A Compositional Approach to Multimedia Documents Dynamics

P. Bertolotti
Università di Torino
bertolot@di.unito.it

O. Gaggi
Università Ca' Foscari di Venezia
ogaggi@dsi.unive.it

M.L. Sapino
Università di Torino
mlsapino@di.unito.it

V.S. Alagar
Concordia University
alagar@cs.concordia.ca

1 Introduction

Multimedia presentations can be described as collections of media items coherently synchronized and presented to the user. Each media item is an independent atomic component with its own behavior. Depending on their types, distinct media may exhibit different behavior. For example, it is possible for a user to stop or pause an already started video stream (i.e. a *dynamic* object), before it naturally ends. Otherwise, a *static* object, like a text page or an image, once displayed, remains on the screen until the user stops it.

In [4], Celentano et al address the problem of retrieving fragments of multimedia presentations modelled according to [7], that have to be consistent with the synchronization relationships defined by the authors of the presentation. To do so, they define an automaton, which formally describes the presentation states entered by the events which trigger media playback. A *state* contains at any time instant, the set of media that are active at that time, according to the set of synchronization relationships defined by the author, and the corresponding channel occupation. Before the presentation starts, no media item is active, thus all channels are free. When an event occurs, the state of the presentation changes: some items that were not active become active, some active items end, other items could be forced to stop due to some interruption.

In this paper, we are still interested in modelling the presentation behavior, but at a different granularity level, and with different goals. Specifically, we aim at describing the behavior of the single media appearing in any given presentation, by means of finite state machines, and modelling the behavior of complex *dynamic systems* by composition of their single atomic components. This solution makes a step further to a fine grain modelling of a multimedia presentation and considers many aspects relating the actual execution of a multimedia document that are not detailed in the previous work [4].

The model we are discussing in this paper associates each single media item with an independent finite state machine, which describes its evolution, from its activation, to its end or stop. Different states correspond to different phases of the media execution. If we assume to deal with a distributed environment, like the World Wide Web, before its playback, a media item must be downloaded to the local machine[1] and buffered. We call these activities *pre-fetching* of a media component.

State transitions are triggered by specific external (that is, user observable) *events*, like users' requests to start or stop the media, and by internal (non observable) events, mainly capturing modification in the buffer occupation. In the presence of those events, transitions are fired provided some *conditions*, expressed in terms of logical predicates, hold.

The model presented in this paper is a general approach that can be applied to several problems relating the modelling of dynamic systems, like for example scheduling problems, estimation of time needed to pre-fetch complex component of a multimedia document, etc. The model is also suited for formal verification [5] of the coherence and correctness of synchronization constraints designed by the author.

2 Multimedia Presentations

For modelling multimedia presentations we refer to the synchronization model previously defined in [4, 7]. The model is oriented to web-based hypermedia presentations, and describes the structure of a presentation and the temporal behavior of its components. In [4] a complete and detailed description of the model can be found. We discuss here only the issues relevant to scope of this paper.

A multimedia presentation is a 4–*tuple* $P = \langle \mathcal{MI}, \mathcal{CH}, \mathcal{E}, \mathcal{SR} \rangle$ where $\mathcal{MI}$ is a set of media items which build the presentation, $\mathcal{CH}$ is a set of channels, i.e. virtual devices used to reproduce media components and mapped to actual resources during their playback, $\mathcal{E}$ is a set of events which will be detailed in Section 3, and $\mathcal{SR}$ is a set of temporal relationships which describe the presentation behavior.

Therefore, an author can design the presentation evolution by imposing a set of temporal constraints among the objects. The proposed model defined five synchronization primitives:

- *a plays with b* (written $a \Leftrightarrow b$), which models the parallel composition of two objects,
- *a activates b* (written $a \Rightarrow b$), which models the sequential composition of two objects,
- *a is replaced by b* (written $a \rightleftharpoons b$), which models the substitution of media item a with b in the same channel,
- *a terminates b* (written $a \Downarrow b$), which models the simultaneous forced stop of media item b when object a is forced to stop by the user or some other external event and
- *a has priority over b with behavior α* (written $a \overset{\alpha}{>} b$), which can be used to design presentation behavior during user interactions. Media item b is paused ($\alpha = p$) or stopped ($\alpha = s$) when the user starts object b.

[1]This step can be omitted using the RTSP[12] as transfer protocol, or any other protocol implementing the *streaming* technology.

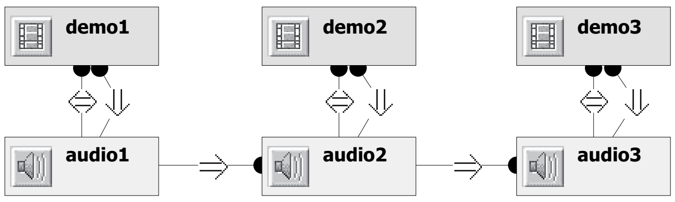

Figure 1: Synchronization schema of a training multimedia presentation

As an example, let us consider a multimedia presentation design to train users to deal with a new program. The multimedia presentation is composed by a set of animations, which reproduce demos of the program at work, while an audio comment explains what the user has to do.

Figure 1 depicts a portion of the synchronization schema of this presentation where $audio_i$ and $demo_i$ for $i = 1 \dots 3$ are the animations and the corresponding three audio comments which explain how to use the program, respectively. The comments and the animations use two different channels, an audio channel, called *comments* and a window on the user screen, called *display.*

The relationship *plays with* ($\Leftrightarrow$) models the parallel execution of each audio comment with the associated animation, and the relationship *activates* ($\Rightarrow$) imposes a temporal order on the set of audio files, i.e. first $demo_1$ is displayed during $audio_1$ playback, then $audio_2$ comments $demo_2$ and last, $audio_3$ and $demo_3$ are activated. If the user stops the audio comments, also the active animation must be stopped: this behavior is modelled by relationship *terminates* ($\Downarrow$).

Therefore, $P = \langle \mathcal{MI}, \mathcal{CH}, \mathcal{E}, \mathcal{SR} \rangle$ where:

- $\mathcal{MI} = \{audio_i, demo_i \ \forall i = 1 \dots 3\}$,
- $\mathcal{CH} = \{comment, display\}$,
- $\mathcal{E} = \{e_m\}$, where $m \in \mathcal{MI}$ and $e\_$ describes the type of event which will be discussed in Section 3 and
- $\mathcal{SR} = \{audio_i \Leftrightarrow demo_i, audio_i \Downarrow demo_i \ \forall i = 1 \dots 3\} \cup \{audio_i \Rightarrow audio_{i+1} \ i = 1, 2\}$.

P describes the static structure of a multimedia presentation, i.e. the media components, the user interface organization, defined by the channels definition, and the temporal constraints among the media items.

Then each item has its internal behavior, e.g a static medium is displayed on the screen, a continuous media components starts its playback and can be paused or stopped before its natural end. This behavior can be formally described with a finite state machine, which shows media reactions to a particular event. Therefore, the presentation evolution can be described by composing several finite state machines representing the behavior of single items.

3 A formal model for complex multimedia documents

We introduce an independent finite state machine modelling a single media object, that encapsulates the functional and timing properties of the media item.

A media item can be in six different *states*:

- **idle**: all objects which are not active are in this state, waiting to be activated. Each media item, when it naturally ends, becomes idle again;
- **init**: this state corresponds to the time used to pre-fetch the media;
- **playing**: the actual playback of the object;
- **paused**: when the media item is paused;
- **stopped**: when the media item is forced to terminate by the user or some other external event;
- **terminating**: this state describes the situation in which the object is terminating. We consider distributed presentations in which the media involved have to be transferred to be played. This usually requires bufferization. Then we can consider a media object of size S, divided into N segments (whose length is equal to the size of the buffer associated with the object), the media enters the state "terminating" when it begins playing the last segment.

For the sake of simplicity, in this characterization we do not explicitly distinguish between *continuous* media and *static* media. For static media, states *terminating* and *paused* are meaningless, and in this case *playing* is intended to model the fact of being displayed.

Another simplification concerns our modelling of bufferization. In this preliminary work, we do not capture information about different buffer sizes and timings. We instead abstract from single buffer details, and work under the hypothesis that every media is associated with a specific buffer, and that the relevant information about that buffer only concern its being empty, full, or partially filled.

The association of distinct media items with their buffers, is expressed by means of the function *buffer()*, having the considered medium as its argument. Analogously, we also assume that every medium is associated to a playback channel, and a stream of data. To denote this association we use the functions *channel()*, and *stream()*, having the medium as their argument. The relevant information to be checked, when a single medium is modelled, is the status of its buffer, its channel and its stream. To check the status of buffers and streams we use the predicates *isEmpty()*, and *isFull()*. The status of channels is checked by means of the predicate *isFree()*.

We denote the set of events that can cause a state transition with $\mathcal{E}$; it includes: **start** (when a user asks for the activation of a medium), **ready** (when the buffer of a starting media is full), **pause** (when a medium playout is temporally interrupted), **stop** (when a user forces the termination of a medium playout), **ending** (when the last segment of the media is starting playing) and **end** (when a medium playout reaches its natural termination).

In the above list of possible events, we distinguish between *external* and *internal* events. External events are those that have an effect immediately perceived by the user. For example, *start*, that corresponds to the user's action of clicking the bottom to activate a medium, is an external event as well as *stop*, which corresponds to the user's request of interrupting a medium playout, *pause* and *end*.

Internal events correspond to some modification in the internal state of the system, that the user is not necessarily aware of. This is the case with *ready*, which represents the fact that the buffer associated with a medium is full, which makes it possible to effectively activate the playout, and *ending*, which indicates that the buffer associated with the object is filled for the last time, then the item is finishing.

Definition 3.1 (Single Item Finite State Machine) The finite state machine characterizing a given media item m_i is $MSM(m_i) = \langle S, s_0, F, next, T \rangle$, where

- $S = \{\mathbf{idle}, \mathbf{init}, \mathbf{playing}, \mathbf{paused}, \mathbf{stopped}, \mathbf{terminating}\}$;
- $s_0 = \mathbf{idle}$;
- $F = \{\mathbf{idle}, \mathbf{stopped}\}$;
- *next* is the function defined as follows:

next(**idle**, start) = **init**	next(**init**, ready) = **playing**
next(**init**, pause) = **paused**	next(**init**, stop) = **stopped**
next(**playing**, ending) = **terminating**	next(**playing**, stop) = **stopped**
next(**playing**, pause) = **paused**	next(**terminating**, end) = **idle**
next(**terminating**, stop) = **stopped**	next(**terminating**, pause) = **paused**
next(**paused**, stop) = **stopped**	next(**paused**, start) = **playing**
next(**stopped**, start) = **init**	

- T is a set of 4-tuples $\langle s, e, C, P \rangle$ describing transitions, where:
  - $s \in S$ is the initial state;
  - $e \in \mathcal{E}$ is an event;
  - $\mathcal{C}$ is a set of enabling conditions for the transition from state s when the event e occurs;
  - $\mathcal{P}$ is a set of postconditions, that is, conditions holding after the transition takes place.

  In the following, to characterize state transitions we use the following notation, where $state_i$ is the initial state and $state_r$ is the resulting state.

$$[\mathcal{C}] \quad \mathbf{state_i} \xrightarrow{e} \mathbf{state_r} \quad [\mathcal{P}]$$

State transitions take place when an event occurs, and their enabling conditions are satisfied. Preconditions and postconditions mentioned in our transitions only concern local predicates, i.e., predicates whose truth value might be affected by the firing transition.

$[isFree(channel(m_i))]$	**idle** $\overset{start}{\rightarrow}$ **init**	$[\neg isEmpty(buffer(m_i))\wedge$ $\neg isFree(channel(m_i))]$
$[isFull(buffer(m_i))]$	**init** $\overset{ready}{\rightarrow}$ **playing**	$[true]$
$[true]$	**init** $\overset{pause}{\rightarrow}$ **paused**	$[true]$
$[true]$	**init** $\overset{stop}{\rightarrow}$ **stopped**	$[isEmpty(buffer(m_i))\wedge$ $isFree(channel(m_i))]$
$[isEmpty(stream(m_i))]$	**playing** $\overset{ending}{\rightarrow}$ **terminating**	$[true]$
$[true]$	**playing** $\overset{stop}{\rightarrow}$ **stopped**	$[isEmpty(buffer(m_i))\wedge$ $isFree(channel(m_i))]$
$[isEmpty(buffer(m_i))]$	**terminating** $\overset{end}{\rightarrow}$ **idle**	$[isEmpty(buffer(m_i))\wedge$ $isFree(channel(m_i))]$
$[true]$	**playing** $\overset{pause}{\rightarrow}$ **paused**	$[true]$
$[true]$	**paused** $\overset{stop}{\rightarrow}$ **stopped**	$[isEmpty(buffer(m_i))\wedge$ $isFree(channel(m_i)]$
$[true]$	**terminating** $\overset{stop}{\rightarrow}$ **stopped**	$[isEmpty(buffer(m_i))\wedge$ $isFree(channel(m_i)]$
$[isFree(channel(m_i))]$	**paused** $\overset{start}{\rightarrow}$ **playing**	$[\neg isFree(channel(m_i))]$
$[true]$	**terminating** $\overset{pause}{\rightarrow}$ **paused**	$[true]$
$[isFree(channel(m_i))]$	**stopped** $\overset{start}{\rightarrow}$ **init**	$[\neg isFree(channel(m_i))\wedge$ $\neg isEmpty(buffer(m_i))]$

Table 1: Set of transition rules for single media items

The set of transitions characterizing a single media item is shown in the following Table 1.
The model we have introduced so far characterizes the behavior of a single media. Given this represen-

tation, we can model a presentation which contains several media objects, by composing the corresponding finite state machines. Several unrelated media may exist in the presentation, therefore we first model a system simply containing a number of independent media. Then, we specialize some transition rules, to model synchronization primitives. In the following definition, we will make use of footers to distinguish different media, and use the footer corresponding to each medium also to refer to its states and events, thus distinguishing between analogous states and events for different media.

We will denote with $\mathcal{C}_{s_i,e_i}$ the enabling condition for the transition corresponding to event e_i in the state s_i, for the medium m_i. Analogously for $\mathcal{P}_{s_i,e_i}$.

Definition 3.2 Let $m_1, \ldots, m_n$ be n independent (i.e. not related to one another by means of synchronization rules) media items, and $MSM_1, \ldots, MSM_n$ be the corresponding finite state machines. Let $MSM_i = \langle S_i, s_i^0, F_i, next_i, T_i \rangle$, for all $i = 1, \ldots, n$.

The overall behavior is modelled by the finite state machine $MSM = \langle S, s^0, F, next, T \rangle$, where

- $S = \{\langle s_1, \ldots, s_n \rangle \mid s_i \in S_i, i = 1, \ldots, n\}$;
- $s^0 = \langle s_1^0, \ldots, s_n^0 \rangle$;
- $F = \{\langle s_{f_1}, \ldots, s_{f_n} \rangle \mid s_{f_i} \in F_i, i = 1, \ldots, n\}$;
- $next(\langle s_1, \ldots, s_i, \ldots, s_n \rangle, e_{m_i}) = \langle s_1, \ldots, next_i(s_i, e_{m_i}), \ldots, s_n \rangle$, for any $s_i \in S_i$, and any event e_{m_i} on the the medium m_i, $i = 1, \ldots, n$.
- T contains the following transitions:
  $\forall t$, if $t = \langle s_i, e_{m_i}, \mathcal{C}_{s_i,e_{m_i}}, \mathcal{P}_{s_i,e_{m_i}} \rangle \in T_i$ for a given i then
  $\langle \langle s_1, \ldots, s_i, \ldots, s_n \rangle, e_{m_i}, \mathcal{C}_{s_i,e_{m_i}}, \mathcal{P}_{s_i,e_{m_i}} \rangle \in T$.

This composition models n objects which are completely independent from one another. Then we must consider objects which are temporally related by the synchronization relationships described in Section 2. For the sake of simplicity, we here introduce the composition of synchronized media only considering pairs of media to be synchronized. The method can be easily generalized to the case of n media.

We can translate the temporal relations into four different kinds of composition, which are mostly based on the one considered above.

Specifically, the finite state machine modelling any temporal composition $m_i \theta m_j$, $\theta \in \{\Leftrightarrow, \Rightarrow, \Downarrow, \rightleftharpoons, \overset{s}{>}, \overset{p}{>}\}$ is defined by: (i) applying the definition 3.2, to model the case of the general composition of items m_i and m_j, and (ii) adding or overwriting some specific transition rules. Given a transition $t \in T$ for an event e from $state_i$ to $state_r$, if $t' = \langle state_i, e, \mathcal{C}_{state_i,e}, \mathcal{P}_{state_r,e} \rangle \in T$ reaching the same state $state_r$ already exists, t replaces t', otherwise t is added to the set of transitions T.

Tables 2, 3, 4, 3, 6, and 7 list the rewritten rules for $\Leftrightarrow$, $\Rightarrow$, $\Downarrow$, $\rightleftharpoons$, $\overset{s}{>}$, and $\overset{p}{>}$, respectively.

$[\mathcal{C}_{idle_i,start_i} \wedge$ $\mathcal{C}_{idle_j,start_j}]$	$\langle \mathbf{s_i}, \mathbf{s_j} \rangle \overset{start_m}{\rightarrow} \langle \mathbf{init_i}, \mathbf{init_j} \rangle$ for $m \in \{i, j\}$ and $\mathbf{s} \in \{\mathbf{idle}, \mathbf{stopped}\}$	$[\mathcal{P}_{idle_i,start_i} \wedge$ $\mathcal{P}_{idle_j,start_j}]$
$[\mathcal{C}_{init_i,ready_i} \wedge$ $\mathcal{C}_{init_j,ready_j}]$	$\langle \mathbf{init_i}, \mathbf{init_j} \rangle \overset{e}{\rightarrow} \langle \mathbf{playing_i}, \mathbf{playing_j} \rangle$ for all $e \in \{ready_i, ready_j\}$	$[\mathcal{P}_{init_i,ready_i} \wedge$ $\mathcal{P}_{init_j,ready_j}]$
$[\mathcal{C}_{idle_i,start_i} \wedge$ $\neg\ \mathcal{C}_{idle_j,start_j}]$	$\langle \mathbf{s_i}, \mathbf{s_j} \rangle \overset{start_i}{\rightarrow} \langle \mathbf{init_i}, \mathbf{s_j} \rangle$ for all $s_j \in S_j$ and $\mathbf{s_i} \in \{\mathbf{idle_i}, \mathbf{stopped_i}\}$	$[\mathcal{P}_{idle_i,start_i}]$
$[\neg\ \mathcal{C}_{idle_i,start_i} \wedge$ $\mathcal{C}_{idle_j,start_j}]$	$\langle \mathbf{s_i}, \mathbf{s_j} \rangle \overset{start_j}{\rightarrow} \langle \mathbf{s_i}, \mathbf{init_j} \rangle$ for all $s_i \in S_i$ and $\mathbf{s_j} \in \{\mathbf{idle_j}, \mathbf{stopped_j}\}$	$[\mathcal{P}_{idle_j,start_j}]$
$[\mathcal{C}_{idle_i,start_i} \wedge$ $\mathcal{C}_{idle_j,start_j}]$	$\langle \mathbf{s_i}, \mathbf{init_j} \rangle \overset{ready_j}{\rightarrow} \langle \mathbf{init_i}, \mathbf{init_j} \rangle$ for $s_i \in \{\mathbf{idle_i}, \mathbf{stopped_i}\}$	$[\mathcal{P}_{idle_i,start_i} \wedge$ $\mathcal{P}_{idle_j,start_j}]$
$[\mathcal{C}_{idle_i,start_i} \wedge$ $\mathcal{C}_{idle_j,start_j}]$	$\langle \mathbf{init_i}, \mathbf{s_j} \rangle \overset{ready_i}{\rightarrow} \langle \mathbf{init_i}, \mathbf{init_j} \rangle$ for $s_j \in \{\mathbf{idle_j}, \mathbf{stopped_j}\}$	$[\mathcal{P}_{idle_i,start_i} \wedge$ $\mathcal{P}_{idle_j,start_j}]$
$[\mathcal{C}_{terminating_i,end_i}]$	$\langle \mathbf{terminating_i}, \mathbf{s_j} \rangle \overset{end_i}{\rightarrow} \langle \mathbf{idle_i}, \mathbf{stopped_j} \rangle$ for all $\mathbf{s_j} \in \mathbf{S_j} \setminus \{\mathbf{idle_j}, \mathbf{stopped_j}\}$	$[\mathcal{P}_{terminating_i,end_i} \wedge$ $\mathcal{P}_{s_j,stop_j}]$
$[false]$	$\langle \mathbf{terminating_i}, \mathbf{s_j} \rangle \overset{end_i}{\rightarrow} \langle \mathbf{idle_i}, \mathbf{s_j} \rangle$ for all $\mathbf{s_j} \in \mathbf{S_j} \setminus \{\mathbf{idle_j}, \mathbf{stopped_j}\}$	$[\mathcal{P}_{terminating_i,end_i}]$

Table 2: Transition rules for the $m_i \Leftrightarrow m_j$ relationship

$[\mathcal{C}_{playing_i,ending_i} \wedge$ $\mathcal{C}_{idle_j,start_j}]$	$\langle \mathbf{playing_i}, \mathbf{s_j} \rangle \overset{ending_i}{\rightarrow} \langle \mathbf{terminating_i}, \mathbf{init_j} \rangle$ for all $\mathbf{s_j} \in \{\mathbf{idle_j}, \mathbf{stopped_j}\}$	$[\mathcal{P}_{playing_i,ending_i} \wedge$ $\mathcal{P}_{idle_j,start_j}]$
$[\mathcal{C}_{playing_i,ending_i} \wedge$ $\neg\ \mathcal{C}_{idle_j,start_j}]$	$\langle \mathbf{playing_i}, \mathbf{s_j} \rangle \overset{ending_i}{\rightarrow} \langle \mathbf{terminating_i}, \mathbf{s_j} \rangle$ for all $\mathbf{s_j} \in \{\mathbf{idle_j}, \mathbf{stopped_j}\}$	$[\mathcal{P}_{playing_i,ending_i}]$
$[\mathcal{C}_{terminating_i,end_i} \wedge$ $\mathcal{C}_{init_j,ready_j}]$	$\langle \mathbf{terminating_i}, \mathbf{init_j} \rangle \overset{e}{\rightarrow} \langle \mathbf{idle_i}, \mathbf{playing_j} \rangle$ for all $e \in \{end_i, ready_j\}$	$[\mathcal{P}_{terminating_i,end_i}]$

Table 3: Transition rules for the $m_i \Rightarrow m_j$ relationship

$[\mathcal{C}_{s_i,stop_i}]$	$\langle \mathbf{s_i}, \mathbf{s_j} \rangle \overset{stop_i}{\rightarrow} \langle \mathbf{stopped_i}, \mathbf{stopped_j} \rangle$	$[\mathcal{P}_{s_i,stop_i} \wedge \mathcal{P}_{s_j,stop_j}]$
	for all $\mathbf{s_i} \in \mathbf{S_i} \setminus \{\mathbf{idle_i}, \mathbf{stopped_i}\}$	
	for all $\mathbf{s_j} \in \mathbf{S_j} \setminus \{\mathbf{idle_j}, \mathbf{stopped_j}\}$	

Table 4: Transition rules for the $m_i \Downarrow m_j$ relationship

$[\mathcal{C}_{init_j,ready_j}]$	$\langle \mathbf{s_i}, \mathbf{init_j} \rangle \overset{ready_j}{\rightarrow} \langle \mathbf{stopped_i}, \mathbf{playing_j} \rangle$	$[\mathcal{P}_{stopped_i,stop_i} \wedge$
	for all $\mathbf{s_j} \in \mathbf{S_j} \setminus \{\mathbf{idle_j}, \mathbf{stopped_i}\}$	$\mathcal{P}_{init_j,ready_j}]$

Table 5: Transition rules for the $m_i \rightleftharpoons m_j$ relationship

$[\mathcal{C}_{init_i,ready_i}]$	$\langle \mathbf{init_i}, \mathbf{s_j} \rangle \overset{ready_i}{\rightarrow} \langle \mathbf{playing_i}, \mathbf{stopped_j} \rangle$	$[\mathcal{P}_{init_i,ready_i} \wedge$
	for all $\mathbf{s_j} \in \mathbf{S_j} \setminus \{\mathbf{idle_j}, \mathbf{stopped_i}\}$	$\mathcal{P}_{stopped_j,stop_j}]$

Table 6: Transition rules for the $m_i \overset{s}{>} m_j$ relationship

$[\mathcal{C}_{init_i,ready_i}]$	$\langle \mathbf{init_i}, \mathbf{s_j} \rangle \overset{ready_i}{\rightarrow} \langle \mathbf{playing_i}, \mathbf{paused_j} \rangle$	$[\mathcal{P}_{init_i,ready_i}]$
	for all $\mathbf{s_j} \in \mathbf{S_j} \setminus \{\mathbf{idle_j}, \mathbf{stopped_j}\}$	

Table 7: Transition rules for the $m_i \overset{p}{>} m_j$ relationship

Example 3.1 Figure 2 and 3 depict a portion of the automata modelling the parallel and the sequential composition of two media. Figure 2 illustrates the behavior obtained with the relationship $audio_1 \Leftrightarrow demo_1$ (see Figure 1). As the user starts media item $audio_1$[2] the system begins to pre-fetch both media items $audio_1$ and $demo_1$, which step in state **init**. Each media item is associated to a buffer; when it is filled, the object is ready for its playback. The system waits until both the objects are ready, as described by the precondition $\mathcal{C}_{init_{audio_1},ready_{audio_1}} \wedge \mathcal{C}_{init_{demo_1},ready_{demo_1}} = isFull(buffer(audio_1)) \wedge isFull(buffer(demo_1))$. When the last media item is ready[3], $audio_1$ and $demo_1$ begin their playback and step to state **playing**. If $audio_1$ naturally ends, $demo_1$ is stopped, while the natural termination of $demo_1$ does not influence $audio_1$ playback.

Figure 3 describes the behavior obtained with relationship $audio1 \Rightarrow audio2$. Both objects $audio_1$ and $audio_2$ use the same channel, therefore $audio_2$ cannot start during $audio_1$ playback. The system begins the pre-fetching of $audio_2$ when the first audio track is terminating, therefore it steps to state s_3, in which $audio_1$ is in state **terminating** and $audio_2$ is in state **init** as answer to event $ending_{audio_1}$. When $audio_1$ ends, the system waits until $buffer(audio_2)$ is full (precondition $\mathcal{C}_{init_{audio_2},ready_{audio_2}} = isFull(buffer(audio_2)))$ and then starts $audio_2$[4].

[2]If the user starts media item $demo_1$, he or she obtains the same behavior.

[3]In Figure 2, $audio_1$ is assumed to be last media to fill the buffer.

[4]In Figure 3 we assume that $buffer(audio_2)$ is already full when $audio_1$ naturally ends. Otherwise, the system steps to

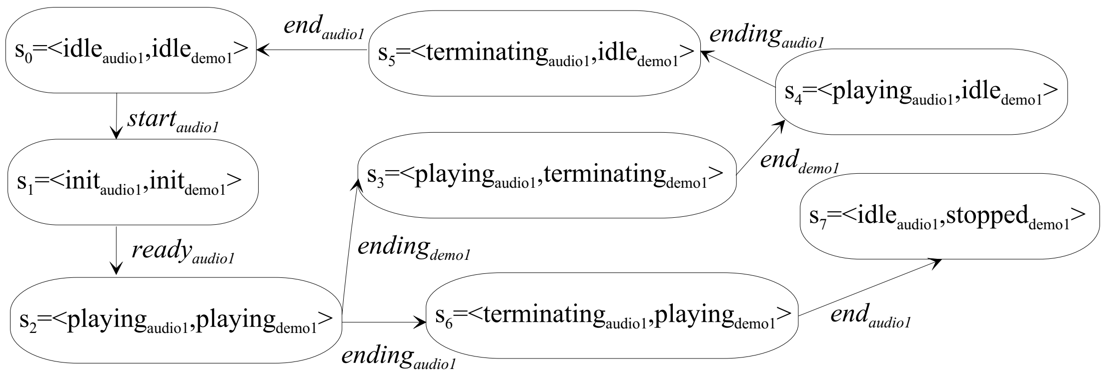
s0=<idle_audio1,idle_demo1>
end_audio1
s5=<terminating_audio1,idle_demo1>
ending_audio1
s4=<playing_audio1,idle_demo1>
start_audio1
s1=<init_audio1,init_demo1>
s3=<playing_audio1,terminating_demo1>
end_demo1
s7=<idle_audio1,stopped_demo1>
ready_audio1
ending_demo1
s2=<playing_audio1,playing_demo1>
s6=<terminating_audio1,playing_demo1>
end_audio1
ending_audio1

Figure 2: The automaton modelling the relationship $audio_1 \Leftrightarrow demo_1$

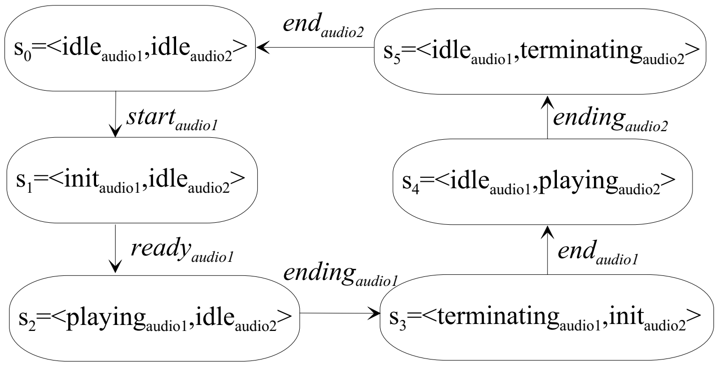
s0=<idle_audio1,idle_audio2>
end_audio2
s5=<idle_audio1,terminating_audio2>
start_audio1
ending_audio2
s1=<init_audio1,idle_audio2>
s4=<idle_audio1,playing_audio2>
ready_audio1
end_audio1
ending_audio1
s2=<playing_audio1,idle_audio2>
s3=<terminating_audio1,init_audio2>

Figure 3: The automaton modelling the relationship $audio_1 \Rightarrow audio_2$

The abstract formal model introduced so far is well suited for reasoning on multimedia documents dynamics, and to prove properties about them.

For example, the finite state machine associated with composite documents allows us to derive the set of single media possibly playing in parallel, or to compute the sets of media that necessarily have to be serialized.

To do so, given a description on the initial state of the overall system, expressed in terms of (positive) predicates on media buffers and channels, and given a sequence of events concerning the involved media items, we inductively reason on the feasibility of that sequence of events, and on the impact that the sequence might have on the overall system, as follows.

Given an event, the corresponding transition can fire if its precondition is true in the current state. If this is the case, the effects of the event are recorded in the new state by (i) deleting, from the current state, those predicate instances which appear negated in the postcondition of the fired transition; (ii) for

state s_4 as answer to event $ready_{audio_2}$.

any positive predicate instance appearing in the postcondition, inserting it in the resulting state. If the predicate instance $p(buffer(m_i))$ is the inserted one, (that is, a fact stating something about the buffer of media item m_i), any other predicate $q(buffer(m_i))$ appearing in the current state (and concerning the same media item) is then removed. This replacement captures the dynamic evolution of the buffer condition.

In the future, we plan to develop a formal system to reason within this model, by properly defining axioms and proof rules, according to the methods usually adopted for program verification and model checking.

4 Related Work

Several synchronization models describe the temporal evolution of a multimedia presentation by defining synchronization relationships among its objects [2, 8, 11, 13]. Candan et al [3], makes a step further, deriving a correct schedule of the media items, which respects the specified temporal relations, and manages different network throughput and buffers resource available.

The models proposed in [6, 10] are mostly based on well known formal models for modeling complex systems. Specifically, in [6] a specification method based on Milner's Calculus of Communicating Systems is proposed, and a visual programming environment based on the specification mechanism is described. In [10] the reference formal model is Petri Net Model.

In both cases, the models are used to capture the system behaviours at the media item granularity level, that is, the models capture the mutual relationships existing among different media items whose composition results in the presentation. No specific attention is devoted to lower level phases needed to play a presentation, such as pre-fetching, buffering, etc. For this reason, we can consider these models as alternative approaches to model media composition, of the same granularity level as the model by Celentano and Gaggi [7] that we choose as our starting point. This last seems more suitable to our purposes, since its five synchronization primitives have an easy to understand semantics, and make the authoring process easier to the user.

On the other hand, different alternative approaches to model complex systems exist in the literature. Among them [1] provides a formal method for specifying the design of reactive objects and systems. The considered systems have the main characteristics of having to react to stimuli from their environment, with strong response synchronization constraints.

In our case, the reaction to the user interaction is not a dominant issue, while we mostly concentrate on the inner dynamics of the single presented media.

Paulo et al. [9] describes a synchronization model based on hypercharts. Hyperchart notation extends the statechart formalism in order to make it able to describe temporal constrains and synchronization requirements of a multimedia presentation. The system performs a single step at each time unit, reacting to all external changes that happen in that time interval.

References

[1] V.S. Alagar R. Achuthan and D. Muthiayen. TROMLAB: an object-oriented framework for real-time reactive systemd development. Technical report, University of Montreal, Canada, 1998.

[2] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Comm. ACM*, 26(11):832–843, November 1983.

[3] K.S. Candan, B. Prabhakaran, and V.S. Subrahmanian. Retrieval Schedules Based on Resource Availability and Flexible Presentation Specifications. *Multimedia Systems*, 6(4):232–250, 1998.

[4] A. Celentano, O. Gaggi, and M.L. Sapino. Retrieving Consistent Multimedia Presentation Fragments. In *Workshop on Multimedia Information Systems (MIS 2002)*, pages 146–154, Tempe, Arizona, USA, November 2002.

[5] Jr. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.

[6] S. B. Eun, E.S. No, H.C. Kim, H.Yoon, and S.R. Maeng. Specification of Multimedia Composition and A Visual Programming Environment. In *Proc. ACM Multimedia Conference*, 1993.

[7] O. Gaggi and A. Celentano. A Visual Authoring Environment for Multimedia Presentations on the World Wide Web. In *IEEE International Symposium on Multimedia Software Engineering (MSE2002)*, pages 206–213, Newport Beach, California, December 2002.

[8] P. King, H. Cameron, H. Bowman, and S. Thompson. Synchronization in Multimedia Documents. In Jacques Andre, editor, *Electronic Publishing, Artistic Imaging, and Digital Typography, Lecture Notes in Computer Science*, volume 1375, pages 355–369. Springer-Verlag, May 1998.

[9] F.B. Paulo, P.C. Masiero, and M.C. Ferreira de Oliveira. Hypercharts: Extended Statecharts to Support Hypermedia Specification. *IEEE Transactions on Software Engineering*, 25(1):33–49, January/February 1999.

[10] B. Prabhakaran and S.V. Raghavan. Synchronization Models for Multimedia Presentations with User Participation. *Springer Verlag Journal of Multimedia Systems*, August 1994.

[11] James A. Schnepf, Joseph A. Konstan, and David Hung-Chang Du. Doing FLIPS: FLexible Interactive Presentation Synchronization. *IEEE Journal on Selected Areas of Communications*, 14(1):114–125, January 1996.

[12] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). http://sunsite.auc.dk/RFC/rfc/rfc2326.html, April 1998. RFC 2326.

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