronization model in PROMELA is explained in [5]. Model checking consists of three phases: modeling, specification, and verification.

A. Modeling

The streams and containers are the most important components to be modeled. In the modeling phase, the model should be kept simple and avoid unnecessary details. Therefore, we make some abstractions to ensure the correctness of the model. The abstraction is performed on the streams.

A container may enter four states. It is in *IdlePoint* state initially. Once started, a container is at *InitPoint* state in which it starts its containers and streams. After the *InitPoint* state, a container enters its *RunPoint* state and then enters enters *IdlePoint* state again. In the backward presentation, the reverse path is followed. A stream is similar to a container. If a stream must signal a realization event, a new state is added to *RunPoint* state per event [5].

The most difficult part in modeling is the modeling of time since PROMELA/SPIN does not support time explicitly. We use three types of time expressions for specification: *minimal time*, *precise time*, and *media time*. For example, the duration of V2 is 3 minutes. When V2 ends, the *minimal time* for the presentation of V2 is at least 3 minutes. We may also conclude that the total minimal time is 24 minutes since the beginning of the presentation. In our model, *precise time* is implemented by using guard conditions. For example, V4 cannot start until A2 starts. When A2 starts, the elapsed time since the beginning of V3 is 1 minute. The *minimal* and *precise* times do not give information about the progress of streams. A stream updates its *media time* as it starts, ends, and plays. When it starts, its media time is 0. When it ends, its media time is its duration. If V2 has no realization event as in Fig. 4, V2 has only one RunPoint state (one interval). The interval for RunPoint state is (0,3min). We get the time values by $getMinTime$ and $getMaxTime$ functions. For example, V2 has 3 RunPoint states for SMIL expression in Fig. 8: “after V2 starts but before S2 starts”, “after S2 starts but before S2 ends”, and “after S2 ends”. Three intervals for 3 RunPoint states are (0,1min), (1min,2min), and (2min, 3min).

B. Specification

Specification consists of the properties that a model should satisfy once the model enters some specific states. There are two basic properties that should be checked: *safety properties* and *liveness properties. Safety properties* assert that the system may not enter an undesired state. *Liveness properties* on the other hand assure that system executes as expected. Liveness includes the progress, fairness, reachability, and termination of a process [5]. Linear Temporal Logic formulas are properties of paths rather than properties of states. Therefore, an LTL formula

	Property Name	Properties	LTL Formulas
1	Active-Start	Stream A can be started if it is already active. (undesirable)	$\Diamond\ (S\ U\ P)$
2	Idle-Terminate	Stream A can be terminated if it is already idle. (undesirable)	$\Diamond\ (R\ U\ Q)$
3	Forward-Play	Stream A is played	$\Diamond\ (P\ \wedge\ \Diamond Q)$
4	Backward-Play	Stream A is played in the backward presentation	$\Diamond(Q\ \wedge\ \Diamond\ P)$

TABLE VII

STREAM PROPERTIES.

	Property Name	Properties	LTL Formulas
5	Allen-Before	Stream A is before stream B	$\Diamond(Q\ \wedge\ \Diamond\ K)$
6	Allen-Start	Stream A starts with stream B	$\Diamond(P\ \wedge\ K)$
7	Allen-End	Stream A ends with stream B	$\Diamond(Q\ \wedge\ L)$
8	Allen-Equal	Stream A is equal to stream B	$\Diamond(P\ \wedge\ K\ \wedge\ \Diamond\ (Q\ \wedge\ L))$
9	Allen-Meet	Stream B starts when stream A ends	$\Diamond(K\ \wedge\ Q)$
10	Allen-During	Stream B is during stream A	$\Diamond(P\ \wedge\ \Diamond(K\ \wedge\ \Diamond\ (L\ \wedge\ \Diamond Q))$
11	Allen-Overlap	Stream B overlaps stream A	$!(\Diamond(Q\ \wedge\ \Diamond K)\ \vee\ \Diamond\ (L\ \wedge\ \Diamond P))$

TABLE VIII

ALLEN'S TEMPORAL PROPERTIES.

is interpreted with respect to a fixed path. The operators $\Box, \Diamond$, and U correspond to *globally, eventually,* and *until*, respectively.

Let $P = streamA\_InitState$, $Q = streamA\_EndState$, $R = streamA\_IdleState$, $S = streamA\_RunState$, $K = streamB\_InitState$, $L = streamB\_EndState$, and $M = streamB\_IdleState$. The properties and corresponding LTL formulas for streams are presented in Table VII. Once it is ensured that streams play, further checks can be performed based on the relationships among streams. Based on Allen's temporal relationships, the properties and the corresponding LTL formulas are listed in Table VIII. The properties and the corresponding LTL formulas for the backward presentation are listed in Table IX.

In [20], some properties between two consecutive user interactions based on time are verified. For example, pause operation for a stream may be performed within t seconds after the start of the presentation where $0 < t < d$ and d is the duration of the stream. In a distributed system,

	Property Name	Properties	LTL Formulas
12	Backward-After	Stream A is after stream B	$\Diamond(K \wedge \Diamond Q))$
13	Backward-Backward	Stream A is backwarded with stream B	$\Diamond(Q \wedge L)$
14	Backward-End	Stream A ends with stream B	$\Diamond(P \wedge K)$
15	Backward-Equal	Stream A is equal to stream B	$\Diamond(Q \wedge L \wedge \Diamond (P \wedge K))$
16	Backward-Meet	Stream A is backwarded when stream B ends	$\Diamond(Q \wedge K)$
17	Backward-During	Stream B is during stream A	$\Diamond(Q \wedge \Diamond(L \wedge \Diamond (K \wedge \Diamond P))$
18	Backward-Overlap	Stream B overlaps stream A	$!(\Diamond(K \wedge \Diamond Q) \vee \Diamond (P \wedge \Diamond L))$

TABLE IX

BACKWARD TEMPORAL PROPERTIES.

	Property Name	Properties	LTL Formulas
19	Time-Before	Stream A is t seconds before stream B	$\Diamond(Q \wedge \Diamond(K \wedge timeCondition))$
20	Time-Start_Start	Stream B starts t seconds after stream A starts	$\Diamond(P \wedge \Diamond(K \wedge timeCondition))$
21	Time-End_Start	Stream B ends t seconds after stream A starts	$\Diamond(P \wedge \Diamond(L \wedge timeCondition))$
22	Time-End_End	Stream B ends t seconds after stream A ends	$\Diamond(Q \wedge \Diamond(L \wedge timeCondition))$
23	Time-After	Stream A is backwarded t seconds after stream B	$\Diamond(K \wedge \Diamond(Q \wedge timeCondition))$
24	Time-Backend	Stream A ends t seconds after stream B ends	$\Diamond(K \wedge \Diamond(P \wedge timeCondition))$
25	Time-Backward	Stream A is backwarded t seconds after stream B is backwarded	$\Diamond(L \wedge \Diamond(Q \wedge timeCondition))$
26	Time-Backward	Stream A ends t seconds after stream B is backwarded	$\Diamond(L \wedge \Diamond(P \wedge timeCondition))$

TABLE X

TIME PROPERTIES.

these constraints cannot be satisfied due to the possible delay of data. We only check time to verify the relationships among streams. For example, we would like to check V2 starts 2 minutes after V1 ends. The time when V1 ends and V2 starts are t_1 and t_2, respectively. We need to check $(time[t_2] - time[t_1] == 2)$ when V2 starts. This is added as $timeCondition$ in the formulas. The properties and corresponding LTL formulas for time relationships are given for both forward and backward presentation in Table X.

By using the previous properties, we can check the absolute temporal properties. However, checking absolute temporal properties does not guarantee that streams have made enough progress. For example, we may check whether the classroom audio (A2) starts 1 minute after

	Property Name	Properties	LTL Formulas
27	Progress-Start	Stream A starts at t seconds progress of stream B	$\Diamond(P \ \wedge pc(B,t))$
28	Progress-End	Stream A ends at t seconds progress of stream B	$\Diamond(Q \ \wedge pc(B,t))$
29	Progress-Backward	Stream A is backwarded at t seconds progress of stream B	$\Diamond(Q \ \wedge pc(B,t))$
30	Progress-Backend	Stream A ends in the backward direction at t seconds progress of stream B	$\Diamond(P \ \wedge pc(B,t))$

TABLE XI

PROGRESS PROPERTIES.

the instructor video (V3) starts but we cannot check whether V3 has made enough progress. Allen's temporal intervals can be checked by comparison of the beginning and the ending of streams. We need to use *media time* of streams to check the progress of streams. The progress condition is only satisfied when $(time == getMinTime(V2) \&\& time == getMaxTime(V2))$. Let $pc(B,t)$ denote that t minutes of stream B is played by using the progress condition. The properties and corresponding LTL formulas for progress relationships are given for both forward and backward presentation in Table XI.

For a multimedia presentation, the states of streams should be reachable in the backward presentation if and only if these states are reachable in the forward presentation. We call this property as *backward consistency* of a presentation, and term such a presentation as a *backward consistent* presentation. If we show the existence of a state that is not reachable in the forward (backward) presentation while it is reachable in the backward (forward) presentation, it is not backward consistent. It is not possible to specify a single LTL formula to check the backward consistency. We need to identify the states that are reachable in the forward presentation. So, the property is stated as two fold (Table XII). The number of states that need to be checked is $\lfloor m^n \rfloor$ where m is the number of states that a stream may enter and n is the number of streams.

V. EVALUATION OF RULES AND AUTHOR GUIDE

Our system provides forward and backward synchronization rules, and allows the author to perform verification tests based on LTL. There are two levels of managing rules: basic and advanced. If the original SMIL specification is correct, there is almost nothing to be done in basic rule management. Advanced rule management is about fine details of the synchronization rules. During our tests, we have used the following guide to reach a correct presentation.

	Property Name	Properties	LTL Formulas
31	Backward-Compatibility	If the *state* is not reachable in the forward presentation then	$\Box\, !state$
		It is possible to reach the *state* in the backward presentation	$\Diamond\, state$,
		else	
		It is not possible to reach the *state* in the backward presentation	$\Box\, !state$

TABLE XII

COMPATIBILITY PROPERTIES.

A. Basic Evaluation

There are two phases of evaluation of the rules. In the first phase, the validity of the forward presentation rules are checked.

1. Check forward synchronization rules first. It is an important mistake to start with checking the validity of the backward rules before checking the validity of forward rules since backward rules depend on the forward rules.

2. Check stream properties. Before checking temporal properties like "a stream starts before another stream", it is better to guarantee that the streams are actually played.

3. Check temporal interval properties. It depends on the author to check which properties satisfy the requirements of the specification. For example, if we want to see whether S2 is displayed with V2 or not, we use *Allen-During* during property and LTL formula.

4. Divide and conquer unsatisfied properties. If an LTL property is not satisfied, it is important to pinpoint the original source of the problem although SPIN provides an instance of a contradiction. For example, when we test *Allen-During* property, SPIN reports a presentation instance where fade-out transition slide (S2) starts and ends earlier than sample transitions video (V2). This property is not satisfied due to possible delay in V2.

5. Link the dependent streams. If a stream A is dependent on another stream B (i.e., its start or ending depends on another stream), the start of stream A should be initiated by stream B. For example, S2 is dependent on V2, so the start of S2 should depend on V2 rather than S1. By using Time-Start property, we can check whether S2 starts 1 minute after the beginning of V2 or ends 1 minute before the end of V2. The presentation in Fig. 4 does not satisfy that S2 starts 1 minute after V2 starts. The rule F6 is converted to "**on** V2(InitPoint) **if** direction=FORWARD

do start (S2,1min)”.

6. Check the progress of streams. The temporal properties may be equivalent to the progress properties in lossless presentations. For example, S2 should start 1 minute after V2 starts, but is 1 minute portion of V2 played when S2 starts? This can be checked with *Progress-Start* properties in Table XI.

7. Use realization events whenever possible instead of temporal relations. If two streams are not overlapping, realization events cannot be used. However, if two streams overlap, there is a higher possibility of using realization events. If the progress properties of a stream are not satisfied , then realization event can be used. For example, the rule F6 is further converted to ”**on** V2(Realization,1min) **if** direction=FORWARD **do** start (S2)”. The best way to solve the synchronized ending of S2 with V2 is also to use a realization event (V2(Realization,2min)).

8. Regenerate SMIL expression. Most of the synchronization rules can be expressed by using SMIL. It is important to have the SMIL specification as correct as possible. Our system has the support to convert master-slave properties by realization events. The author has to decide whether his updates could be specified with SMIL expression or not. If his updates could be specified with SMIL, it is better if the author updates the original SMIL expression. Fig. 8 depicts the updated SMIL expression.

8. Regenerate the rules from SMIL. In this case, the forward synchronization rules are more consistent. If there is any missing correction, the corresponding rule is updated.

9. Check the backward synchronization rules. At this level, we apply similar steps as in forward presentation: check stream and temporal properties, divide and conquer unsatisfied properties, link dependents of streams, and check the progress of streams.

B. Advanced Evaluation

In most cases, the basic evaluation of the rules is enough for SMIL-based presentations. Even though the forward synchronization and backward synchronization rules are consistent, the author may like to update, insert, or delete rules.

1) Rule Updates: In the previous section, we have explained examples of updating synchronization rules (like linking dependent streams). An update on a synchronization rule may require the update of the event expression and action expression. The possible issues with event expression are unnecessary events, missing events, and composition of events. If the SMIL

```
<seq>
 <par endsync="last">
  <seq>
   <audio id="A1" src="Instructor.au"/>
   <video id="V2" src="SampleTransitions.mpg" begin ="id(A1)2min"/>
  </seq>
  <video id="V1" src="SampleVideo.mpg" begin ="id(A1)16min" end="id(A1)19min"/>
  <img id="S1" src="Transitions.gif" begin ="id(A1)(5min)" end="id(A1)15min"
                                           dur="10min"/>
  <img id="S2" src="FadeOut.gif" begin ="id(V2)(1min)" end="id(V2)(2min)"
                                           dur="1min"/>
 </par>
 <par>
  <video id="V3" src="Instructor.mpg" dur="20min"/>
  <video id="V4" src="Student.mpg" begin="id(V3)(5min)" dur="15min"/>
  <audio id="A2" src="Classroom.au" begin="id(V3)(1min)" dur="19min"/>
 </par>
</seq>
```

Fig. 8. The new SMIL expression.

specification has been done properly, the problem of unnecessary and missing events will not exist. Even though the composition of events are correct, the author may still need to change the composition of rules either to introduce more *flexibility* or to make the rule more *strict*. A *flexible rule* has a more chance to be triggered than a *strict rule*. A *strict rule* can be converted into a *flexible rule* by 1) removing events from AND compositions, 2) inserting events into OR compositions, and 3) replacing AND compositions with OR compositions. In a similar fashion, a *flexible rule* can be converted into a *strict rule* by 1) inserting events into AND compositions, 2) removing events from OR compositions, and 3) replacing OR compositions with AND compositions. Thre are 8 possible ways of composing 3 events (V3(InitPoint), V4(InitPoint), and

A2(InitPoint)) of rule B11. Our system chooses AND composition (strict rule) by default. The author may convert this *strict rule* to a *flexible rule* by using OR Compositions. For example, it can be converted as **on** (V3(InitPoint) || A2(InitPoint) || V4(InitPoint)) **if** direction=BACKWARD **do** backend(PAR2)".

In the event expression of Rule B11, the parallel container ascertains that all of its elements finish before its end. However, Rule B12a and B12b ensures the termination of V4 and A2 by a realization event. If the user checks backward LTL formulas for V3, V4, and A2, it is noted that V4 and A2 is terminated earlier. Rule B11 can be made flexible by ignoring V4(InitPoint) and A2(InitPoint) from the event expression as "**on** V3(InitPoint) **if** direction=BACKWARD **do** backend(PAR2)". In this case, it is still equivalent to the original expression.

2) Rule Deletion: Especially, there might be streams that are terminated by other streams to support backward compatibility. If the author is not interested in backward compatibility at all levels, he may remove some of the rules. For example, rules B12a and B12b terminate streams V4 and A3, respectively. If the user thinks that terminating V4 and A2 by realization event is strict and deletes those rules, he allows V4 and A2 to be played even if there is a delay.

3) Rule Insertion: The rules are generated for starting and ending streams. For the backward rules, the streams are only terminated (backend) if there are master-slave and realization properties. In other cases, the streams are allowed for their presentation. For example, there is no rule to terminate (backend) V2. If the author really requires the termination of V2 in the backward presentation, he could use the following rule: "**on** S2(InitPoint) **if** direction=BACKWARD **do** backend(V2,1min)".

VI. EXPERIMENTS

In our system, the author must specify a multimedia specification in SMIL or using synchronization rules. In our environment, only SMIL components that are used by the system are supported. The rest of the properties of the SMIL 1.0 and SMIL 2.0 can be added later. We have implemented a program that takes SMIL expressions and generate the forward synchronization rules. Once the forward synchronization rules are extracted, backward rules are generated. Our real examples are usually composed of audio/video associated with slides. The NetMedia [29] system is used as a testbed for our synchronization model. Besides being a synchronization model, *SynchRuler* synchronizes the author with the synchronization model. *SynchRuler* has helped us reduce the number of incorrect specifications while increasing the number of correct presentations. The author is able to provide a desirable and controlled flexibility when multimedia presentations are created.

The complexity of model checking especially depends on the type of containers and LTL property that is verified. There are three factors that affect the performance of the verification: the number of streams, the structure of the presentation, and the property to be verified. The structure of a presentation is related with the sequential and parallel presentation of streams. For example, sequential presentation with 3 streams corresponds to playing 3 streams back to back, and a parallel presentation with 3 streams corresponds to playing 3 streams in parallel.

We first investigated the complexity of the number of streams in parallel and sequential presentations (Table XIII). Only safety properties are checked for these presentations. In Table XIII, there is no increase in terms of time with respect to the number of streams in sequential presentations. We also tested sequential presentation with 100 streams. The time elapsed to verify safety properties for 100 sequential streams is 0.12 seconds. This shows that the verification can be completed within in a tenths of a second for sequential presentations. The depth, states, and transitions increase linearly with the number of streams for sequential presentations. When a new stream is added into the sequential presentation, there are phases where the new stream starts, plays, ends, and becomes idle. The complexity of the running time and the number of states and transitions is $O(mn)$ where n is the number streams in the sequential presentation and m is the number of states a stream may enter. In these experiments, m is 4.

If the number of streams that are played in parallel is less than 7, the verification is completed within a second. In fact, this is equivalent to the limit on the number of tracks available for a presentation. In realistic examples, mostly the number of tracks is limited by 3. For a parallel presentation, there are more combinations of playing streams. The number of possible presentations for n streams that have m states is

$$I(n,m) = \frac{(nm)!}{(m!)^n}.$$

This explains the steep increase in running time, memory, states, transitions, and depth. Nevertheless, the running time is still within a second for 6 tracks. Although the number of tracks looks like 3 for the presentation in Fig. 2, the number of parallel streams that can be played by

PAR1 is 4. S1 and S2 can be played in parallel since they are components of a parallel container.

Presentation	# Streams	Depth	States	Transitions	Memory (Mbyte)	Time (in seconds)
single	1	4	5	5	1.5	0.03
sequential	2	12	13	13	1.5	0.03
sequential	3	20	21	21	1.5	0.03
sequential	4	28	29	29	1.5	0.03
sequential	5	36	37	37	1.5	0.03
sequential	6	44	45	45	1.5	0.03
sequential	7	52	53	53	1.5	0.03
sequential	8	60	61	61	1.5	0.03
sequential	9	68	69	69	1.5	0.03
parallel	2	16	51	93	1.5	0.03
parallel	3	21	151	388	1.5	0.03
parallel	4	26	539	1835	1.5	0.06
parallel	5	31	2079	8754	1.7	0.18
parallel	6	36	8227	41017	2.4	0.71
parallel	7	41	32807	188480	6.0	3.96
parallel	8	46	131115	852039	20.8	16.50
parallel	9	51	524335	$3.8*10^6$	2.8	84.80

TABLE XIII

EXPERIMENTS BASED ON THE STRUCTURE OF PRESENTATIONS.

We also checked the time elapsed for the verification of LTL properties for sample presentations. Our goal is to identify the cost of verification of LTL properties. Table XIV lists the elapsed time for a sample presentation (Fig. 8). All verification properties except *during* and *overlap* are completed within a second. Actually, the time for overlap is far more than we expected. It depends on the complexity of the LTL formulas specified for the overlap property. If the overlap property is divided into 2 parts (similar to the before property), it can be verified within a second.

Except for the overlap property, the verification time for safety properties gives an upper bound on the verification of the properties. These results consider the non-progress cycles and other possible errors. Verification of properties takes less time than verification of safety properties since only a specific condition is checked in the model. The verification of a property can be completed without checking all states if the property is satisfied in early stages of the verification.

	Time (in seconds)
Play Stream	0.03
Before	0.16
Start	0.06
End	0.06
Equal	0.01
Meet	0.65
During	1.14
Overlap	21.78
Time Condition	0.11
Progress Condition	0.07

TABLE XIV

VERIFICATION OF PROPERTIES FOR A SAMPLE PRESENTATION.

VII. CONCLUSION AND FUTURE WORK

Previous verification techniques focused on the correctness on the specification and did not question the synchronization model. We believe our work will pioneer in the evaluation of a specification with the synchronization model for multimedia systems. Our work promotes the standardization of properties for multimedia presentations. Model checking technique allows to check the correctness of a specification for a synchronization model. In the future, multimedia presentation systems will be able to answer the question whether presentations can be played properly in a specified environment. This helps authors choose another system or improve their specification. The major limitation of SPIN is not being able to support time explicitly. The properties and LTL formulas to be verified may need to be updated according to the application.

REFERENCES

[1] J. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of ACM*, 26(11):823–843, November 1983.

[2] R. Aygun and A. Zhang. Interactive multimedia presentation management in distributed multimedia systems. In *Proc. of Int.Conf. on Information Technology: Coding and Computing*, pages 275–279, Las Vegas, Nevada, April 2001.

[3] R. Aygun and A. Zhang. Middle-tier for multimedia synchronization. In *2001 ACM Multimedia Conference*, pages 471–474, Ottawa, Canada, October 2001.

[4] R. Aygun and A. Zhang. Management of backward-skip interactions using synchronization rules. In *The 6th World Conference on Integrated Design & Process Technology*, Pasadena, California, June 2002.

[5] R. Aygun and A. Zhang. Modeling and verification of interactive flexible multimedia presentations using promela/spin. In *The 9th International SPIN Workshop, LNCS 2318*, pages 205–212, Grenoble, France, April 2002.

[6] B. Bailey, J. Konstan, R. Cooley, and M. Dejong. Nsync - a toolkit for building interactive multimedia presentations. In *Proceedings of ACM Multimedia*, pages 257–266, September 1998.

[7] M. C. Buchanan and P. T. Zellweger. Scheduling multimedia documents using temporal constraints. In *Proceedings Third International Workshop on Network and Operating Systems Support for Digital audio and Video*, pages 237–249. IEEE Computer Society, November 1992.

[8] J. P. Courtiat and R. C. D. Oliveira. Proving temporal consistency in a new multimedia synchronization model. In *Proceedings of ACM Multimedia*, pages 141–152, November 1996.

[9] S. Gibbs, C. Breiteneder, and D. Tsichritzis. Data Modeling of Time-Based Media. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pages 91–102, Minneapolis, Minnesota, May 1994.

[10] R. Hamakawa and J. Rekimoto. Object composition and playback models for handling multimedia data. *Multimedia systems*, 2:26–35, 1994.

[11] I. Herman, N. Correira, D. A. Duce, D. J. Duke, G. J. Reynolds, and J. V. Loo. A standard model for multimedia synchronization: Premo synchronization objects. *Multimedia systems*, 6(2):88–101, 1998.

[12] N. Hirzalla, B. Falchuk, and A. Karmouch. A Temporal Model for Interactive Multimedia Scenario. *IEEE Multimedia*, 2(3):24–31, 1995.

[13] G. Holzmann. *Design and Validation of Computer Protocols*. Englewood Cliffs, N.J.: Prentice Hall, 1991.

[14] G. J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[15] W. Hurst and R. Muller. A synchronization model for recorded presentations and its relevance for information retrieval. In *Proceedings of ACM Multimedia*, pages 333–342, October 1999.

[16] M. Jourdan, N. Layaida, C. Roisin, L. Sabry-Ismail, and L. Tardif. Madeus, an authoring environment for interactive multimedia documents. In *Proceedings of ACM Multimedia*, pages 267–272, September 1998.

[17] T. Little and A. Ghafoor. Synchronization and Storage Models for Multimedia Objects. *IEEE Journal on Selected Arears in Communications*, 8(3):413–427, April 1990.

[18] T. Little and A. Ghafoor. Interval-based conceptual models for time-dependent multimedia data. *IEEE Transactions on Knowledge and Data Engineering*, 5(4):551–563, 1993.

[19] D. McCarthy and U. Dayal. The architecture of an active data base management system. In *Proceedings ACM SIGMOD Conference on Management of Data*, pages 215–224, 1989.

[20] I. Mirbel, B. Pernici, T. Sellis, S. Tserkezoglou, and M. Vazirgiannis. Checking temporal integrity of interactive multimedia documents. *VLDB Journal*, 9(2):111–130, 2000.

[21] J. Nang and S. Kang. A new multimedia synchronization specification method for temporal and spatial events. In *Proceedings of International Conference on Multimedia Computing and Systems*, pages 236–243. IEEE Computer Society, June 1997.

[22] B. Prabhakaran and S. Raghavan. Synchronization Models for Multimedia Presentation with User Participation. *Multimedia Systems*, 2(2), 1994.

[23] P. Pzandak and J. Srivastava. Interactive multi-user multimedia environments on the internet: An overview of damsel and its implementation. In *Proceedings of International Conference on Multimedia Computing and Systems*, pages 287–290. IEEE Computer Society, June 1996.

[24] L. Rutledge and L. Hardman. The rise and fall of multimedia authoring. In *Proceedings International Conference on Media Futures*, pages 17–20, May 2001.

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

[26] SMIL. http://www.w3.org/AudioVideo.

[27] M. Vazirgiannis, Y. Theodoridis, and T. Sellis. Spatio-temporal composition and indexing for large multimedia applications. *Multimedia Systems*, 6(4):284–298, 1998.

[28] K. Yoon and P. B. Berra. Topcn: Interactive temporal model for interactive multimedia documents. In *Proceedings of International Conference on Multimedia Computing and Systems*, pages 136–144. IEEE Computer Society, June 1998.

[29] A. Zhang, Y. Song, and M. Mielke. *NetMedia*: Streaming Multimedia Presentations in Distributed Environments. *IEEE Multimedia*, 9(1):56–73, 2002.

Ramazan S. Aygün Ramazan S. Aygün received the B.S. degree in computer engineering from Bilkent University, Ankara, Turkey in 1996, the M.S. degree from Middle East Technical University, Ankara in 1998, and the Ph.D. degree in computer science and engineering from State University of New York at Buffalo in 2003. He is currently an Assistant Professor in Computer Science Department, University of Alabama in Huntsville. His research interests include multimedia databases, multimedia synchronization, and video processing.

Aidong Zhang Aidong Zhang received her Ph.D degree in computer science from Purdue University, West Lafayette, Indiana, in 1994. She was an assistant professor from 1994 to 1999, an associate professor from 1999 to 2002, and has been a professor since 2002 in the Department of Computer Science and Engineering at State University of New York at Buffalo. Her research interests include databases, multimedia systems, content-based image retrieval, bioinformatics, and data mining. She is an author of over 150 research publications in these areas. Dr. Zhang's research has been funded by NSF, NIMA, NIH.

Dr. Zhang serves on the editorial boards of International Journal of Bioinformatics Research and Applications, ACM Multimedia Systems, the International Journal of Multimedia Tools and Applications, and International Journal of Distributed and Parallel Databases. She was co-chair of the technical program committee for ACM Multimedia 2001. She has also served on various conference program committees. Dr. Zhang is a recipient of the National Science Foundation CAREER award and SUNY Chancellor's Research Recognition award.