A State-Transition Model for Distributed Multimedia Documents

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

Abstract

In this paper we present a state-transition model to describe a multimedia presentation evolution, i.e., its run-time behavior. Each object is modelled as an independent entity with its own behavior and resources allocation: a certain amount of bandwidth, buffer and a display (or an audio channel) for its playback. The evolution of a single media is modelled by means of a finite state machine, in which states transitions are triggered when some specific events occur, provided some conditions hold. The overall presentation is modelled as parallel (or sequential) composition of single media items' executions. The model is well suited for reasoning on multimedia documents dynamics, and to prove properties about them.

1. Introduction

Multimedia presentations can be defined by a collection of different types of media items and a set of spatial and temporal constraints over them. If we consider a distributed environment, media objects are dispersed over a computer network and must be downloaded before playback. Their retrieval from the server(s) is influenced by the network throughput, and buffer resources on the client side must be correctly sized to avoid jitters and stops in the presentation playback.

In this paper we present a model to describe a multimedia presentation evolution, i.e., its run-time behavior. Each object is modelled as an independent entity with its own behavior and resources allocation: a certain amount of bandwidth, buffer and a display (or an audio channel) for its playback. Each object can be considered as a *process* which requires specific *resources*. A correct presentation playback is the result of a correct *scheduling* of retrieval and display of media items.

The purpose of the model presented in this paper is to describe the run-time behavior of a multimedia presentation as the parallel (or sequential) composition of single media items' executions. The model describes the buffers allocation and deallocation of each media components and can be used as the basis for an algorithm to schedule the download and the playback of complex dynamic elements.

2. A model for multimedia presentations

We refer to the synchronization model for multimedia presentations defined in [3, 4], which we briefly describe addressing the reader to the bibliography for the rationale and the details.

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. An author can design the presentation evolution by imposing a set of temporal constraints among the objects, by means of five synchronization primitives: ***a plays with b*** ($a \Leftrightarrow b$) models the parallel composition of objects a and b; both objects start when either of them is activated, and when object a ends, also b does (if still active); the relation is therefore asymmetric; ***a activates b*** ($a \Rightarrow b$) models the sequential composition of objects a and b; b starts when a ends; ***a is replaced by b*** ($a \rightleftharpoons b$) models the replacement of media item a by b in the same channel; ***a terminates b*** ($a \Downarrow b$) models the stop of media item b, as a consequence of the forced stop of media item a; ***a has priority over b with behavior*** α ($a \overset{\alpha}{>} b$) is used to design presentation behavior during user interactions; media item b is paused ($\alpha = p$) or stopped ($\alpha = s$) when the user starts object a, e.g., through a hyperlink.

3. Description of the system

We consider distributed presentations in which the media to be displayed have to be previously downloaded. This usually requires bufferization. Each channel is therefore associated to a *buffer* to be used by the media item currently occupying that channel.

Buffer management is critical in distributed communication, since it affects the performance and, ultimately, the feasibility of a distributed application, even if limited to media download and presentation. In order to abstract from technical issues which do not limit the model power, we make a number of simplifying assumptions which are plausible in the framework of the multimedia presentations we approach.

First, we assume that the resources provided by the network are adequate, i.e., we face neither QoS problems nor strict real-time constraints in media synchronization; this assumption is acceptable for multimedia presentations, since fine-grain synchronization is resolved by putting synchronized media in the same file, such as multitrack video and audio file. Then, we assume that the time to process a media segment for display, once downloaded, is negligible wrt download time. In the same way, there can be some tolerances, e.g., at the end of a group of objects, there can be a little interval before the next one. Finally, since we assume that the resources are adequate for the whole presentation, we also assume that any parallel combination of media defined in the presentation can be played independently from the media download order, as long as all media are available in core memory when playback starts. In other words, we are interested only in the mutual *logical* relationships among media, and not in performance constraints that can be induced on them by the implementation.

We refer to a simplified double buffer schema over segmented media streams: a media object is divided into segments of equal length, and equal to the length of the buffer. Each time a new segment is required to start, the system switches to the unused buffer and begins to fill it. When the buffer is full, the application can begin media playback (constrained by the synchronization relationships), while the system retrieves from the network a new segment in the other buffer. Other variants (e.g., copying the buffer content into another location, or using a buffer pool) do not introduce significant changes. We shall use the word *buffer* to refer to a buffer area whose allocation policy is not detailed.

We call **pre-fetch** the activity of filling the first buffer for a media item; it defines the minimum delay between download and play in a streaming environment.

Therefore for the remainder of the paper we abstract from any 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, partially filled, or full.

The association of distinct media items with the buffers they are using, is expressed by means of the function $bf(m)$, shorthand for *buffer*, where m denotes the media. Analogously, we assume that every media m is associated to a playback channel, and a stream of data. To denote this association we use the functions $ch(m)$ and $st(m)$.

The relevant information to be checked, when a single media 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 $isEm()$ and $isFu()$, for empty and full respectively.

The channel occupation is given by the function $isUs()$, shortand for *isUsed*: $\mathcal{CH} \rightarrow \mathcal{MI} \cup \{\_\}$ that returns, for every channel, the media item that occupies it. The underscore symbol _ denotes the absence of media item; it is the value used to identify free channels.

In addition, it is possible to verify if the channel associated with a media is available. This control is done using the $isUs()$ function: the channel is available for the media m if the channel is free ($isUs(ch(m)) = \_$) or it is occupied by the media itself ($isUs(ch(m)) = m$). The predicate $isAv()$, that has the media item as its argument, defines the channel availability through these checks.

Each media exhibits its own behavior, which we model in terms of a sequence of different *states* of the media. Media objects are classified as *continuous* media, like video and audio, that once started have their own behavior, and *static* media, like still images, that are simply displayed on the user screen.

Media from the two classes have some states in common, while some other states are specific of continuous or static media items. Both continuous and static media can be **idle**, i.e., not active, waiting to be activated, as well as **init**, that is, pre-fetching data to (dis)play. Continuous media can also be **playing**, the state of media being delivered. The corresponding state for static object is **active**, i.e., actually rendered. Continuous media can be **paused**. When the last segment of a continuous media is playing, the item is in state **terminating**.

If we observe the presentation along time, it can be divided into a number of *states*: these states are more complex than the simple states in which a media is modelled, since the state of a media is embedded in the general state of the system, represented also by means of some *conditions*, that are a set of *facts*, which list a number of atomic conditions that are true in the state. These conditions concern the status of buffer, stream and channel of the media involved in the presentation: this status is checked by means of the predicates and the functions previously introduced in this section.

Therefore, we model the behavior of every single media by means of a *finite state machine*, and the evolution of a presentation, which is a complex dynamic system, as the composition of the machines corresponding to the single atomic components plus the conditions that describe the actual situation of the system.

State transitions are triggered by specific *external* events, that have an effect immediately perceived by the user (requests to start or stop the media) and by *internal*, non observable events, that correspond to some modification in the internal state of the system, that the user is not necessarily aware of (modifications in buffer status, like the start or the end of pre-fetching phase). In the presence of these events, state transitions are fired, provided some *preconditions*, expressed in terms of logical predicates, hold. The effects of the events on a state of the system are captured by *postconditions* associated to the events.

We denote the set of events that can cause a state transition with $\mathcal{E}$. It includes: **start**$_m$, when a media item m is activated; **ready**$_m$, when the pre-fetch of a media item is terminated; **pause**$_m$, when m playout is temporally interrupted; **stop**$_m$, when a user forces the termination of item m; **FF**$_m$**(p)** (fast forward), when a user asks to play an already active media item m, jumping at position **p**[1]; **RW**$_m$**(p)** (rewind), when a user asks to go back to position **p** in the playout of media item m; **ending**$_m$, when the last segment of the media is starting playing (the item is finishing) and **end**$_m$, when a media playout reaches its natural termination.

The external events are *start, stop, pause, FF(p), RW(p)* and *end*. The internal events are *ready* and *ending*.

An event has a direct impact on the state of the media, causing a state transition of the media, and, more in general, on the state of the system: given an event, its effects 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 any positive predicate instance appearing in the postcondition, inserting it in the resulting state. If the predicate instance $p(bf(m_i))$ is the inserted one, (that is, a fact stating something about the buffer of media item m_i), any other predicate $q(bf(m_i))$ appearing in the current state (and concerning the same media item) is then removed (this replacement captures the dynamic evolution of the buffer condition). Some predicates become true (and then are inserted in the current state) as a consequence of the interaction with the environment (these predicates are: $isFu(bf())$, $isEm(bf())$ and $isEm(st())$). That is, some changes that are captured in the state of the system are not induced by any media state transition. They are instead a reaction to some modification in the environment.

4. Single media item and composition of items

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

Definition 4.1 (Single Item Finite State Machine) The finite state machine characterizing a *continuous* media item m is $MSM_m = \langle S, s_0, F, next, T\rangle$, where (i) $S = \{\mathbf{idle}, \mathbf{init}, \mathbf{playing}, \mathbf{paused}, \mathbf{terminating}\}$; (ii) $s_0 = \mathbf{idle}$; (iii) $F = \{\mathbf{idle}\}$; (iv) $next$ is the function defined in Table 1; (v) T is a set of 4-tuples $\langle s, e, \mathcal{C}, \mathcal{P}\rangle$ 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 and $\mathcal{P}$ is a set of postconditions, that is, conditions holding after the transition takes place.

[1]The position **p** is determined by the user. Then the system selects the correct segment number and begins to fill the buffer of the media item.

next(**id**, start$_m$) = **in**	next(**in**, ready$_m$) = **pl**
next(**pl**, RW$_m$(p)) = **in**	next(**pl**, FF$_m$(p)) = **in**
next(**pl**, ending$_m$) = **tr**	next(**tr**, end$_m$) = **id**
next(**ps**, start$_m$) = **pl**	
next(**x**, pause$_m$) = **ps**, **x** $\in$ { **in, pl, tr**}	
next(**x**, stop$_m$) = **id**, **x** $\in$ { **in, pl, tr, ps**}	

Table 1. The function next: id=idle, in=init, pl=playing, s=paused, tr=terminating.

A finite state machine MSM_m modelling a static media m is obtained by the MSM_m previously defined removing states **paused** and **terminating** (and the transactions involving these states) and renaming state **playing** in state **active**.

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

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. For not mentioned predicates, persistency is assumed.

The set of transitions characterizing a media item is shown in Table 2.

Given this representation, 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 containing a number of independent media. Then, we specialize some transition rules, to model synchronization primitives.

In the following definition, we shall use footers to distinguish different media, and the footer corresponding to each media also refers 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 media item m_i. Analogously for postconditions $\mathcal{P}_{s_i,e_i}$.

Definition 4.2 Let $m_1, \ldots, m_n$ be n independent 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

$[isAv(m_i)]$	$\textbf{id} \xrightarrow{start} \textbf{in}$	$[\neg isEm(bf(m_i)) \wedge isUs(ch(m_i)) = m_i]$
$[isFu(bf(m_i))]$	$\textbf{in} \xrightarrow{ready} \textbf{pl}$	$[\neg isEm(st(m_i))]$
$[true]$	$\textbf{in} \xrightarrow{pause} \textbf{ps}$	$[true]$
$[true]$	$\textbf{in} \xrightarrow{stop} \textbf{id}$	$[isEm(bf(m_i)) \wedge isUs(ch(m_i)) = \_]$
$[isEm(st(m_i))]$	$\textbf{pl} \xrightarrow{ending} \textbf{tr}$	$[true]$
$[true]$	$\textbf{pl} \xrightarrow{FF(p)} \textbf{in}$	$[isEm(bf(m_i)]$
$[true]$	$\textbf{pl} \xrightarrow{RW(p)} \textbf{in}$	$[isEm(bf(m_i)]$
$[true]$	$\textbf{pl} \xrightarrow{pause} \textbf{ps}$	$[true]$
$[true]$	$\textbf{pl} \xrightarrow{stop} \textbf{id}$	$[isEm(bf(m_i)) \wedge isUs(ch(m_i)) = \_]$
$[isAv(m_i)]$	$\textbf{ps} \xrightarrow{start} \textbf{pl}$	$[isUs(ch(m_i)) = m_i]$
$[true]$	$\textbf{ps} \xrightarrow{stop} \textbf{id}$	$[isEm(bf(m_i)) \wedge isUs(ch(m_i)) = \_]$
$[isEm(bf(m_i))]$	$\textbf{tr} \xrightarrow{end} \textbf{id}$	$[isEm(bf(m_i)) \wedge isUs(ch(m_i)) = \_]$
$[true]$	$\textbf{tr} \xrightarrow{pause} \textbf{ps}$	$[true]$
$[true]$	$\textbf{tr} \xrightarrow{stop} \textbf{id}$	$[isEm(bf(m_i)) \wedge isUs(ch(m_i)) = \_]$
$[isFu(bf(m_i))]$	$\textbf{in} \xrightarrow{ready} \textbf{ac}$	$[true]$
$[true]$	$\textbf{ac} \xrightarrow{stop} \textbf{id}$	$[isEm(bf(m_i)) \wedge isUs(ch(m_i)) = \_]$

Table 2. Transition rules for independent media items: id=idle, in=init, pl=playing, ps=paused, tr=terminating, ac=active.

by the finite state machine $MSM = \langle S, s^0, F, next, T\rangle$, where (i) $S = \{\langle s_1, \ldots, s_n\rangle \mid s_i \in S_i, i = 1, \ldots, n\}$; (ii) $s^0 = \langle s_1^0, \ldots, s_n^0\rangle$; (iii) $F = \{\langle s_{f_1}, \ldots, s_{f_n}\rangle \mid s_{f_i} \in F_i, i = 1, \ldots, n\}$; (iv) $next(\langle s_1, \ldots, s_i, \ldots, s_n\rangle, e_{m_i}) = \langle s_1, \ldots, next_i(s_i, e_{m_i}), \ldots, s_n\rangle$, for any $s_i \in S_i$, and any event e_{m_i} on the the media m_i, $i = 1, \ldots, n$; (v) 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 $\langle\langle s_1, \ldots, s_i, \ldots, s_n\rangle, e_{m_i}, \mathcal{C}_{s_i,e_{m_i}}, \mathcal{P}_{s_i,e_{m_i}}\rangle \in T$.

In a presentation some media are related each other, therefore we must consider objects which are temporally related by the synchronization relationships described in Section 2. We translate the temporal relations into different composition of finite state machines, which are mostly based on the one considered above.

Specifically, the finite state machine modelling media items related by 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 4.2, to model the case of the general composition of items m_i and m_j, and (ii) adding some transition rules which will be described later in this section possibly overwriting already existing transition rules. Given a transition $t \in T$ for an event e from $\langle s_{m_i}, s_{m_j}\rangle$ to $\langle s'_{m_i}, s'_{m_j}\rangle$, if $t' = \langle\langle s_{m_i}, s_{m_j}\rangle, e, \mathcal{C}_{\langle s_{m_i}, s_{m_j}\rangle, e}, \mathcal{P}_{\langle s_{m_i}, s_{m_j}\rangle, e}\rangle \in T$ reaching the same state $\langle s'_{m_i}, s'_{m_j}\rangle$ already exists, t replaces t', otherwise t is added to the set of transitions T.

In order to modify the transition rules described in Table 2 for taking into account the effect of synchronization relationships on the media finite state machines composition, we define the notion of closure of an item, with respect to some synchronization relations, to capture the effects of event propagation among media.

Definition 4.3 ($SC_\Leftrightarrow$) The symmetric closure of a media item a wrt. $\Leftrightarrow$ is the set $SC_\Leftrightarrow(a)$ such that (i) $a \in SC_\Leftrightarrow(a)$, and (ii) for any item $c \in \mathcal{MI}$, if $\exists b \in SC_\Leftrightarrow(a)$ such that $b \Leftrightarrow c \in \mathcal{SR}$ or $c \Leftrightarrow b \in \mathcal{SR}$, then $c \in SC_\Leftrightarrow(a)$.

$SC_\Leftrightarrow(a)$ contains all media items related by a $\Leftrightarrow$ relationship, that are required to start simultaneously, when one of them is activated. From the definition $SC_\Leftrightarrow(b) = SC_\Leftrightarrow(a)$ iff $b \in SC_\Leftrightarrow(a)$.

The set $SC_\Leftrightarrow(a)$ results from the closure of a transitive chaining process. Basically, it includes all items which are transitively connected to a, by means of $\Leftrightarrow$ relationship. As it will be clearer in the following, there are cases in which some of the connected items have to be discarded. In this case, when computing the symmetric closure of a wrt. $\Leftrightarrow$, items connected to a have to be included in the closure only if the “connecting chain” does not include any discarded item. This notion of restricted closure is formalized in the following definition.

Definition 4.4 ($SC_\Leftrightarrow(a)_M$) The symmetric closure of item a wrt. $\Leftrightarrow$, limited by the set of items M, is the set $SC_\Leftrightarrow(a)_M$ such that (i) if $a \in M$ then $SC_\Leftrightarrow(a)_M = \emptyset$, else (ii) $a \in SC_\Leftrightarrow(a)_M$, and for any item $c \in \mathcal{MI} \setminus M$, if $\exists b \in SC_\Leftrightarrow(a)_M$ such that $b \Leftrightarrow c \in \mathcal{SR}$ or $c \Leftrightarrow b \in \mathcal{SR}$, then $c \in SC_\Leftrightarrow(a)_M$.

All the synchronization primitives of the model exhibit an asymmetric behavior. For some of them, a notion of transitive (forward) closure is needed, to deal with the forward propagation of the effects of an event. For the sake of space we introduce a *parameterized* asymmetric closure, in which the parameter rel acts as a place holder for $\Leftrightarrow$ or $\Downarrow$ synchronization primitives.

Definition 4.5 (C_{rel}) Let a be a media item in $\mathcal{MI}$ and $rel \in \{\Leftrightarrow, \Downarrow\}$. The closure of a wrt. rel is the set $C_{rel}(a)$ such that (i) $a \in C_{rel}(a)$, and (ii) for any item $b \in \mathcal{MI}$, if $b \in C_{rel}(a)$ and $b\ rel\ c \in \mathcal{SR}$, then $c \in C_{rel}(a)$.

Six new transition rules define the evolution of a composite presentation in case of events which have a wider impact on the document activating a cascade of simultaneous media activations or stops. Such events are **start**, **ready**,

stop, **ending** and **end**. When the system receives an event $start_{m_i}$, all media items which are related by a $\Leftrightarrow$ relationship, i.e., all media items in $SC_{\Leftrightarrow}(m_i)$, should begin to fill the buffer. The system controls if their channels are available (media items in $SC_{\Leftrightarrow}(m_i)_{NotAvailable}$) or if there are some media objects to replace, and in this case changes their states to **init** (and consequently, the states of the replaced objects become **idle**).

Event: $\mathbf{ready_{m_i}}$ for any $i \in \{1 \dots n\}$
Notation:
$NotAv = \{m_k \in \mathcal{MI} | \neg isAv(m_k)\}$
$TR = \{m_k \in NotAv | s_k = \mathbf{tr_{m_k}}\}$
$StartCl = SC_{\Leftrightarrow}(m_i)_{NotAv}$
$IP = \{m_j \in StartCl | s_j = \mathbf{in_{m_j}} \vee \mathbf{s_j} = \mathbf{ps_{m_j}}\}$
$Ready = \{m_j \in StartCl | isFu(bf(m_j))\}$
$Free = \{m_j \in StartCl | s_j = \mathbf{id_{m_j}} \vee \mathbf{s_j} = \mathbf{ps_{m_j}}$
$\wedge isAv(m_j)\}$
$Paused = \{m_k | (m_k^p \overset{p}{>} m_k) \in \mathcal{SR}$ such that $s_k \neq \mathbf{id_{m_k}}$
for some $m_k^p \in SC_{\Leftrightarrow}(m_i)\}$
Precondition:
$IP = Ready \wedge Free = \emptyset \wedge TR = \emptyset \wedge isFu(bf(m_i))$
Initial state: $\langle \mathbf{s_1}, \dots, \mathbf{s_n} \rangle$
Final state: $\langle \mathbf{s'_1}, \dots, \mathbf{s'_n} \rangle$, where
$\forall m_j \in IP$, $s'_j = \mathbf{pl_{m_j}}$ if m_j continuous,
$s'_j = \mathbf{ac_{m_j}}$ if m_j static;
$\forall m_j \in Paused$, $s'_j = \mathbf{ps_{m_j}}$;
$\forall m_j \notin IP \notin Paused$, $s'_j = s_j$
Postcondition: $\forall m_j \in IP$, $\neg isEm(st(m_j))$

Table 3. Event: $\mathbf{ready_{m_i}}$.

In case of event $ready_{m_i}$ the system controls if all media items have their buffers full, and in this case the presentation begins its playback (Table 3). Otherwise, the system waits for media objects already buffering and controls if there are other media items in $SC_{\Leftrightarrow}(m_i)$ for which the channel is now available and begins their bufferization.

If the user stops the playback of an item m_i, the system looks for all media items which must be stopped at the same time, i.e., all objects contained in $C_{\Downarrow}(m_i)$, frees their channels and changes their states to **idle** again. Otherwise, if m_i naturally ends, the system first notices that its stream is empty (event *ending*) and then that also the buffer is empty (event *end*). When the system receives the event $ending_{m_i}$, it checks what media objects must be started after its end, i.e., media items in $SC_{\Leftrightarrow}(m)$ such that a relationship $m_i \Rightarrow m$ exists and checks if their channels are available[2], or if some media items can be replaced. Then the system changes the states of media objects whose channels are available to **init** and begins their bufferization (Table 4).

[2]Since the stream is already empty, channel of m_i is considered available for new items.

Event: $\mathbf{ending_{m_i}}$ for any $i \in \{1 \dots n\}$
Notation:
$Started = \{m_k | (m_i \Rightarrow m_k) \in \mathcal{SR}\}$
$NotAv = \{m_k \in \mathcal{MI} | \neg isAv(m_k)\}$
$Repl^{ing} = \{m_j^r | m_j^r \in SC_{\Leftrightarrow}(m_k)$ for some $m_k \in Started$
$\wedge isUs(ch(m_j^r)) = m_j$ for some $(m_j \rightleftharpoons m_j^r) \in \mathcal{SR}$
$\vee (m_j^r \overset{\alpha}{>} m_j) \in \mathcal{SR} \vee m_j = m_i\}$
$StartCl = \bigcup_{m_k \in Started} SC_{\Leftrightarrow}(m_k)_{NotAv \setminus Repl^{ing}}$
$IP = \{m_j \in StartCl | s_j = \mathbf{id_{m_j}} \vee \mathbf{s_j} = \mathbf{ps_{m_j}}\}$
$Repl^{ed} = \{m_k | (m_k \rightleftharpoons m_k^r) \in \mathcal{SR} \vee (m_k^r \overset{\alpha}{>} m_k) \in \mathcal{SR}$
for some $m_k^r \in Repl^{ing}\}$
$StopCl = \bigcup_{m_k \in Repl^{ed}} C_{\Downarrow}(m_k)$
$I = \{m_j \in StopCl | s_j \neq \mathbf{id_{m_j}}\}$
Precondition: $isEm(st(m_i)) \wedge |IP| = |channel(IP)|$
Initial state: $\langle \mathbf{s_1}, \dots, \mathbf{pl_{m_i}}, \dots, \mathbf{s_n} \rangle$
Final state: $\langle \mathbf{s'_1}, \dots, \mathbf{tr_{m_i}}, \dots, \mathbf{s'_n} \rangle$, where
$\forall m_j \in IP$, $s'_j = \mathbf{in_{m_j}}$;
$\forall m_j \in I$, $s'_j = \mathbf{id_{m_j}}$;
$\forall m_j \notin IP \notin I$, $s'_j = s_j$
Postcondition:
$\forall m_j \in I$, $isEm(bf(m_j))$;
$\forall m_j \in I \setminus Repl^{ed}$, $isUs(ch(m_j)) = \_$;
$\forall m_j \in IP$, $\neg isEm(bf(m_j)) \wedge isUs(ch(m_j)) = m_j$

Table 4. Event: $\mathbf{ending_{m_i}}$.

When m_i naturally ends (Table 5), the system stops all media items in $C_{\Leftrightarrow}(m_i)$, i.e., media items currently playing in parallel, and frees their channels. Then it checks if there are some media objects that must be started after its end, and whose channels are now available. In this case changes their states to **init** and begins their bufferization.

A complete description of these transition rules, which are not detailed here due to lack of space, can be found in [1]; Tables 3, 4 and 5 summarize the events which are used in the example of Section 5.

5. An example

We introduce now an example to show how these rules are used. Let us consider a multimedia presentation about an artwork: an initial video ($intro$) introduces the history period related to the artwork and the artist who made it. At its end, activates another video clip ($vclip$) illustrating the artwork itself. This second clip plays in parallel with a soundtrack ($sound$) and a comment page ($caption$). At the end of $vclip$, a text page ($text$) is displayed, with information about the museum which contains the artwork. The channels are *video*, *audio* and *window*, such that $ch(intro) = ch(vclip) = video$, $ch(sound) = audio$ and $ch(caption) = ch(text) = window$.

Due to space constraints, we only comment a fragment

Event: $\mathbf{end_{m_i}}$ for any $i \in \{1 \ldots n\}$
Notation:
$End = \{m_j \in C_{\Leftrightarrow}(m_i) | s_j \neq \mathbf{id_{m_j}}\}$
$Started = \{m_k | (m_i \Rightarrow m_k) \in \mathcal{SR}\}$
$NotAv = \{m_k \in \mathcal{MI} | \neg isAv(m_k)\}$
$Repl^{ing} = \{m_j^r | m_j^r \in SC_{\Leftrightarrow}(m_k)$ for some $m_k \in Started$
$\wedge isUs(ch(m_j^r)) = m_j$ for some $(m_j \rightleftharpoons m_j^r) \in \mathcal{SR}$
$\vee (m_j^r \overset{\alpha}{>} m_j) \in \mathcal{SR} \vee m_j \in End\}$
$StartCl = \bigcup_{m_k \in Started} SC_{\Leftrightarrow}(m_k)_{NotAv \setminus Repl^{ing}}$
$IP = \{m_j \in StartCl | s_j = \mathbf{id_{m_j}} \vee \mathbf{s_j} = \mathbf{ps_{m_j}}\}$
$Ready = \{m_j \in StartCl | isFu(bf(m_j))\}$
$Repl^{ed} = \{m_k | (m_k \rightleftharpoons m_k^r) \in \mathcal{SR} \ \vee (m_k^r \overset{\alpha}{>} m_k) \in \mathcal{SR}$
for some $m_k^r \in Repl^{ing}\}$
$StopCl = \bigcup_{m_k \in (Repl^{ed} \cup End \setminus \{m_i\})} C_{\Downarrow}(m_k)$
$I = \{m_j \in StopCl | s_j \neq \mathbf{id_{m_j}}\}$
$ChFree = \{c | \exists m \in (I \cup End) \wedge c = ch(m) \wedge$
$isUs(c) = m\} \setminus \{c | \exists m \; c = ch(m) \wedge m \in IP\}$
Precondition: $isEm(bf(m_i)) \wedge |IP| = |channel(IP)|$
Initial state: $\langle \mathbf{s_1}, \ldots, \mathbf{tr_{m_i}}, \ldots, \mathbf{s_n} \rangle >$
Final state: $\langle \mathbf{s'_1}, \ldots, \mathbf{s'_n} \rangle$, where
$\forall m_j \in End \cup I$, $s'_j = \mathbf{id_{m_j}}$;
if $StartCl = Ready$ $\forall m_j \in Ready$, $s'_j = \mathbf{pl_{m_j}}$
if m_j continuous, $s'_j = \mathbf{ac_{m_j}}$ if m_j static;
else $\forall m_j \in IP$, $s'_j = \mathbf{in_{m_j}}$;
$\forall m_j \notin End \notin Ready \notin IP \notin I$, $s'_j = s_j$
Postcondition:
$\forall m_j \in I \cup End$, $isEm(bf(m_j))$;
$\forall c_j \in ChFree$, $isUs(c_j) = \_$;
if $StartCl = Ready$ $\forall m_j \in Ready$, $\neg isEm(bf(m_j))$;
else $\forall m_j \in IP$, $\neg isEm(bf(m_j)) \wedge isUs(ch(m_j)) = m_j$

Table 5. Event: $\mathbf{end_{m_i}}$.

of the presentation; a more detailed discussion on the example is in [1]. Consider the presentation when only the introduction video is playing, i.e., media items are in the following states: $\mathbf{playing_{intro}}$, $\mathbf{idle_{vclip}}$, $\mathbf{idle_{sound}}$, $\mathbf{idle_{caption}}$, $\mathbf{idle_{text}}$ and the following conditions hold: $isUs(\textit{video}) = intro$, $isUs(\textit{audio}) = \_$, $isUs(\textit{window}) = \_$ and $isFu(bf(\textit{intro}))$.

Suppose $ending_{intro}$ occurs (i.e., the data stream corresponding to the introduction became empty, i.e., $isEm(st(intro))$). The system then controls which items should be activated at the end of video $intro$, i.e., $vclip$, $sound$ and $caption$ and checks if their channels are available as described in Table 4. Therefore, the current states of media items are: $\mathbf{termitating_{intro}}$, $\mathbf{init_{vclip}}$, $\mathbf{init_{sound}}$, $\mathbf{init_{caption}}$, $\mathbf{idle_{text}}$.

When the condition $isEm(bf(intro))$ holds, event end_{intro} occurs. A state transition takes place (Table 5), and the states of the media become $\mathbf{idle_{intro}}$, $\mathbf{init_{vclip}}$, $\mathbf{init_{sound}}$, $\mathbf{init_{caption}}$, $\mathbf{idle_{text}}$. When the buffers of the video clip, the music and the caption page are full, the system processes the last $ready$ event as described in Table 3, activating these elements (i.e., moving presentation to $\mathbf{idle_{intro}}$, $\mathbf{playing_{vclip}}$, $\mathbf{playing_{sound}}$, $\mathbf{active_{caption}}$, $\mathbf{idle_{text}}$).

When also the stream of the video about the artwork $vclip$ becomes empty, the system receives the event $ending_{vclip}$, and changes the states of media items beginning the download of the final text pages according to Table 4: $\mathbf{idle_{intro}}$, $\mathbf{terminating_{vclip}}$, $\mathbf{playing_{sound}}$, $\mathbf{active_{caption}}$, $\mathbf{idle_{text}}$. When $bf(vclip)$ is empty, $vclip$ ends. As described in Table 5, the states of the media become $\mathbf{idle_{intro}}$, $\mathbf{idle_{vclip}}$, $\mathbf{idle_{sound}}$, $\mathbf{idle_{caption}}$, $\mathbf{init_{text}}$. Then, event $ready_{text}$ occurs when the buffer $bf(text)$ is full. Therefore, there is a transition to $\mathbf{idle_{intro}}$, $\mathbf{idle_{vclip}}$, $\mathbf{idle_{sound}}$, $\mathbf{idle_{caption}}$, $\mathbf{active_{text}}$ (see Table 3).

6. Conclusion

The abstract formal model introduced so far describes the behavior of a multimedia presentation in terms of resources allocation (buffers and network bandwidth) and synchronization among the media objects.

For this reason it can be used to define and check a sequence of media items download: during the presentation playback, the system calculates *a priori* what happens at the end of a component, i.e., which objects are activated (see Table 4), and finds out a correct scheduling download sequence for their bufferization.

The proposed model is also well suited for reasoning on multimedia documents dynamics, and to prove properties about them. For example, given a set of media items Act, the model can check if it is possible that the presentation reaches a state in which all of them are active. In this case, we need the complete finite state machine associated with the multimedia documents and a description of the initial state of the overall system, expressed in terms of (positive) predicates on media buffers and channels. Then, we can look in the composite finite state machine for a presentation state in which all media $m \in Act$ are in state **active** (if m is a static media) or **playing** (if m is a continuous item) and return the shortest sequence of events to reach that state.

Another interesting property, is the correctness of a sequence of media items download with respect to the modelled behavior of a presentation. The model can check if a given sequence is correct since the finite state machine completely describes the status of the buffers and channels at each moment. Therefore, we can control if the sequence of events *ready* is compatible with the finite state machine associated to the presentation. For example, if n media items must begin to playback in parallel, they change their states from state **init** to **playing** when the last buffer (i.e., the buffer corresponding to the object with biggest delay) is full (see Table 3). Therefore the set of possible correct download sequences contains all sequences which respect this property, no matter whether the system begins to fill some buffers before the others.

Other works approach the problem of multimedia scheduling. Candan et al [2] define a model to design and play multimedia presentations. Differently from our approach, it does not describe all the possible run-time behaviors of a multimedia document, but only the dynamic structure as designed by the author, through the use of a graph in which media items are the nodes and the edges are flexible temporal constraints among the objects. A possible presentation schedule (and the resources allocation) can be derived by the graph, but the model is not well suited to check other properties of the document.

In [7] the authors propose a new CPU scheduling technique to improve performance of multimedia and real-time applications, in which the management of events delivery is a critical point to avoid delays. The idea is to coordinate event scheduling and task scheduling by making the multimedia applications *event-aware*. The domain here is little different from the one addressed in this paper and includes multimedia application like virtual worlds or multi-player games.

Paulo et al. [6] describe an approach very similar to the one addressed here. The paper presents a synchronization model based on hypercharts, an extension of the finite state machine formalism. A hyperchart contains timed transitions to specify the temporal behavior of presentation activities whose firing depends on the state of the system, and the system performs a single step at each time unit, reacting to all external changes that happen in that time interval. Hypercharts provide mechanisms for specifying hypermedia requirements such as objects duration, delays, jitters and user interactions, but require a explicit definition of the time instant at which an event occurs, therefore our model allows a easier management of further modifications of a multimedia document.

In [5] Layaïda et al. discuss the effect of uncertainty in the duration of some media objects in multimedia scenarios. Media items can be distributed over the internet and the access delay can be very different. Different from our approach, users interactions with the document cause de-synchronization. The model proposes a scheduling algorithm based on flexibility to solve the problem of re-synchronization.

Our model provides a more general framework that allows to define a correct sequence of download for the media items of a multimedia presentation, as well as to investigate other properties of the real time behavior of the document. This second feature is not considered in the models present in literature.

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.

Acknowledgements

The authors would like to thank prof. Augusto Celentano for the helpful discussions and encouragements.

References

[1] P. Bertolotti, O. Gaggi, and M.L. Sapino. A State-Transition Model for Distributed Multimedia Documents. Technical Report 77/04, Department of CS, University of Turin, http://www.di.unito.it/∼bertolot/tech-report.pdf, March 2004.

[2] 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.

[3] A. Celentano, O. Gaggi, and M.L. Sapino. Retrieval in Multimedia Presentations. *ACM Multimedia Systems Journal*, to appear.

[4] O. Gaggi and A. Celentano. Modelling Synchronized Hypermedia Presentations. *Multimedia Tools and Applications*, to appear.

[5] N. Layaïda, L. Sabry-Ismail, and C. Roisin. Dealing with Uncertain Durations in Synchronized Multimedia Presentations. *Multimedia Tools and Applications*, 18(3):213–231, december 2002.

[6] 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.

[7] C. Poellabauer, K. Schwan, and R. West. Coordinated CPU and Event Scheduling for Distributed Multimedia Applications. In *ACM Multimedia Conference*, pages 231–240, 2001.