Rule-based Flexible Synchronization Modeling with Model Checking *

Ramazan Savaş Aygün and Aidong Zhang

Department of Computer Science and Engineering

State University of New York at Buffalo

201 Bell Hall Box 602000

Buffalo, NY 14260-2000

{aygun,azhang}@cse.buffalo.edu

*This research is supported by NSF grant IIS-9733730.

Abstract

Specification as in SMIL usually considers forward presentation without considering interactions that change the course of the presentation like backward and skip. Also, management of the presentation constraints become harder when the course of the presentation changes. In this report, we introduce a synchronization model, termed *RuleSync*, for management of robust, consistent, and flexible presentations that include a comprehensive set of interactions. RuleSync manipulates synchronization rules that are managed 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 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.

Keywords: Multimedia Synchronization, Model Checking, Synchronization Rules, Multimedia Presentations, Linear Temporal Logic.

1 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. Applications like video teleconferencing are live presentations and user interactions are usually limited in terms of accessing the presentation. On the other hand, applications like asynchronous learning and collaborative engineering may exploit recorded presentations and users may later access and interact with these multimedia presentations. 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 user interactions that change the course of a presentation either increase the complexity of the specification or are not allowed.

Multimedia presentation management research started with organization of streams that participate in the presentation and VCR-based user interactions are incorporated at different levels at the later research. Initial models only consider simple interactions like play, pause, and resume. Flexible models do not enforce timing constraints and rather temporal organization is performed by relating events in the presentation. For example, *stream A starts after (meets) stream B.* There is no enforcement on media clock time like *stream B* has to end at an instant and at that instant *stream A* has to start. Since there may be delay in the play of

stream B, to start *stream A* after *stream B* brings flexibility by not enforcing timing constraints. Speed-up and slow-down operations are included at a later stage in initial models. Skip and backward interactions are able to be incorporated in time-based models. But flexible models could not incorporate these functionalities. There are only few flexible models considering skip operation. These models have some restrictions on the application of skip functionality.

Nevertheless, to our knowledge, there is no implemented flexible model that supports backward functionality. The main difficulty behind backward and skip is that these interactions change the course of the presentation. Although conceptual models have been proposed to handle backward and skip [31, 26], the implemented systems could not use these models since they require the authors to spend enormous time on modeling. More generally, we believe that these implemented systems ignored skip and backward due to following reasons:

1. Most of the presentations included compressed video stream like MPEG. Compressed video is designed for forward display and important frames appear first in the nominal presentations. But they appear the last in the backward direction thus making the backwarding very difficult. If video cannot be played in backward direction, there is no need to backward a presentation. Skipping is not easy since each frame in the video has a different size. It is not possible to skip to a specific frame since its exact location in the video is unknown. Nowadays, by using buffering and preprocessing techniques it is possible to perform backwarding and skipping to any point in the video [32].

2. Most of the presentations also included audio. Playing audio in the backward direction does not make any sense. Nowadays, audio is accompanied with closed caption. Even though audio is not played, the text can be displayed in the backward direction.

3. Inherent deficiency of the models could not deal with these operations. Especially, it is hard to manage these user interactions in event-based models. Because some events may be skipped or signaled again yielding false presentations. For the backward presentations, it is even unknown what to perform in the backward direction.

4. It is assumed that management of these functionalities is easy and they can be incorporated into the system easily later. Management of these functionalities looks like easy but is actually hard. At first sight, it looks like reversing the relationships yields the backward presentation, which is not true. There has been a lot of time spent for proper specification of forward presentations to capture the synchronization requirements. Since, presentations are not delivered in a perfect world, each minor

difference in the specification may yield a different presentation. From a given forward presentation specification, a number of backward presentations may be specified. The problem is to determine the best one, which is compliant with the author's forward presentation.

The user interactions that change the course of the presentation are necessary in applications where a detailed investigation of a presentation is required. Distance education and sports presentations are examples where this type of user interactions are needed. Backward and skip are two examples of user interactions of this kind at the lowest level. Skip directs to a new position where some constraints may be violated and the presentation may yield a false presentation. The backward presentation on the other hand requires more information that is usually not specified. We focus on VCR-based user interactions in this report.

1.1 Related Work

Allen [1] introduced 13 primitive temporal relationships for time intervals. This model describes how the time instants and presentation durations of two multimedia objects are related. It does not quantify the temporal parameters for time instants and duration of presentations. Little et. al. [20] extended these with n-ary and reverse relationships. The reverse relationships can declare relationships for backward presentations. *Overlaps, during,* and *finishes* can only be expressed with time values in these models. The verification of a multimedia presentation usually consists of verification of these relationships.

One important factor in modeling of presentations is whether the model is a time-based or an event-based. A lot of work has been done on time-based modeling. Timed Petri Nets are first introduced for multimedia presentations in OCPN [19] and extended with user interactions in [31]. The modeling of user interactions using Petri-Nets has been covered in [26]. The backward and skip has also been covered but for each possible skip and backward operation (including the current position of presentation and where skip is performed), a Petri-Net has to be constructed. These petri-nets are not connected to each other, verification and consistency of the whole system is difficult to handle. So, a model where all operations can be incorporated and verified is needed. Gibbs [11] proposed a way of composing objects through BLOB stored in the databases and created objects by applying interpretation, derivation and temporal composition to a BLOB. The temporal composition is based on the start times of media objects on a timeline. Hamakawa et al. [12] has 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 [17]. NSync [6] is a toolkit which manages synchronous and asynchronous interactions, and

fine-grained synchronization. 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*". Time-based models usually keep the start time and duration of each stream and these models modify the duration and start time after each interaction of the system or the user.

In an event-based model, the start of a stream depends on an event signal. Events for multimedia applications are discussed in [30] and a model that includes temporal and spatial events is given in [24]. SSTS [24] is a combination of Firefly's [7] graph notation and transition rules of OCPN [19]. SSTS has AND-start, AND-end, OR-start and OR-end nodes to satisfy the synchronization requirements of multiple streams. The relationships among streams are based on binary relations. SSTS does not cover any user interactions. DAMSEL [27] 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". The occurrence of an event may also defer the occurrence of another event. Temporal constraints that are used in Madeus [18] are based on Allen's temporal relations [1]. FLIPS [28] is an event-based model that has barriers and enablers to satisfy the synchronization requirements at the beginning and the end of the streams. It does not have complex user interactions such as fast-forward and fast-backward but it has a limited skip operation which moves to the beginning of another object. The functionalities of an application are grouped as pre-orchestrated and event-based in [30]. The pre-orchestrated actions are the actions that are known prior to the presentation whereas the event-based ones are triggered by events. A timeline approach with event-based modeling is proposed in [14]. User interactions are considered but not VCR functions such as fast-forward, fast-backward, or skip. They emphasize the synchronous and asynchronous events. PREMO [13] presents an event-based model that also manages time. They have synchronization points which may also be AND synchronization points to relate several events. Time for media is managed with clock objects and time synchronizable objects which contain timer. Multimedia synchronization using ECA rules is covered in [3, 2, 4]. These projects show how synchronization requirements can be modeled using ECA rules and form the basis of ECA rule modeling in this report. A hierarchical synchronization model that has events and constraints is given in [9].

SMIL [29] is a mark-up language for publishing synchronized multimedia presentations via the Internet. It is unclear how the user's input affects the presentation. Marcus and Subrahmanian [21] consider presentation creation depending on the consecutive queries and constraints submitted by the user. The user specifies how the query results will be presented.

1.2 Motivation and Our Approach

Defining reverse temporal relationships depending on forward temporal relationships does not solve the problem. For example, the reverse temporal relationship for "A meets B" is "B meets A". There are a couple of scenarios that "A meets B" holds. Ending of A may start B, beginning of B may end A or an independent event may end A and start B. This kind of specification is possible in SMIL. In all cases, *meets* relationship holds. Should backwarding A terminate B in backward direction or should ending of B in backward direction backward A or should an independent event end B and backward A? Even for a single relationship, there are many options. "Which option is better than the others?" is one of the questions that we try to answer for event-based models.

There are tools that enable the querying of temporal relationships in multimedia presentations. These tools compare the time intervals of streams and may check the correctness of the specification at a level. Flexible models usually do not consider time instants and may yield different presentations from expected. These models do not aim to achieve deadlines for the playout of streams. In most cases in a distributed environment, these expected deadlines are violated. The author or the user may receive an unsynchronized presentation. Some of the questions that the author should consider are "is there a better specification for the presentation?" and "does the synchronization tool really satisfy the temporal relationships?". The querying tools usually assume perfect on-time presentations of streams. The author cannot verify whether the requirements are really satisfied by the model.

The previous work has limited user interactions, not supported necessary synchronization requirements, or has assumptions on the structure of the multimedia presentation. FLIPS [28] has powerful synchronization requirements but cannot support backward and it has limited skip functionality. Hurst et.al. [17] requires at least one master stream throughout the presentation. NSync [6] does not allow backward and it can only allow skip after user also specifies the operations for each interval of skip. Either robust synchronization models were proposed with limited user interactions and/or complex specification or less robust synchronization models while enabling comprehensive user interactions were presented. The backwarding or skipping is considered at the modeling level using Petri-Nets. This requires the author to model different petri-net for each different skip and backwarding. The authors usually do not have enough information about Petri-Nets or they do not want to spend so much time on detailed modeling. In the previous (time-based) models, most of the information given by the user was satisfactory for the model and user interactions. As the synchronization requirements are specified by events, constraints or synchronization expressions, the

information to the system became implicit. To overcome this, either the user has to specify more information as in NSync or the functionalities are limited as in FLIPS and PREMO [13]. It is still a question what to specify and how much to specify. We consider SMIL expressions as the level of specification and show how our model is powerful. The reason using SMIL expressions is to show the compatibility of model with SMIL. However, it is also possible to process synchronization rules directly.

It has been shown that event-based models have been more robust and flexible for multimedia presentations. The bottleneck of the event-based models is the inapplicability of the model in case there is a change in the course of the presentation (like backwarding and skipping). Most of the previous models are based on event-action relationships. The condition of the presentation and participating streams also influence the actions to be executed. Thus event-condition-action (ECA) rules [22] 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*. Since the structure of a synchronization rule is simple, the manipulation of the rules can be performed easily in existence of user interactions. 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 synchronization rules can easily be regenerated from SMIL expressions. The authors usually detain from providing extra information for backward and skip operations. This kind of information should be deduced by the model and corrected by the author if it is necessary. This deduction is based on author's specification. We assume that there are reasons behind the way the author makes the specification and deduction of the rules are bound by these reasons. A middle layer between the specification and the synchronization model which assists the synchronization model to provide user interactions while keeping the synchronization specification minimal is proposed. We call this middle layer as *middle-tier*. The middle tier is previously defined as the logical layer in a distributed system between a user interface or Web client and the database. It is a collection of business rules and functions that generate and operate upon receiving information. The middle-tier for multimedia synchronization handles synchronization rules that can be extracted explicitly from the user specification and synchronization rules that can be deduced implicitly from explicit synchronization rules. The middle-tier reduces the amount of user specification while increasing the power of the synchronization model. The synchronization model also generates a virtual timeline to manage the user interactions that change the course of the presentation.

The verification and correctness of schedules are also important. Most of the time, it is the responsibility of the author to verify the requirements in the specification. Today any user without sophisticated information

about verification can specify a multimedia presentation. The general method to check these is either theoretical verification, simulation or testing. These are usually hard for the user(author) to handle. Model checking is a technique that automatically detects all the states that a model can enter and checks the truthness of well-formed formulas. Moreover model checking can present contradictory examples if the formulas are not satisfied. PROMELA/SPIN [15, 16] tool has been used for model checking which checks LTL (Linear Temporal Logic) formulas. These formulas can automatically be generated and verified. We give examples of how our model is built using PROMELA and truthness of formulas are checked by SPIN [16]. The author does not have to know this tool but needs to know what to verify. We developed a software that the author can verify the truthness of the properties.

This report is organized as follows. The following section explains the synchronization rules. Section 3 presents the components of the synchronization model. The user interactions are covered in Section 4. The modeling and specification are explained in Section 5 and 6, respectively. Section 7 concludes the report.

2 Synchronization Rules

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

Definition 1 *A synchronization rule is composed of an event expression, condition expression and action expression which 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.

2.1 Events, Conditions and Actions for a Presentation

In a multimedia system, the events may be triggered by a media stream, user or the system. Each media stream is associated with events along with its data and it knows when to signal events. When events are received, the corresponding conditions are checked. If a condition is satisfied, the corresponding actions are executed.

Definition 2 *An event expression manages the relationships among the events and can be defined in the following format:*

eventExpression = *eventExpression && eventExpression*

| *eventExpression || eventExpression*

| *(eventExpression)*

| *event;*

The event expression enables the composition of events may be required to trigger actions instead of a single event. Composite events can be created by boolean operators && and ||.

The goal in inter-stream synchronization is to determine when to start and end streams. The start and end of streams depend on multimedia events. The hierarchy of multimedia events are depicted in Fig. 1. The user has to 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. Temporal events may be *absolute* with respect to a specific point in a presentation (e.g. the beginning of a presentation). Temporal events may also be *relative* with respect to another event.

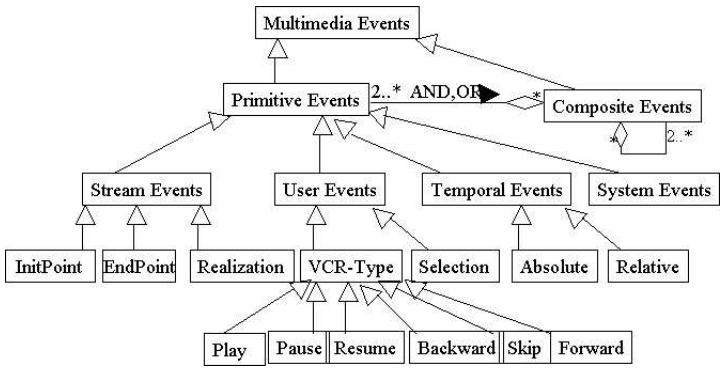


Figure 1: The event hierarchy.

Definition 3 *An event is represented with source(event_type[, event_data]) where source points the source of the event, event_type represents the type of the event and event_data contains information about the event.*

Event source can be the user or a stream. Optional event data contains information like a realization point. Event type indicates whether the event is *InitPoint*, *EndPoint* or *realization* if it is a stream event. Each stream has a series of events. Users can also cause events such as start, pause, resume, forward, backward

and skip. These events have two kinds of effects on the presentation. Skip and backward change the course of the presentation. Others only affect the duration of the presentation.

Definition 4 *The condition expression determines the set of conditions to be validated when the event expression is satisfied and has the following form:*

```
conditionExpression =   conditionExpression && conditionExpression
                      | conditionExpression || conditionExpression
                      | ( conditionExpression )
                      | condition;
```

Definition 5 *A condition in a synchronization rule is a 3 tuple* $C = condition(t_1, \theta, t_2)$ *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.*

A condition indicates the status of the presentation and its media objects. The most important condition is whether the direction of the presentation is forward. The receipt of the events matter when the direction is forward or backward. Other types of conditions include the states of the media objects.

Definition 6 *The action expression is the list of the actions to be executed when condition is satisfied:*

```
actionExpression  =  action | actionExpression;
```

Definition 7 *An action is represented with* $action\_type(stream[, action\_data], sleeping\_time)$ *where* $action\_type$ *needs to be executed for stream using* $action\_data$ *as parameters after waiting for* $sleeping\_time$*. Action_data can be the parameter for speeding, skipping, etc.*

An action indicates what to execute when conditions are satisfied. *Starting* and *ending* a stream, and *displaying* or *hiding* images, slides and text are sample actions. For backward presentation, *backwarding* is used to backward and *backend* is used to end in the backward direction. 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 associated with some specific time. The deferred action can only start after this *sleeping_time* has been elapsed. If an action has started and had not finished yet, that action is considered as an alive action.

3 The Synchronization Model

The middle-tier is responsible for extraction of synchronization rules. In this section, firstly the role of the middle-tier is explained briefly. The elements of a multimedia presentation is explained along with SMIL expressions. The rule generation from SMIL expressions is covered with an example. Receivers, controllers, and actors are basic components of the synchronization model and responsible for management of synchronization rules. The timeline is used to keep track of expected time of actions, receipts of events, satisfaction of controllers, and activation of actors. The timeline is mainly used for user interactions that change the course of the presentation.

3.1 The Middle-Tier

The middle-tier for multimedia synchronization first handles the rules that can be extracted from the synchronization specification. Synchronization requirements are stored in rules since each synchronization rule is simple and can be processed easily to generate other rules. Once the rules from the specification are extracted, the synchronization rules for the backward presentation are generated. The extracted rules are fed into the synchronization model. The synchronization model contains a rule manager to manage these rules. The timeline for events and actions are generated in case the course of the presentation changes after user interactions like skip and change direction. When the presentation module receives an event from the user or one of the stream handlers, it informs the event and the current condition of the presentation to the synchronization model. The synchronization model determines if any of the rules are satisfied and if a rule is satisfied it informs the necessary actions to the presentation module. The framework is shown in Fig. 2.

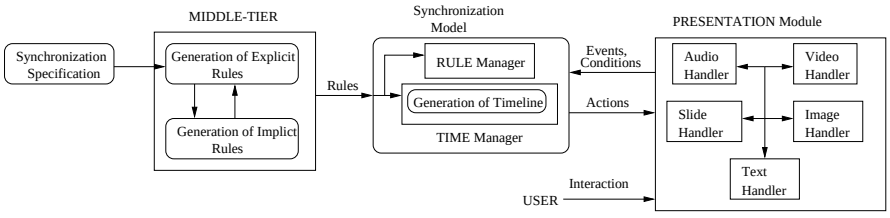


Figure 2: The role of middle-tier.

3.2 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 containers or other streams. This allows grouping of streams and creation of

subpresentations. The containers can signal InitPoint and EndPoint events. This means that the container initiates its presentation and the container ends either one or more of its components end or is ended by other containers or streams.

Explicit rules are generated by processing the synchronization specification. In SMIL 1.0, there are two kinds of grouping, parallel and sequential. The beginning of a group is determined by an event signaled from another group, a stream or the user. If the group is the first group that is presented in the multimedia presentation, the user event *USER(Start)* determines the beginning of the presentation. The parallel grouping corresponds to the list of all actions that will start when the group starts. Thus, if the parallel grouping is like

<par>
  <... id="id1" ...>
  <... id="id2" ...> ...
  <... id="idn" ...>
</par>,

the synchronization rule is as follows:

R_{par} : **on** ... **if** direction=forward **do** start(id1)
start(id2)
...
start(idn)

In the sequential grouping, the end of a stream triggers start of another stream. If the sequential group has n elements, there are n-1 rules for the group. Thus, if sequential grouping is like

<seq>
  <... id="id1" ...>
  <.. id="id2" ...> ... <... id="id(n-1)"> <.. id="idn" ...>
</seq>

The synchronization rules that will be generated are as follows:

R_{seq1} : **on** ... **if** direction=forward **do** start(id1)
R_{seq2} : **on** id1(EndPoint) **if** direction=forward **do** start(id2)

...

$R_{seq(n-1)}$: **on** id_{n-1}(EndPoint) **if** direction=forward **do** start(idn)

Notice that the direction is considered as forward in the condition part since the user specifies the requirements for the forward presentation. If time is associated to start a stream (e.g. start a stream after 2 seconds), time is considered part of the action rather than part of the event. Because including time in the event expression increases the number of rules significantly (i.e. the same event may also trigger other actions).

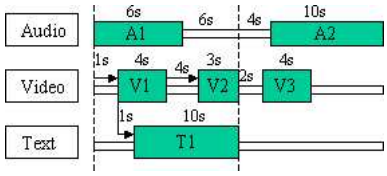


Figure 3: Sample Presentation.

A sample presentation is depicted in Fig. 3. There are 6 stream elements: A1, A2, V1, V2, V3 and T1. A1 and A2 are audio elements. V1, V2, and V3 are video elements and T1 is a text element. Assume that the presentation is grouped according to the SMIL expression given in Fig. 4.

There are four containers in the presentation: sequential presentation of V1 and V2 (SEQ1), parallel presentation of A1, T1 and SEQ1 (PAR1), parallel presentation of A2 and V3 (PAR2) and sequential presentation of PAR1 and PAR2 (MAIN). We have the following synchronization rules given in Fig. 5. The event-action relationships for PAR1 container is depicted in Figure 6.

3.3 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.

Definition 8 *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 will receive e. When receiver R receives event e, it sends information of the receipt of e to all its controllers in C. A

```
<seq>
    <par endsync=''last''>
        <audio id=''A1'' src=''cnn.aiff''/>
        <seq>
            <video id=''V1'' src=''cnn1.mpv'' begin =''1s''/>
            <video id=''V2'' src=''cnn2.mpv'' begin =''4s''/>
        </seq>
        <text id=''T1'' src=''leader_title.html'' begin =''id(V1)(1s)'' dur=''10s''/>
    </par>
    <par>
        <video id=''V3'' src=''cnn3.mpv'' begin =''2s''/>
        <audio id=''A2'' src=''cnn2.aiff'' begin=''4s''/>
    </par>
</seq>
```

Figure 4: The SMIL expression.

receiver object is depicted in Fig. 7(a). There is a receiver for each single event. The receivers can be set and reset by the system anytime.

According to the synchronization rules given in Fig. 5, there are 13 receivers for each event specified in the event expression. These receivers are R1: user(START), R2: MAIN(InitPoint), R3: PAR1(InitPoint), R4: SEQ1(InitPoint), R5: V1(InitPoint), R6: V1(EndPoint), R7: V2(EndPoint), R8: SEQ1(EndPoint), R9: PAR1(EndPoint), R10: PAR2(InitPoint), R11: V3(EndPoint), R12: A2(EndPoint) and R13: PAR2(EndPoint).

Definition 9 *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 Fig. 7(c). Controllers can be set or reset

```
(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 SEQ1(InitPoint)                      if direction=FORWARD   do start(V1,1s)
(F5)   on V1(InitPoint)                        if direction=FORWARD   do start(T1,1s)
(F6)   on V1(EndPoint)                         if direction=FORWARD   do start(V2,4s)
(F7)   on V2(EndPoint)                         if direction=FORWARD   do end(SEQ1)
(F8)   on (SEQ1(EndPoint) && A1(EndPoint)
          && T1(EndPoint))                     if direction=FORWARD   do end(PAR1)
(F9)   on PAR1(EndPoint)                       if direction=FORWARD   do start(PAR2)
(F10)  on PAR2(InitPoint)                      if direction=FORWARD   do start(A2,4s)
                                                                      start(V3,2s)
(F11)  on (V3(EndPoint) && A2(EndPoint))       if direction=FORWARD   do end(PAR2)
(F12)  on PAR2(EndPoint)                       if direction=FORWARD   do end(MAIN)
```

Figure 5: Forward synchronization rules.

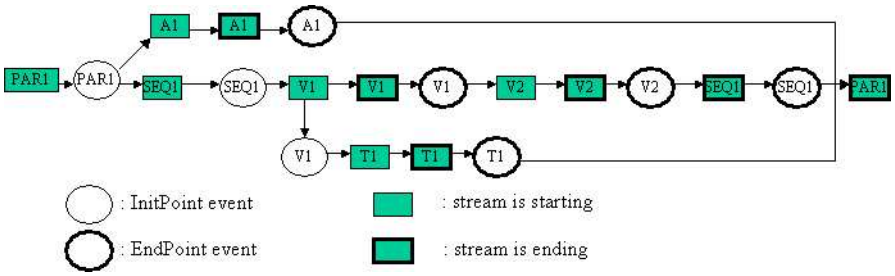


Figure 6: The event action relationships for PAR1.

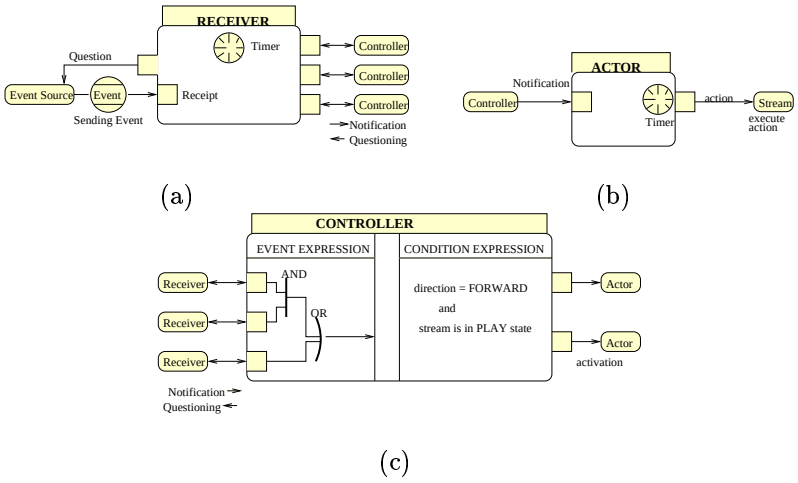


Figure 7: (a) A receiver object (b) An actor object, and (c) A controller object.

by the system anytime.

For the synchronization rules given in Fig. 5, we have a controller for each synchronization rule. So, we have 12 controllers (C1,C2,...,C12) which are listed in Fig. 8.

Definition 10 *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. 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 Fig. 7(b).

For the synchronization rules given in Fig. 5, we have one actor for each action. So, we have 15 actors (A1,A2,...,A15) which are listed in Fig. 8.

3.4 Timeline

If multimedia presentations are declared in terms of constraints, synchronization expressions or rules, the relationships among streams are not explicit. Because they only keep the relationships that are temporally adjacent or overlapping. The status of the presentation must be known at any instant. In our work, the timeline object keeps track of all temporal relationships among streams in the presentation.

Definition 11 *A timeline object is a 4-tuple $T = (receiverT, controllerT, actorT, actionT)$ where $receiverT$, $controllerT$, $actorT$, and $actionT$ are time-trackers for receivers, controllers, actors and actions, respectively.*

The time-trackers $receiverT$, $controllerT$, $actorT$ 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. Since skip and backward operations are allowed, alive actions, received or not-received events, sleeping actors and satisfied controllers must be known for any point in the presentation. The time of actions can be retrieved from the timeline object.

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*, *InitPoint* and *EndPoint* stream events 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 problem is the determination of the start time of the action. 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 the event composition condition of the controller is satisfied. Algorithms 1 and 2 find the time of the receipt of an event for a receiver and start time of actions, respectively.

Algorithm 1 Time Receiver::findReceiptOfEvent()

```
eventSource ← source of event e (of this receiver R = (e, t, C))
if (eventSource=USER and eventType=START) then
  start time ← 0
else
  find the action (start or display) for the eventSource
  start time ← action.findStartTime()
  if (event type=END) then
    start time ← start time + duration of the stream
  else if (event type=REALIZATION) then
    start time ← start time + realization time of the stream
  end if
end if
return start time
```

Algorithm 2 findStartTimeOfActions()

```
for each controller C = (R, ee, ce, A) do
  (the expected time of C) ← C.findExpectedTimeOfController()
  for each actor Actor = (a, t) in A do
    expectedTime ← (the expected time of C) + t
    (the expected time of a) ← expectedTime
  end for
end for
```

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 handled in the following way: In our model, events can be composed using && and || 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 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. Although the above algorithms seem to be in infinite loop since the expected time for receivers depend on the expected time of controllers and the expected time of controllers depend on the receipts of events by receivers, the time of the first controller and the receiver is 0 which only depends on the user event to start the presentation. The timeline for receivers, controllers and actors for synchronization rules given in Fig. 4 are listed in Fig. 8. On top of the figure the receivers, the controllers and the actors for the presentation are listed. The four time-trackers are shown at the bottom side. 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 the actor shows its activation (sleeping time) and underlined actor shows the ending of sleeping time. The actions are also displayed in the same way. The name of the container or the stream shows its starting time and if it is underlined it shows the 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.

4 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 and access of multimedia presentations. Event-based models can handle play, pause, resume, speed-up, and slow-down operations easier than time-based models. 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 event-based models, these interactions only need to inform active streams. In our case, the time is connected to actors. 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

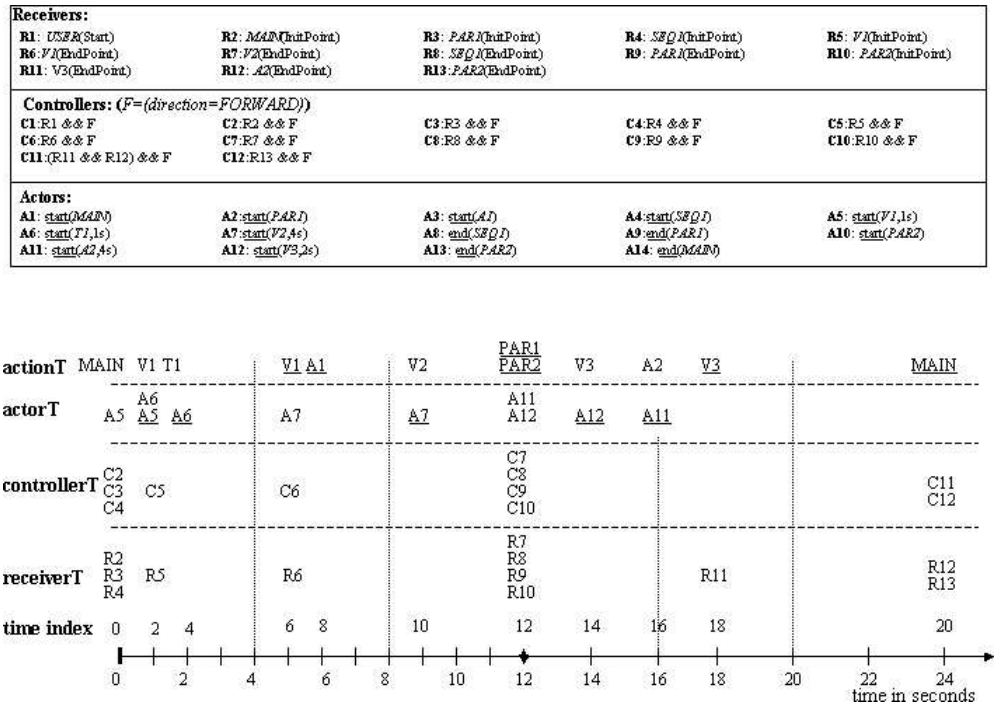


Figure 8: The timeline.

needs to sleep for $(sleepingTime)/(|speedOfPresentation|)$. Speeding up or slowing down only requires the update of the speed of the presentation.

4.1 Skip Operation

A multimedia presentation has a lifetime. 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 that may cause ignorance of future streams that depend on the receipt of these events. The problem can be 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 time-trackers of the timeline. It is not always reasonable to start the actors whose controllers are satisfied, since their actions must 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* has not expired are allowed to sleep for the remaining time.

For example, if a skip is requested to the mid (12 seconds) of the presentation that is shown in Fig. 4, the timeline is followed in Fig. 8. Receivers R2, R3, R4, R5, R6, R7, R8, R9 and R10 are assumed to receive

their events. Receivers R11, R12 and R13 are assumed that they did not receive their events. Controllers C2, C3, C4, C5, C6, C7, C8, C9, and C10 are assumed to be satisfied. C11 and C12 are assumed to be not satisfied. A satisfied controller cannot notify its actors. It is assumed that it already notified its actors. At the middle point, there is no sleeping actor. The actors A11 and A12 are activated. So, all the actors should be set to their original time. MAIN container should be active since it has elements that are still active. V1, T1, V2 and A1 should be idle. PAR1 should be idle too. PAR2 should be set to its InitPoint so that it can start the streams that it contains. V3 and A2 must also be idle.

4.2 Backward Presentation

If the direction of the presentation is modified, then receiver conditions, controllers, and actors still 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 or skipping, and this alleviates the specification of the presentation substantially. It is desirable that the backward synchronization rules are compliant with the forward presentation. 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 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 by starting or ending, the author considered A as a master stream over B. A should be master at this point in the backward direction. 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 will participate in ending C in the backward direction. A container ends when all elements are played.*

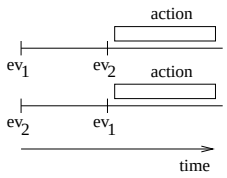


Figure 9: Co-occurrence.

Author Property 4 ***Co-occurrence*** *If (ev_1 && ev_2) influences action, their co-occurrence is effective in the forward direction. That is, the action will take place after both events are signaled (Fig. 9). The action should be terminated when one of the events is signaled in the backward direction (Fig. 9). This corresponds to self-occurrence in the backward direction.*

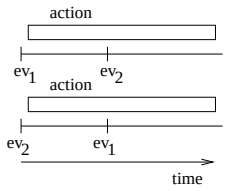


Figure 10: Self-occurrence.

Author Property 5 ***Self-occurrence*** *If (ev_1 || ev_2) influences action, their self-occurrence is effective in the forward direction. That is, the action will take place after one of the events is signaled (Fig. 10). The action should be terminated when both events are signaled in the backward direction (Fig. 10). This corresponds to co-occurrence in the backward direction.*

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 corresponds to when frame P is displayed. If $A(realization, P)$ influences B in forward direction, then realization of $P-1$ is important for B in the backward direction. This corresponds to the end of playing P.*

The following logic is used for the generation of the backward presentation based on the previous *author properties*. The relationships are depicted in Fig. 11.

- **EndPoint-Start Relationship.** If the end of A participates in starting B, when B reaches its beginning in the backward presentation, it will participate in starting A in the backward direction *(Dependency property)*.

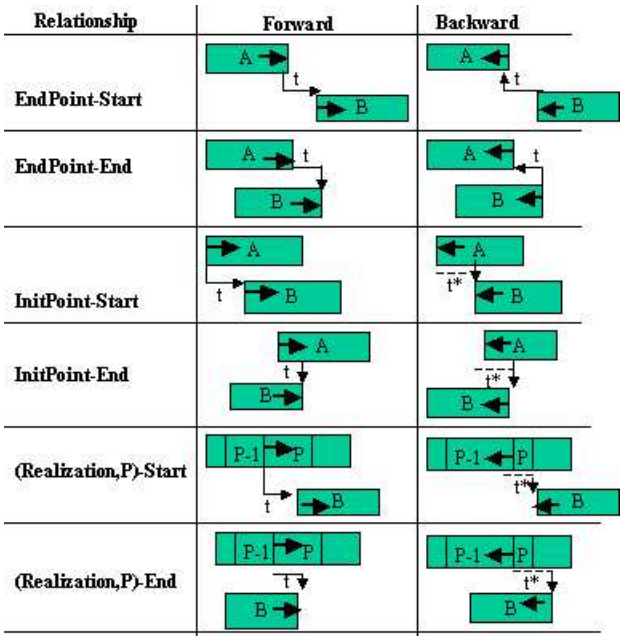


Figure 11: Forward-backward relationships without time.

- **EndPoint-End Relationship.** If the end of A participates in ending B, when A starts in the backward direction, it will participate in starting B in the backward direction *(Master-Slave property)*.

- **InitPoint-End Relationship.** If the start of A participates in ending B, when A ends in the backward direction, it will participate in starting B in the backward direction *(Master-Slave property)*.

- **InitPoint-Start Relationship.** If the start of A participates in starting B, when A ends in the backward direction, it will participate in ending B in the backward direction *(Master-Slave Rule)*. If the start of container C starts its elements , when its elements reach their beginning in the backward presentation, they will participate in ending C in the backward direction (*Hierarchy property)*.

- **InitPoint and EndPoint Events in Composite Events.** If an InitPoint event or an EndPoint event exists in a composite event, they are treated individually whether the event composition is an AND or OR composition. Therefore, one of the previous rules are applied.

- **Realization-Start Relationship.** The realization points are monotonically increasing within a stream. If $A(realization, P)$ participates in starting B, then $A(realization, P-1)$ will participate in ending B in the backward presentation. At first sight it seems that P should cause the end of B which is not true. Consider the presentation of an image along with slides. Each slide has a duration of 1 minute. The image is displayed when the second slide is displayed. On the timeline, the display

of the image will be at the beginning of the 2^{nd} minute. The image must be closed when the first slide is displayed in the backward presentation and must be visible during the second slide *(Realization property)*.

- **Realization-End Relationship.** Assume that the realization point is P again. If $A(realization, P)$ participates in ending B, then $A(realization, P-1)$ will participate in starting B in the backward presentation *(Realization property)*.

- **Realization Events in Composite Events.** Let $A(realization, P_1)$ and $B(realization, P_2)$ be realization events for streams A and B. If $(A(realization,\ P_1)\&\&\ B(realization, P_2))$ causes some actions in the forward presentation, $A(realization, P_1 - 1)\ ||\ B(realization, P_2 - 1)$ 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 is realized in the backward direction. In this way, the integrity and the consistency of the presentation can be protected. If $(A(realization, P_1)\ ||\ B(realization, P_2))$ causes some actions in the forward presentation, $(A(realization, P_1 - 1)\ \&\&\ B(realization, P_2 - 1))$ will cause actions in the backward presentation *(Co-occurrence and self-occurrence properties)*.

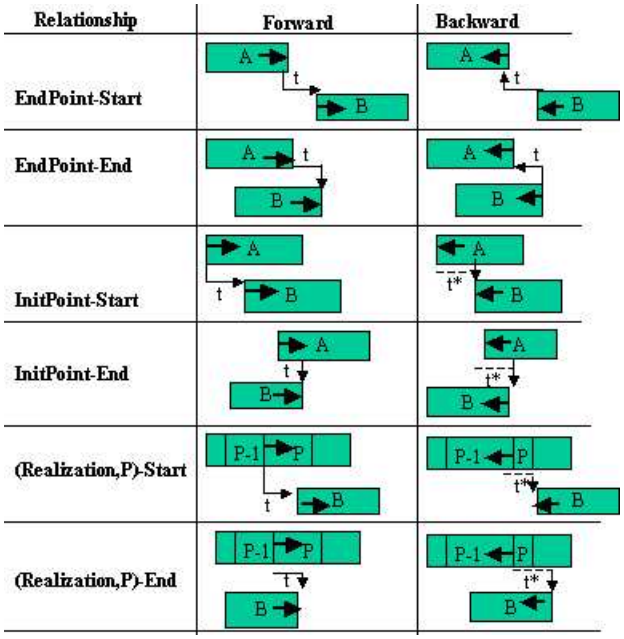


Figure 12: Forward-backward relationships with time.

4.2.1 Management of Time in Backward Presentation

The management with time for backward presentation may introduce some ambiguities. In Fig. 12, the time between relationships is indicated with t. EndPoint-Start and EndPoint-End relationships can be converted to backward relationships as shown in Fig. 12. Backwarding InitPoint-Start and InitPoint-End relationships are not easy to handle. We denote time with t^* for backward presentation. There is ambiguity in using of time in the literature and in synchronization models. Assume a video stream has a duration of 10 seconds. The question is whether expressions "7 seconds after the beginning" and "3 seconds before ending" are equivalent or not. In a network, the presentation of a video stream may have different durations due to delay and loss on the network. It is even harder to figure out when the video will end if the data has not arrived yet. In the specification, this may be considered as equivalent. Although the author uses time, it is favored because of simplicity. In most cases, this kind of usage corresponds to the realization event. Most of the time expression "7 seconds after beginning" means "after 7 seconds in nominal display". Thus, in this one, it is 70% of the video stream. This clarification should be made for the forward presentation so that backward rules can be converted using realization events.

The problem is even more clear when realization events are used in a composite way or alone. We believe that if an action happens based on a time referring to a realization point, it does not make much sense. Instead of using time and realization point, the realization point may be moved to a new point thus removing time.

Time should be used if the referring stream has already ended. There is no other choice in this case. EndPoint-Start relationship is an example to this. EndPoint-End relationship may utilize realization event in the backward direction, since they are overlapping.

If there is a case like in Fig. 13 where an event starts many other streams with different starting times, the backwarding of A is not clear since B and C may not finish backwarding at the expected times. We suggest that stream A may be backwarded after $minimum(t1, t2)$ after B and C finish backwarding.

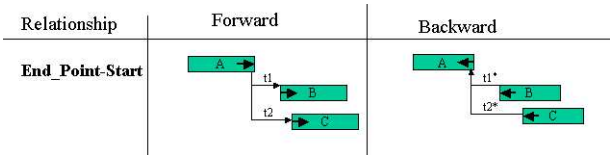


Figure 13: Ambiguity in relationships with time.

The synchronization requirements need to be specified more accurately. If two streams are overlapping

and time-based relationship between them is specified, it has to be distinguished from a realization event. For backward presentation, it is favorable to use realization events for forward presentation if possible instead of time. Whatever synchronization rules are generated for the backward presentation, the author may have different backward presentation. The author should always be allowed to update the rules.

4.2.2 Synchronization Rules for Backward Presentation

Each forward synchronization rule is processed to generate backward synchronization rules. The events in the event expression and the actions in the action expression are extracted to determine the relationships given in the previous subsections.

In our system, InitPoint and EndPoint events have dual actions. The dual actions for InitPoint and EndPoint is *backend* and *backward*, respectively. In the backward direction, InitPoint and EndPoint events are signaled when *backend* and *backward* actions are performed, respectively. The actions *start* and *end* also have dual events. The dual events for *start* and *end* actions are InitPoint and EndPoint, respectively. Actions have also dual actions. The actions *start* and *end* have dual actions *backend* and *backward*, respectively. Conditions have also dual conditions. The condition (direction=BACKWARD) is the dual condition for (direction=FORWARD).

Figure 14 lists the backward synchronization rules that are generated from forward synchronization rules in Figure 5. The synchronization rule F1 declares what to do when the user starts the presentation. There will be a corresponding rule for the backward presentation. This rule (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. This corresponds to an InitPoint-Start relationship. MAIN is the container of PAR1 and backward rule will be generated using the hierarchy rule. The dual event for *start* action is InitPoint. The event expression will be PAR1(InitPoint). The action expression will be backend(MAIN). All the condition expressions will be *direction=BACKWARD*. The corresponding backward rule is B2 for F2. The rest of the rules are generated in a similar fashion.

5 Modeling

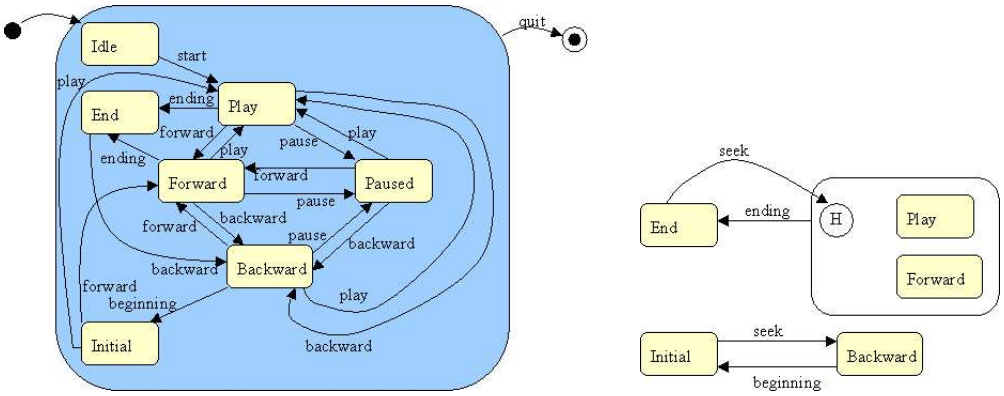
The most common methods for verification of finite-state concurrent systems are simulation, testing and deductive reasoning. It is not possible to consider all the cases in simulation and testing. If there is a severe

```
(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))      if direction=BACKWARD   do backend(PAR1)
(B4)  on V1(InitPoint)                           if direction=BACKWARD   do backend(SEQ1,1s)
(B5)  on V1(Realization,1s)                      if direction=BACKWARD   do backend(T1)
(B6)  on V2(InitPoint)                           if direction=BACKWARD   do backward(V1,4s)
(B7)  on SEQ1(EndPoint)                          if direction=BACKWARD   do backward(V2)
(B8)  on PAR1(EndPoint)                          if direction=BACKWARD   do backward(A1,6s)
                                                                            backward(SEQ1)
                                                                            backward(T1)
(B9)  on PAR2(InitPoint)                         if direction=BACKWARD   do backward(PAR1)
(B10) on (V3(InitPoint) && A2(InitPoint))        if direction=BACKWARD   do backend(PAR2,2s)
(B11) on PAR2(EndPoint)                          if direction=BACKWARD   do backward(A2)
                                                                            backward(V3,8s)
(B12) on MAIN(EndPoint)                          if direction=BACKWARD   do backward(PAR2)
```

Figure 14: Backward synchronization rules.

problem it may even be costly for the system to verify by testing and simulation. Deductive reasoning usually requires experts to verify and even sometimes they may not be able to verify. The major advantages of model checking is that it is automatic and usually fast. The counter examples are produced by the model checking tools. We believe that multimedia models should consider model checking first before implementation of the real system. We use PROMELA [15] as the specification language and SPIN [16] as the verification tool. These tools are publicly available and Linear Temporal Logic (LTL) formulas can be verified. The conversion of rules to PROMELA is explained in [5].

Model checking consists of three phases: modeling, specification, and verification. In our system, the user, the user interface, streams, containers, receivers, controllers and actors have to be modeled firstly. In the modeling phase, the model should be kept simple and avoid unnecessary details. The unnecessary details increase the computation for verification. More importantly they may cause false results. Therefore, we make some abstractions to ensure the correctness of the model. The major components of the model are



(a) (b)

Figure 15: The presentation states, (a) general state transitions (b) state transition for skip.

events, conditions, actions, receivers, controllers, actors, the presentation and the streams. The abstraction is performed on the streams and the user interface.

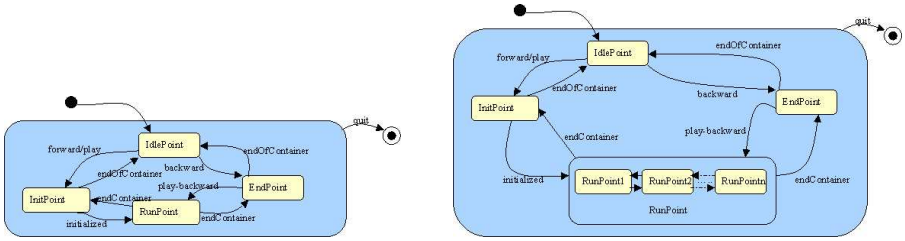
5.1 Presentation

The presentation can be in idle, initial, play, forward, backward, paused, and end states (Fig. 15). The presentation is initially in idle state. When the user clicks START button, the presentation enters play state. The presentation enters end state, when the presentation ends in the forward presentation. The presentation enters the initial state when it reaches its beginning in the backward presentation. The user may quit the presentation at any state. Skip can be performed in play, forward, backward, initial, and end states. If the skip is clicked in play, forward and backward states, it will return to the same state after skip unless skip to initial or end state is not performed. If the presentation state is in end state or in initial state, skip interaction will put into the previous state before reaching these states (Fig. 15 (b)). The presentation changes states as the user clicks on the button. The most important state variable of the presentation is the direction.

5.2 Containers and 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 the containers and streams that it contains. After the InitPoint state, a container enters its RunPoint state. In RunPoint state, a container knows that it has some streams that are being played. When all the streams it contains reaches to their end or when the container is notified to end, it stops

execution of the streams and signals its end and then enter idle state again. In the backward presentation, the reverse path is followed (Fig. 16 (a)).



(a) (b)

Figure 16: (a) container states (b) stream states

A stream is similar to the container. Since the number of states grow exponentially, some abstractions have to be made on modeling a stream. Since we are interested in interstream relationships, the points that affect the interstream relationships are considered. From the modeling point of view, if the displaying or playing a specific segment of a stream does not affect the playout of the presentation, there is no need to handle each segment of the stream. The streams only affect the model by the events that are triggered by the streams. The (dis)play of a stream element is in the model, if it triggers an event. If a video stream has 100 frames without an event for a frame, displaying a frame at an instant is not important if it does not trigger an event. If the display of frames are slow, a delay is assumed in playing all 100 frames.

If a stream does not signal any event except its start and end, the stream enters the same four states as a container. If a stream has to signal an event, a new state is added to RunPoint state per event. So after the stream signals its event, it is still in the RunPoint state (playing mode) (Fig. 16 (b)). Since the realization for backward presentation is also considered, there is another event (also state) for the backward presentation. In this sense, each realization event adds two states. One is used for forward presentation and the other is used for the backward presentation. The following is a PROMELA code for playing a stream.

```
1   proctype playStream (byte stream) {
2   #if (FC==3 || FC==4 || FC==5 || FC==6)
3   progressIdleStreams:
4   #endif
5   do
6   #if FC!=4
7      ::  atomic{ (eventHandled && getState() == RUN) &&
8           (getStream(stream) == InitPoint) ->
9           printf("Starting stream %d",stream);
```

```
10          setStream(stream, RunPoint);
11          if
12          ::  (stream==A1)->timeIndex=1;
13          ::  else -> skip;
14          fi; }
15      ::  atomic{ (eventHandled && getState() == RUN) &&
16          (getStream(stream) == RunPoint) ->
17          printf("Playing stream %d",stream);
18          setStream(stream, EndPoint);}
19      ::  atomic{ (eventHandled && getState() == RUN) &&
20          (getStream(stream) == EndPoint) ->
21          to_end:  printf("Ending stream %d",stream);
22                  setStream(IdlePoint);
23                  signalEvent(stream,EndPoint) }
24  #endif
25  #if (FC!=3 && FC!=5 && FC!=6)
26      ::  atomic{ (eventHandled && getState() == BACKWARD) &&
27          (getStream(stream) == InitPoint) ->
28          to_init:  printf("Ending backwarding stream %d",stream);
29                  setStream(IdlePoint);
30                  signalEvent(stream,InitPoint);}
31      ::  atomic{ (eventHandled && getState() == BACKWARD) &&
32          (getStream(stream) == RunPoint) ->
33          printf("Playing stream %d backward",stream);
34          setStream(stream, InitPoint);}
35      ::  atomic{(eventHandled && getState() == BACKWARD) &&
36          (getStream(stream) == EndPoint) ->
37          to_backward:  printf("Backwarding stream %d",stream);
38                  if
39                  ::  (stream==A1)->timeIndex=1;
40                  ::  else -> skip;
41                  fi;
42                  signalEvent(stream,EndPoint);
43                  setStream(stream,RunPoint); }
44  #endif
45      ::  atomic{ (eventHandled && getState() == QUIT) ->
46          to_playStream_quit:  goto playStream_quit;}
47      ::  else -> skip;
```

```
48      od;
49      playStream_quit:  skip;
50  }
```

The #if directives are used for hard-coded fairness constraints. There are three states for forward and backward presentation. The cases at lines 7, 14 and 15 correspond to forward presentation. The cases at lines 26, 31 and 35 correspond to the backward presentation. The case at line 45 is required to quit the process. The else statement at line 49 corresponds to IdlePoint state. Streams signal events as they reach to the beginning and end (lines 23 and 30). The variable *eventHandled* is used to check whether the system enters a consistent state after an user interaction. The *atomic* command enables execution of statements in a single step thus reduces the complexity and increases safety. The checking and updating the stream state has to be performed in a single step since the stream state may also be updated by the system after an user interaction. Labels like *to_init, to_end, playStream_quit* are added to create LTL formulas related with these points of the presentation. PROMELA may proceed to any state separated with :: in a *do* loop.

5.3 Receivers, Controllers and Actors

A receiver is set when it receives its event. As long as there is no user interaction, a receiver stays at its state for the rest of the presentation. Thus a controller that requires a receipt of this event can be satisfied later. When a controller is satisfied, it activates its actors. And to disable the reactivation of the actors, the controller is reset. An actor is either idle or it is in its running state meaning that it is sleeping to start its action. Once it wakes up, it starts its action and enters the idle state.

5.4 User and User Interface

The user interface provides seven buttons: START, PLAY, PAUSE, FORWARD, BACKWARD, SKIP, and QUIT. Each button may enter two states in the model. A button is either in enabled or disabled state. As the presentation changes states, the states of the buttons may change. Fig. 15 shows the possible state transitions with enabled user interactions. The user interface is based on the model presented at [8].

For example, if a skip is requested to the mid (12 seconds) of the presentation that is shown in Fig. 3, the timeline is followed in Fig. 8. Receivers R2, R3, R4, R5, R6, R7, R8, R9, and R10 are assumed to receive their events. Receivers R11, R12, and R13 are assumed that they did not receive their events. Controllers C2, C3, C4, C5, C6, C7, C8, C9, and C10 are assumed to be satisfied. C11 and C12 are assumed to be not

satisfied. A satisfied controller cannot notify its controllers. It is assumed that it already notified its actors. At the middle point, there is no sleeping actor. The actors A11 and A12 are activated. So, all the actors should be set to their original time. MAIN container should be set to running point. V1, T1, V2, and A1 should be idle. PAR1 should be idle too. PAR2 should be set to its InitPoint so that it can start the streams that it contains. V3 and A2 should also be set to the IdlePoint.

There are infinite number of skips that can be performed by the user. The timeline shown in Fig. 8 is divided into pieces where the streams perform similar behavior in each piece. There are 21 pieces which are determined by starting and ending actors and actions. So, it is only possible to perform 21 skips.

On the other hand, the backward modifies the direction of the presentation. The synchronization model needs to synchronize after changing direction since streams may proceed at different speeds. To synchronize, the time at that instant should be known. We define a time index which is initially 0 and can be maximum the number of pieces. Some specific streams are allowed to increase or decrease after the time index and the current time index can be determined (lines 12,19). The necessary actors, actions, receivers, and controllers are set and reset after changing the direction.

6 Specification and Verification

6.1 Specification

Specification consists of the properties that a model should satisfy once the model enters some specific states. SPIN automatically checks whether the elements like user, streams, and containers reach their possible states. If not this is reported by the tool.

Fairness Constraint 1 *The user is only allowed to click START button and clicks START button and no user Interaction is allowed after that. This constraint is expressed as:* $\Box\ (userStart \rightarrow \Diamond noInteraction)$

Fairness Constraint 2 *The user always clicks enabled button. This is expressed as* $\Box\ (userClickButton \rightarrow buttonEnabled)$

If a property is stated as undesirable, the system should not allow it. We first start with the properties about transitions that are allowed by buttons.

Property 1 *Clicking button for START enables buttons for PAUSE, FORWARD, and BACKWARD, and it changes the simulations state to RUN. (requires fairness constraint 2)*

Property 2 *Clicking button for PAUSE enables buttons for PLAY, FORWARD, and BACKWARD and it changes the presentation's state to PAUSED. (requires fairness constraint 2)*

Property 3 *The buttons for BACKWARD and SKIP are enabled, and the buttons for START, PLAY, PAUSE, and FORWARD are disabled after the presentation reaches its end. (requires fairness constraint 2)*

Property 4 *The user interface should ignore if the user clicks a disabled button. (requires fairness constraint 2)*

Property 5 *The button for PAUSE is enabled only when the presentation is in RUN, FORWARD, or BACKWARD states. (requires fairness constraint 2)*

Property 6 *The button for SKIP is enabled when the presentation is in RUN, FORWARD, BACKWARD, INITIAL, or END states. (requires fairness constraint 2)*

Property 7 *The buttons for PLAY, FORWARD, and BACKWARD are enabled when the presentation is in PAUSED state. (requires fairness constraint 2)*

Property 8 *The button for START is enabled only when the presentation is in IDLE state. (requires fairness constraint 2)*

Property 9 *The button for PAUSE is enabled outside RUN, FORWARD, and BACKWARD. (undesirable, requires fairness constraint 2)*

Property 10 *Buttons for START, PAUSE, PLAY, FORWARD, and BACKWARD are in enabled condition at any particular time. (undesirable, requires fairness constraint 2)*

Some refinements are needed to convert the properties to LTL formulas. In the following formulas, *actionButtonClicked* corresponds to successful clicking *Button* when the button is enabled. *actionToState* corresponds to state transition to *State*. ButtonEnabled corresponds to *Button* is enabled. *UserButton* corresponds to clicking of *Button* by the user. The operators $\Box$, $\Diamond$, and **U** correspond to *globally, eventually,* and *unless*, respectively. Some of the specification patterns are presented in [10, 25]. These specification patterns can be used in the verification. For each property, the following LTL formulas are generated.

LTL 1 $\Box\ (actionStartClicked \rightarrow\ \Diamond\ actionToRun)$

LTL 2 $\Box\ (actionPauseClicked \rightarrow\ \Diamond\ actionToPaused)$.

LTL 3 $\Box\ (backwardEnabled \wedge skipEnabled \wedge !startEnabled \wedge !playEnabled \wedge !pauseEnabled \wedge !forwardEnabled \rightarrow stateEnd)$

LTL 4 *1.* $\Box\ ((userStart \wedge !startEnabled) \rightarrow !eventHandled\ \mathbf{U}\ userInterfaceIgnore)$

2. $\Box\ ((userPause \wedge !pauseEnabled) \rightarrow !eventHandled\ \mathbf{U}\ userInterfaceIgnore)$

3. $\Box\ ((userPlay \wedge !playEnabled) \rightarrow !eventHandled\ \mathbf{U}\ userInterfaceIgnore)$

4. $\Box\ ((userForward \wedge !forwardEnabled) \rightarrow !eventHandled\ \mathbf{U}\ userInterfaceIgnore)$

5. $\Box\ ((userBackward \wedge !backwardEnabled) \rightarrow !eventHandled\ \mathbf{U}\ userInterfaceIgnore)$

6. $\Box\ ((userSkip \wedge !skipEnabled) \rightarrow !eventHandled\ \mathbf{U}\ userInterfaceIgnore)$

7. $\Box\ ((userQuit \wedge !quitEnabled) \rightarrow !eventHandled\ \mathbf{U}\ userInterfaceIgnore)$

LTL 5 $\Box\ ((stateInitial \vee stateEnd \vee statePaused \vee stateIdle) \rightarrow !pauseEnabled)$

LTL 6 $\Box\ ((stateRun \vee stateForward \vee stateBackward \vee stateEnd) \rightarrow skipEnabled)$

LTL 7 $\Box\ (statePaused \rightarrow (playEnabled \wedge forwardEnabled \wedge backwardEnabled)$

LTL 8 $\Box\ ((stateRun \vee stateEnd \vee statePaused \vee stateForward \vee stateBackward \vee stateInitial) \rightarrow !startEnabled)$

LTL 9 $\Box\ ((stateInitial \vee stateEnd) \rightarrow !pauseEnabled)$

LTL 10 $\Box\ (startEnabled \wedge pauseEnabled \wedge playEnabled \wedge forwardEnabled \wedge backwardEnabled)$

A liveness property that should be checked whether the presentation reaches to its end once it starts.

Property 11 *The presentation will eventually end. (requires fairness constraints 1 and 2)*

LTL 11 $\Box\ (stateRun \rightarrow !\Diamond\ stateEnd)$

There are also some properties that should be satisfied for streams. If a stream is in RunPoint state, the stream cannot be started by other streams. This is assumed to be a wrong attempt. So, a warning should be signaled to the author. In the same way, a stream cannot be terminated if it is already idle. The properties are as follows:

Property 12 *A stream can be started if it is already active. (undesirable, requires fairness constraints 1 and 2)*

Property 13 *A stream can be terminated if it is already idle. (undesirable, requires fairness constraints 1 and 2)*

The LTL formulas are:

LTL 12 $\Diamond\,(streamRunPoint\ \mathbf{U}\ streamInitPoint)$

LTL 13 $\Diamond\,(streamIdlePoint\ \mathbf{U}\ streamEndPoint)$

Further checks can be performed based on the relationships among streams. Based on Allen's temporal relationships, the following properties may be checked:

Property 14 *Stream A is before stream B. (requires fairness constraints 1 and 2)*

Property 15 *Stream A starts with stream B. (requires fairness constraints 1 and 2)*

Property 16 *Stream A ends with stream B. (requires fairness constraints 1 and 2)*

Property 17 *Stream A is equal to stream B. (requires fairness constraints 1 and 2)*

Property 18 *Stream B is not during stream A. (undesirable, requires fairness constraints 1 and 2)*

Property 19 *Stream B does not overlap stream A. (undesirable, requires fairness constraints 1 and 2)*

Let $P = streamA\_InitState$, $Q = streamA\_EndState$, $R = streamA\_IdleState$, $K = streamB\_InitState$, $L = streamB\_EndState$, $M = streamB\_IdleState$. The LTL formulas are as follows:

LTL 14 $(Q\ \mathbf{U}\ (R\ \wedge\ M)\ \mathbf{U}\ K)$

LTL 15 $\Diamond(P\ \wedge\ K)$

LTL 16 $\Diamond(Q\ \wedge\ L)$

LTL 17 $\Diamond(P\ \wedge\ K\ \wedge\ \Diamond\,(Q\ \wedge\ L))$

LTL 18 $!(\Diamond(P\ \wedge\ \Diamond K)\ \vee\ \Diamond\,(Q\ \wedge\ \Diamond L))$

LTL 19 $!(\Diamond(Q\ \wedge\ \Diamond K)\ \vee\ \Diamond\,(L\ \wedge\ \Diamond Q))$

In [23], some properties between two consecutive user interactions based on time are verified. In a distributed system, these constraints cannot be satisfied due to delay of data. 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 our model, user cannot change the state of a stream directly but he/she can change the state of the presentation thus changing the state of a stream indirectly. Since there are relationships among streams and containers, these can start and end each other. In our case, time is associated with actors. Since there is no delay in passing of time, the actor elapses its time right away once it is activated.

Since SPIN does not support time explicitly, time proceeds in the presentation as the streams proceed. In other words, if none of the streams can be played, it is assumed that time does not proceed. The streams update the time index as they start, end and are actually played. There are 21 (from 0 to 20) time indices for Figure 8. For example, stream V2 starts at time index 10 which corresponds to 9 seconds. A stream only updates the time index, if the current index at that instant is lower than its time index. Otherwise, it means that there is a delay in the play of the stream. For example, we would like to check V2 starts 3 seconds after V1 ends. The time index when V1 ends and V2 starts are t_1 and t_2. A *time* array is kept to map time indices to real times. We need to check $(time[t_2] - time[t_1])$ when V2 starts. This is added as *timeCondition* in the formulas.

Property 20 *Stream A is t seconds before stream B (requires fairness constraints 1 and 2).*

Property 21 *Stream B starts t seconds after stream A starts(requires fairness constraints 1 and 2).*

Property 22 *Stream B ends t seconds after stream A ends (requires fairness constraints 1 and 2).*

The corresponding LTL formulas are as follows:

LTL 20 $(Q\ \mathbf{U}\ (R\ \wedge\ M)\ \mathbf{U}\ (K \wedge timeCondition))$

LTL 21 $\Diamond(P\ \wedge\ (K \wedge timeCondition))$

LTL 22 $\Diamond(Q\ \wedge\ (L \wedge timeCondition))$

Note that delay may always occur. So it is meaningless to check that A2 starts 10 seconds after A1 ends. However, it is more meningful to check that A2 starts 1 second after V3 starts since there is no delay in time.

One of the basic queries is whether all streams are played or not. If one of the streams is not played, this may be considered as an undesired behavior and the author may correct it.

Property 23 *Stream A is played. (requires fairness constraints 1 and 2)*

LTL 23 $\Diamond(P \wedge \Diamond Q)$

For a multimedia presentation, the states of streams that are possible to visit in the backward presentation should also be reachable in the forward presentation. We call this property as *backward consistency* of a presentation and term such a presentation as *backward consistent* presentation. If we show the existence of a state that is not reachable in forward presentation while it is reachable in backward presentation, it is not backward consistent.

There are a couple of ways writing the LTL formula to check the backward consistency of a presentation. The major problem is the state that is reachable in the forward presentation may also be given (if exists) as a contradictory example. This complicates the verification since we also need to distinguish the states that are reachable in the forward presentation. Another problem is that the presentation may enter in an inconsistent state after backward operation and from that inconsistent state, the desired state may be reachable in the forward presentation. So, the property is stated as two fold.

Property 24
1. *The state is reachable in forward presentation (undesirable, requires fairness constraints 1 and 2)*
2. *It is possible to reach the state in the backward presentation. (requires fairness constraint 2)*

Notice that first part requires the existence check. The corresponding LTL formulas are as follows:

LTL 24
1. $!\Diamond state$
2. $\Box !state$

If the first part is wrong, then the second part is verified. 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. Eventually, we need to convert the following property into the following one:

Property 25
1. *The state is reachable in forward presentation (undesirable, requires fairness constraints 1 and 2)*
2. *It is possible to reach the state after user interactions. (requires fairness constraint 2)*

The previous LTL formula, in fact, corresponds to this property. Fig. 17 shows the user interface for verification.

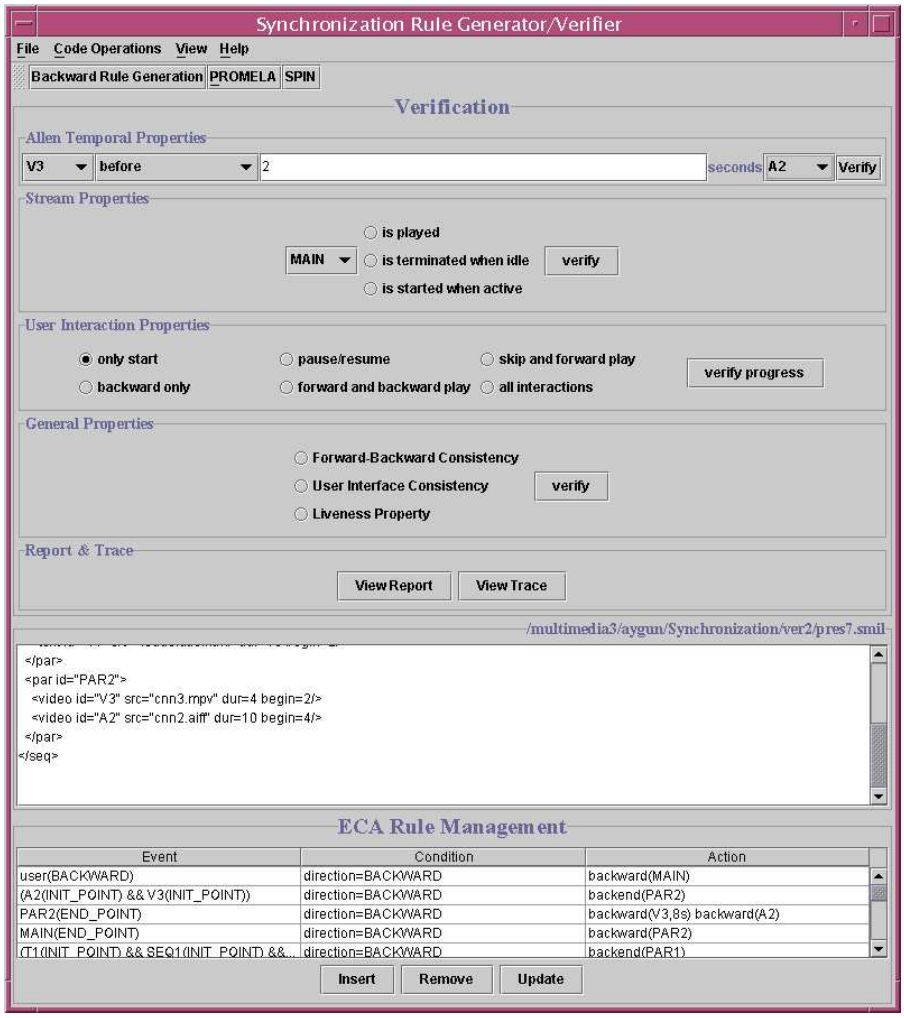


Figure 17: The user interface for verification.

6.2 Analysis

Different kinds of presentations have been used to check the correctness of the presentation model. We considered the number of streams and their organization. The streams are presented in a sequential order or in parallel. If the streams are presented in parallel, they may also be presented in a synchronized fashion.

The fairness constraints are hard-coded in the presentation. For each interaction, there is a fairness constraint and these are hard-coded in the model (lines 2,6,25). FC==3, FC==4, FC==5, FC==6 and FC==7 correspond to interactions where only start, only backward, pause-resume, skip, and backward-play

Presentation	No of Streams	Depth	States	Transitions	Memory (in Mbytes)	Time (in seconds)
single	1	67	177	306	1.5	0.03
sequential	2	99	432	865	1.5	0.04
sequential	3	143	1021	2321	1.6	0.08
sequential	4	209	2347	5868	2.0	0.25
parallel	2	101	488	1021	1.5	0.05
parallel	3	139	1699	4642	1.8	0.13
parallel	4	173	6678	26132	2.8	0.76
synchronized	2	73	185	334	1.5	0.03
synchronized	3	78	201	398	1.5	0.05
synchronized	4	83	233	542	1.5	0.04

Table 1: Experiments without interaction.

are allowed, respectively.

We first investigated the complexity of the number of streams and the organization when no interaction (except to start the presentation) is allowed. The results are given in Table 1.

When a new stream is added into the sequential presentation, there are phases where the new stream starts, plays, and ends. The ending of stream does not add any complexity since they will all be idle at the end of the presentation. Since each stream adds 3 more phases, the number of states is nearly tripled after each addition of a stream in a sequential presentation. The complexity of number of states is $O(m^n)$ where n is the number streams in the sequential presentation and m is one less than the number of states that a stream may enter (to exclude idle state). In our experiments, m is 3. The running time and the depth also increases in the same way.

For a parallel presentation, there are more combinations of playing streams. In the parallel presentations, the streams may be interleaved. The number of possible interleavings for n streams that have m states is

$$I(n,m) = \frac{(2n)!}{(m!)^n}. \tag{1}$$

This explains the steep increase in running time, memory, states, transitions, and depth. Nevertheless, the running time is still within a second for 4 streams. The verification can be performed for a presentation having a fair number of parallel presentations. On the other hand, a synchronized parallel presentation's

Presentation	No of Streams	Depth	States	Transitions	Memory (Mbyte)	Time (in seconds)
single	1	94	279	487	1.5	0.04
sequential	2	168	718	1435	1.6	0.06
sequential	3	304	1814	4060	1.8	0.12
sequential	4	559	4564	11341	2.5	0.36
parallel	2	178	829	1810	1.6	0.07
parallel	3	258	3519	11174	2.1	0.32
parallel	4	423	22913	101443	6.1	2.76
synchronized	2	106	287	517	1.5	0.03
synchronized	3	111	303	591	1.5	0.04
synchronized	4	122	335	749	1.5	0.06

Table 2: Experiments with Pause-Resume

complexity is $O(n)$ for depth, transitions, and running time but $O(2^n)$ for the states.

To evaluate the effect of user interactions, we tested user interactions separately. The experiments with pause-resume interactions are given in Table 2. The pause-resume interactions increase the complexity in linear time. Therefore, the presentations having pause and resume interactions do not add more complexity and this is an expected result. But these interactions increased the initial number of states, depth, and complexity.

The skip interaction increases the time complexity more than pause-resume due to possible number of skips at each interaction. The time complexity of synchronized presentations is $O(2^n)$. On the other hand, the complexity of states for parallel presentations increased from $O(4^n)$ to (10^n). The complexity of states for sequential presentations increased from $O(3^n)$ to $O(4^n)$.

The play interaction is allowed along with the backward interaction. The time complexity of synchronized presentations is $O(2^n)$. On the other hand, the complexity of states for parallel presentations increased from $O(4^n)$ to (10^n). The complexity of states for sequential presentations increased from $O(3^n)$ to $O(4^n)$. These results show that the complexity of backward is similar to the skip. Since the direction of the presentation may change in the backward presentation, the number of initial states doubled and this caused severe exponential increase in the running time.

These verification results give an upper bound on the verification of properties. These results consider the non-progress cycles and other possible errors. Verification of properties take less since only a specific

condition is checked in the model. For the presentation given in Fig. 3, the verification for Allen's temporal properties takes less than 3 seconds.

7 Conclusion

The RuleSync synchronization model is developed to support the NetMedia [32] system, a middleware design strategy for streaming multimedia presentations in distributed environments. The synchronization is handled by synchronization rules based on event-condition-action (ECA) rules. The backward rules are generated automatically based on *author properties* and forward presentation. In this report, not only forward temporal relationships are converted to reverse temporal relationships but also the relationships between events and actions are considered for backward presentation. The model checking technique is used for verification of the model. This technique is better than testing and simulation since all the states that a model may enter is considered. The system may be verified whether it conforms to the specification. If a property is not satisfied, a counter is provided by the SPIN tool. There have been methods proposed for temporal querying of presentations. But it is not tested whether the model really satisfies the author specification. The PROMELA code also supports delayed presentation. Thus it is possible whether the system satisfies the specification. In the report, we also considered satisfaction of Allen's temporal relationships.

There are also limitations that are put forward by the model checking. For example, it is not possible to check the *meet* relationship using SPIN. There is more than one state between ending a stream and starting a new stream. This is also true in real applications. The major limitation of SPIN is that it does not support time explicitly. It is left to the programmer how they handle time. As mentioned in the report, this can be performed at a level.

During the real time presentation those constraints that are checked at the specification level may not be satisfied in a real time environment. The specification may in fact lead to false presentations due to delay and the synchronization model. This model enables the author to check whether the system really satisfies the requirements in the real life. Hence, the author may better specify the presentation using this model checking since extra-ordinary conditions are considered. As a future work, the system may provide a better synchronization specification to satisfy the violated constraints.

References

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

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

[4] R. S. 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. S. 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] CMIS. http://www.cis.ksu.edu/ robby/classes/spring1999/842/index.html.

[9] 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.

[10] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of 21st International Conference on Software Engineering*, May 1999.

[11] 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.

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

[13] 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.

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

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

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] S. Marcus and V. Subrahmanian. Foundations of multimedia database systems. *Journal of the ACM*, 43(3):474–523, 1996.

[22] 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.

[23] 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.

[24] 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.

[25] D. O. Paun and M. Chechik. Events in linear-time properties. In *Proceedings of 4th International symposium on Requirements Engineering*, June 1999.

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

[27] 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.

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

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

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

[31] 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.

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