A Laboratory for Prototyping and Testing Multimedia Presentations

OMBRETTA GAGGI and AUGUSTO CELENTANO
Dipartimento di Informatica, Università Ca' Foscari di Venezia,
Via Torino 155, 30172 Mestre (VE), Italia
{*gaggi,auce*}*@dsi.unive.it*

In this article we describe a prototyping environment, which allows an author to set up and test a complex hypermedia presentation. It contains a visual editor, based on a graph notation, in which the nodes are media objects and the edges are the synchronization relations between them; an execution simulator, which helps the author to test the presentation dynamics by manually triggering media related events; and a player, which allows the author to preview the presentation and to visually relate the execution evolution with the interpretation of the synchronization schema.

Keywords: Hypermedia authoring and prototyping; media synchronization; multimedia presentation; execution simulation

1. Introduction

The integration of multimedia material into hypermedia documents on the World Wide Web is widely used in many applications like distance learning, web advertising and e-business, virtual tourism, cultural heritage, news delivery, entertainment, and so on. The improvements in CD and DVD technology, the price decrease and the growth of network bandwidth for home connections bring hypermedia documents and applications to the consumer market with great acceleration.

Authoring multimedia documents (often called multimedia presentations to highlight their active, dynamic behavior) is a complex task, since the author must deal not only with the structure and the layout of the document, but also with its temporal behavior. The task is more difficult when dealing with interactive documents, since unanticipated user interaction can alter the expected timing relationships between media. In this situation the integration and verification of a multimedia presentation is a critical activity which cannot rely on repeatedly checking the execution under all possible circumstances. Rather, a structured model should support the document design, and simulation of media related events and user interaction should be possible in the prototyping phase, in order to test the synchronization and coordination among media under many event combinations, besides their unattended execution.

In this article we illustrate *LAMP*, an authoring system oriented to fast prototyping and testing of interactive multimedia presentations. The acronym stands for *LA*boratory for *M*ultimedia presentations *P*rototyping; it is based on a temporal synchronization model for multimedia presentations we have developed in past

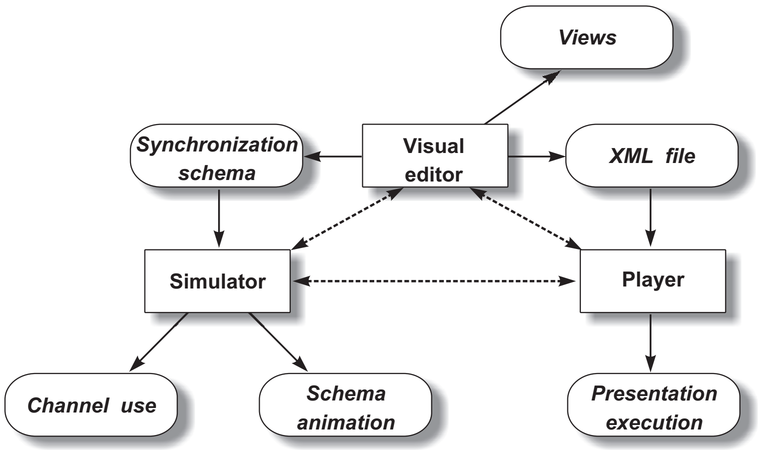


Fig. 1. The architecture of the *LAMP* environment

years and presented in Ref. 13. It allows the author to set up and test the dynamic behavior of a complex multimedia presentation by defining the synchronization relationships among media in an event-driven way. The model was designed with a focus on the World Wide Web, where the media coordination can be suitably defined by events triggering media download and activation, but can be used as well for synchronizing locally stored media collections, as in a CD-ROM or in a DVD.

The prototyping system (Fig. 1) contains a visual editor for authoring the presentation, an execution simulator to test the presentation behavior on different media-related and user-related events, and a player for the preview and the actual execution of the complete presentation. In Fig. 1 rectangles are system components, and ovals are data structures and visual results. Dotted arrows connecting the three components mean execution integration, while solid arrows show the flow of data among them.

The main component of the authoring environment is a visual editor based on a graph notation in which the nodes are media objects and the edges are synchronization relations between them, defined upon the occurrence of events like begin and end of the media components. Visual layout and playback channels can be defined.

An execution simulator interprets the presentation synchronization graph and the layout specification. It does not require the actual media file to be available, using placeholders if they are missing. Its goal is to help the author to test the presentation behavior not only during a normal continuous play, but also in presence of user interaction, like pausing and resuming a presentation, stopping a media playback, or following a link in a hypermedia document. Simulating the behavior of a presentation with media placeholders rather than executing it with actual media files speeds up the test phase, since the media events can be issued on an arbitrary time scale, without waiting for long playbacks in each test case. It allows also the designer to test the presentation under different relationships among the mutual time properties of media, e.g., by changing the order in which related media end, immediately observing the synchronization behavior for each case. In such a way, the number of cases which can be checked is increased without the need of supplying alternate media files. This possibility is valuable when designing not a single presentation, but a generic *presentation schema* which can be instantiated with several media files, generating different presentation instances derived from the same template. The design of a schema instantiated with variable data is common practice in dynamic Web sites, and is a practice emerging also in the multimedia domain[8]: the example that will be discussed in Section 3.4 is very close to this case.

The visual editor generates an XML external representation, suitable for further processing. It is used by the player, and supports also other applications (not shown in Fig. 1) devoted to multimedia presentation retrieval[9] and automatic generation of standard presentations from templates and variable data[8]. Many different views are supported by the editor, giving the author a complete control of the presentation layout, synchronization and media relationships.

The authored presentation can be translated into other models and languages for use outside the LAMP environment, e.g., in SMIL[29], the W3C standard for multimedia document integration. However, differences with SMIL exist which affect the translation, as it will be discussed later.

A player supports the presentation delivery to final users. It can be used also during the authoring phase as a previewer, showing the actual execution evolution according to the synchronization schema designed.

This article is organized as follows. After presenting the relevant literature on multimedia and hypermedia authoring in Section 2, in Section 3 we review the media synchronization model on which the prototyping system is built. The visual authoring system is illustrated in Section 4. Section 5 and 6 discuss the features and the use of the execution simulator and of the player. Section 7 discusses the presentation translation into SMIL, pointing out the relevant differences between the two models. Section 8 draws the concluding remarks.

2. Related Work

2.1. *Commercial tools*

Many metaphors are proposed by existing authoring tools. The simplest is the timed-based metaphor, used by Adobe Premiere[1] and Apple iMovie[3]: multimedia elements are presented and organized in tracks along a time line. Macromedia Director[23] implements a theatrical metaphor in which text, audio and other media objects are cast members on a stage, and the score is a sequencer which animates the actors. These metaphors are simple and intuitive, but are not easy to manage and maintain, since a modification to the time of an event can require to adjust the time relationships between several objects.

Macromedia Authorware[22] uses a flowchart-based paradigm where media objects are placed in sequence and grouped into sub-routines, like commands in procedural programming. With this metaphor the author needs not to explicitly define the time intervals ruling the multimedia presentation, which are computed according to the execution order of the media components.

Microsoft Producer[24] and Accordent's PresenterOne[26] are simple but effective tools for creating presentations for the Web which synchronize an audio or a video file with a set of slides, i.e., images or HTML pages. Video and audio files are divided into scenes and synchronized to images on a timeline by automatically calculating their download time. Transition effects can be added to enrich the presentation.

The MPEG-4 standard[15] has solicited a number of authoring tools for producing coded media streams and scene descriptions. Envivio Broadcast Studio[12] supports both spatial composition and temporal composition of media objects. iVAST Studio Author[16] is a visual environment dedicated to the creation of interactive MPEG-4 content that provides fine-grain control over audio, video, 2D graphics, animation and interactivity. It allows the user to build rich media presentations using familiar graphical user interface paradigms. A critical review of the issues related to the specific aspects of MPEG-4 standard is in Ref. 31.

2.2. *Research works*

Besides commercial products, many research works have designed multimedia authoring models based on different paradigms, able to create and manage temporal scenarios.

Many works are based on the SMIL language[29]. GRiNS[7], a GRaphical INterface for creating SMIL documents[29], provides three views, the *logical structure* view to define the temporal structure, the *virtual timeline* view to define and adjust fine-grain temporal interaction and the *playout* view to preview the presentation behavior and to define the spatial layout.

SMILAuthor[32] is an authoring tool for SMIL-based multimedia presentations implementing the timeline metaphor. It is a flexible tool which helps authors to generate reusable multimedia presentations. Since the editing functions are very difficult to perform in the language domain, the system includes an algorithm to calculate the playback duration of each object, based on the *Real-Time Synchronization Model, (RTSM)*, a model to represent the temporal relationships between media objects.

Madeus[21] is an authoring and presentation tool which uses graphs to represent both temporal and spatial constraints of a multimedia document. Graphs are used also for scheduling and time-based navigation.

The Hypermedia Presentation and Authoring System (HPAS) is based on the Media Relation Graph, a simple and intuitive hypermedia synchronization model[34]. A hypermedia presentation is modeled by a direct acyclic graph, where vertices represent media objects and directed edges represent the flow of time.

HyperProp[28] offers three graphical views of the document: the structural view,

Table 1. Comparison of the most relevant authoring tools

Tool	**Metaphor**	**Re-use**	**Testing**	**Layout**
Adobe Premiere	timeline	no	partial[1]	n/a
Macromedia Director	theatre	partial[2]	no	yes
Macromedia Authorware	flow-chart	yes	no	yes
iVast Studio Author	icon-based	partial[2]	yes	yes
GRiNS	icon-based	yes	partial[1]	yes
SMILAuthor	timeline	yes	partial[1]	yes
Madeus	graph-based	yes	no	yes
HPAS	graph-based	yes	no	no
HyperProp	timeline	yes	no	yes
MICE	graph-based	yes	partial[1]	yes
Mdefi	timeline	yes	no	yes
LAMP	graph-based	yes	yes	yes

[1]Preview only.
[2]Objects can be re-used only in the same presentation.

the temporal view and the spatial view. The author can use the structural view to browse and edit the logical structure of the document, the temporal view to represent the objects along a timeline and the spatial view to preview objects layout.

MICE[14], a visual software engineering tool, implements the graph-based metaphor, where nodes represent media objects (TAO) and directed edges represent both spatial and temporal relationships. The result is described using TAOML, an extension of HTML which allows authors to rapidly prototype multimedia applications using a standard web browser.

In Ref. 30, Tran-Thuong et al. integrate MPEG-7 tools for content modeling in a SMIL-like multimedia model, to make the authoring process easier. *Mdefi* provides three views: the timeline view for editing the temporal structure, the execution view to lay out media items in the user interface, and the hierarchical view, i.e., a tree view of the structure of the document.

2.3. *A comparison of authoring approaches*

Table 1 summarizes the differences among the most relevant systems reviewed with respect to the authoring metaphor used, the re-use of presentation modules and fragments, the presence of integrated testing, debugging and preview tools, and the layout design.

With respect to a timeline approach[1,3,24,28,30], a description based on graphs[14,21,34], flow-charts[22] and virtual timelines[7] is more flexible, since it does not depend on the actual length of the media items. On the other hand, the timeline description is more intuitive, since it is not always easy to capture the real presentation evolution from an event based description like a graph. But an event-based description, while less intuitive, provides a better specification of relationships among objects, mainly for presentations designed to be delivered on the World Wide Web.

In the WWW media are delivered independently, and timing is subject to delays due to server and network load. For the applications we address, the synchronization of separate media needs not to be fine-grained (e.g., lip synchronization is achieved by merging into the same file the video and the audio tracks), while it is important to address issues like the user interaction, that can modify the temporal behavior of a presentation with stops, reloads, VCR commands provided by players, hyperlink jumps, and so on.

Moreover, while continuous media have a duration defined by their playing time, for static media the concept of duration is fuzzy, since they do not evolve in time, but their life depends on the timing of other media (in our model it depends also on explicit timers). Therefore a time line specification for hypermedia presentations that integrate continuous and non continuous media with user interaction can only be an approximation of the real behavior. An event-based description is simpler to draw and to adapt, since it allows the designer to concentrate on the relationships between media without anticipating the actual behavior of each media items.

A user friendly authoring tool must allow the author to easily design and maintain multimedia presentations, but also to understand their final execution. Such different features can be provided by an event based approach (e.g., a flow-chart or a graph notation) integrated with tools to test the real presentation evolution, with particular attention to the changes in presentation behavior such as the ones produced by user interaction.

2.4. *Multimedia presentation testing*

Most approaches described in this section lack tools to test the multimedia presentation during design, before producing the final version. GRiNS, SMILAuthor, MICE, and also some commercial products like Adobe Premiere, provide tools to *preview* the presentation, but the user control during the preview is poor. In general, none of the examined systems is provided with true *testing* functionalities, through which the designer can exercise the presentation behavior under different execution conditions, with the goal of finding errors in the design or in the execution configuration[5,25].

Chan et al.[10] point out two major difficulties of multimedia software testing: the size and variability of media objects, and the environmental settings. At one side, in today's multimedia presentations, designed to be delivered in different environmental conditions, each media object is in effect a placeholder that can be mapped to several instances with different properties and behaviors. Checking a presentation with specific instances could not exercise the behaviors induced by different albeit compatible instances, and the extension in time of continuous media limits the number of executions that can be run.

At the other side, the uncertainty in the environment of a typical multimedia application prevents one of the fundamental requirements of traditional software testing, the predictability of the results. Therefore, real-time testing models are employed, related to resource management, quality of service and performance[35]. In such a way, the author can check if the expected behavior of the presentation can be obtained under the constraints and the variability of the delivery environment.

The testing of the conceptual design, however, is a different testing activity. It is crucial whenever the multimedia presentations deviate from simple schemas based on sequential and parallel execution of synchronized media: i.e., whenever the possible unexpected behaviors do not come from resource failures, but from wrong assumptions about the design of the presentation dynamics and of the media relations.

As in traditional software verification, formal approaches can be used[4,27,33], but they are difficult to apply when hypermedia concepts are introduced which multiply the execution paths to account for user interaction. They are also hard to apply to presentations based on a fixed template enriched with varying media instances taken from a repository and instantiated at run-time. As anticipated in Section 1, this scenario, drawn from the technology of dynamic Web sites, is becoming popular also in the multimedia domain.

In our authoring environment we aim at providing specific testing features to help the designer to prototype a multimedia presentation by checking its behavior under different media relationships, during all the design process. The testing activity can be performed in a simulated environment, abstracting the relevant synchronization properties of the media without using actual file instances, or using a previewer, thus obtaining the same view of the final user.

According to software testing techniques, our approach belongs to the category of *structural* or *white box* testing[25], based on the solicitation of the presentation control elements, i.e., the synchronization relations and the channel allocations.

As we shall discuss in Section 5, our system provides a clear connection between the media synchronization schema defined and the behavior obtained. The author can test the presentation or a part of it by triggering media related events, viewing which relationships are activated and possibly changing those relations which induce an unexpected behavior. In this way, *LAMP* provides a good trade-off between a flexible presentation description and a user friendly interface.

3. The Hypermedia Presentation Model

The authoring environment we present is based on an underlying model for multimedia presentations we have defined and discussed in Ref. 13. In this section we briefly review the components of the model that are necessary for understanding the *LAMP* overall philosophy.

The temporal behavior of a multimedia presentation is based on reactions of media items to events. Start and end of continuous media are the basic events which can trigger changes in the state of other media. The user can interact by pausing and resuming the media, by skipping forward or backward, and by following hyperlinks leading to a different location or time in the same document, or to a different document. At each user action the media must be resynchronized in order to keep the presentation coherent.

Due to the event-based style of the synchronization relationships, a graph notation is suitable for visually representing a dynamic multimedia document: media are represented as nodes, and synchronization relationships are represented as typed edges.

Compared to other approaches, discussed in Section 2, this model is simpler because it does not require the knowledge of actual media timing; it is more flexible, e.g., than a timeline-based model, since a change in a temporal property of an object does not propagate to the definition of other objects' behavior.

3.1. *Media and events*

A hypermedia presentation is modeled as a set of media and media containers (called *composites*), hierarchically organized; the outermost composites are called *modules.*

Continuous media like video and audio, called *clips*, are the units to which synchronization events are associated. Static media objects like texts and images, are called *pages*, and are synchronized by the continuous media behavior.

Two special media control user interaction and timing: a *user interaction object* is a continuous media object rendered as a button or as an active area, whose end is ruled by the user clicking on it; a *timer* is an invisible continuous media object whose length in time is set statically.

Each medium (except a timer) requires a device or a set of resources to be rendered or played. A *channel* is a virtual device which is used by one media item at a time, for all the duration of its playback. A channel can be a window browser on the user screen, a frame in a window, an audio channel, or a combination of video and audio resources like those required to play a movie with an integrated soundtrack.

3.2. *Synchronization relations*

Table 2 shows the five basic synchronization relations. The "*plays with*" ($\Leftrightarrow$) and the "*activates*" ($\Rightarrow$) relations roughly correspond to the `par` and `seq` tags of SMIL, but differences exist[a]. They behave as described in Table 2 only if media objects evolve naturally: in the relation $a \Leftrightarrow b$, object b is not terminated if object a is stopped before its end, e.g., by a user action; in the relation $a \Rightarrow b$, if object a is stopped before its end, object b is not activated.

The synchronization relations $\Downarrow$, $\rightleftharpoons$ and $\overset{\alpha}{>}$ are mostly used with user interaction, which involve not only the control of media execution but also the activation of hyperlinks. For example, the relation $a \Downarrow b$ is used to propagate a "stop" from an object to other objects, and can model skipping forward or backward inside a timed

[a]Section 7 discusses the problems of translating the presentation designed with *LAMP* into SMIL.

Table 2. The synchronization relations

Relation	**Editor's label**	**Behavior**
$a \Leftrightarrow b$	play	*a plays with b*: when a or b is activated, a and b play in parallel; object a is master, when it ends b is forced to terminate, if still active.
$a \Rightarrow b$	act	*a activates b*: when a ends, b is activated.
$a \Downarrow b$	stop	*a terminates b*: a forced end of a is propagated to b.
$a \rightleftharpoons b$	repl	*a is replaced by b*: when b starts, a is forced to terminate and to release the channel, which is used by b.
$a \overset{\alpha}{>} b$	pri α	*a has priority over b with behavior* α: when a is activated, b is paused ($\alpha = p$) or stopped ($\alpha = s$). If $\alpha = p$, b is resumed on a's end.

presentation. In the relation $a \overset{\alpha}{>} b$, a may be the target of a hyperlink that moves the user focus from the current document to another document, thus requiring the suspension or the stop of the one currently playing.

Other synchronization effects can be obtained by combining media into composites, which have an observable behavior as a whole. Finally, timers and user interaction objects can be used for obtaining behaviors not directly supported by the five basic synchronization relations: e.g., the overlap between two media a and b (as defined by Allen[2]) can be obtained by introducing a timer t setting the delay between a and b (a executes first), and the synchronization relations $a \Leftrightarrow t$, $t \Rightarrow b$.

The reader is referred to the cited work[13] for a complete discussion of details and motivations about this synchronization model.

3.3. *Execution specification*

If we observe a hypermedia presentation along time, its playout can be split into phases corresponding to the time intervals between two events. It can therefore be formally described by an automaton, whose states contain, among other information, the set of media active during the time span which corresponds to that state and the association between channels and media objects currently using them. Given an event, the transition function of the automaton moves the presentation to another state, thus describing how the presentation is affected by that event[9].

The automaton completely describes all the possible evolution of the corresponding presentation, under all conditions about triggering of events related both to unattended play and to user interaction. Therefore it can be used to compute the presentation reactions during the testing phase of the authoring process.

3.4. *A sample presentation*

As an example, we describe the summary of a news-on-demand presentation. The presentation is modeled following the introductory presentation of the highlights of

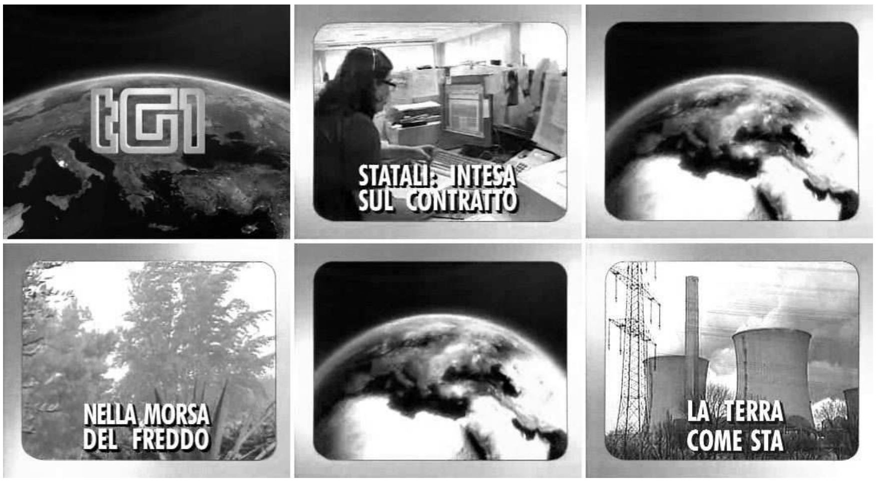


Fig. 2. A sequence of screenshots of a television news summary

the television news broadcast (named TG1, TG2 and TG3) of the public Italian network, which follow a common schema illustrated by the sequence of images of Fig. 2, taken from TG1.

After the opening titles (first image in Fig. 2), which are accompanied by a theme tune, a short summary of each story is read by a speaker and illustrated by a video (images 2, 4 and 6); a title superimposed on the video appears immediately (in TG2) or after a short delay (in TG1 and TG3). A short break between the stories shows an animation, which is a loop with the logo image in TG2, a fragment of the opening title in TG1 and TG3 (images 3 and 5). A background tune is played during the summary.

In our example we follow a mix of features of the three television news, and assume that after the last story an index of the articles is displayed, from which the user can select the ones to be delivered.

Figure 3 pictorially shows the synchronization structure of such a presentation. The media items are represented by symbolic icons which recall their type: video, audio, text, etc.. The voice of the speaker, the video and the title of each story are played in parallel, with a 2 seconds delay, set by a timer, between the video start and the title. When the speaker ends reading a story, an animated break is activated by the relation $\Rightarrow$ between the story audio and the animation timer, and by the relation $\Leftrightarrow$ between the timer and the animation; the animation loops, if it is shorter than the time set by timer, due to the relation *animation* $\Rightarrow$ *animation*. The start of the break also removes the title of the previous story; this action has to be defined explicitly because the title is a static medium (a text) which has no dynamic behavior by its own.

When the delay set by the break timer has elapsed, the break ends and activates

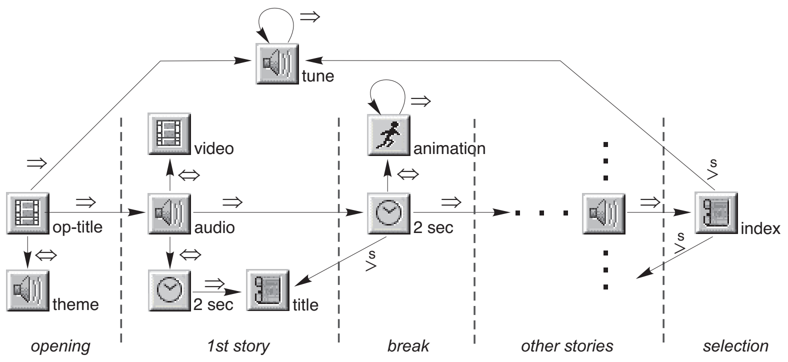


Fig. 3. The synchronization structure of a news-on-demand presentation

the following story. The background tune is activated after the opening title, as modeled by the relation *op-title* $\Rightarrow$ *tune*, and loops continuously, as modeled by the relation *tune* $\Rightarrow$ *tune*. In the last story, the end of the audio activates an index of the articles, stopping the background tune. The index is a list of hyperlinks to other presentations each of which features a complete article. The synchronization between the summary and the articles is defined through $\overset{\alpha}{>}$ relations, but are not discussed in the example.

It is worth to note that the synchronization structure, while giving no information about the time length of the media involved, states that the duration of each story is equal to the duration of the audio comment, which is the master medium in the relation *audio* $\Leftrightarrow$ *video*, causing the video to stop its playback when the audio ends.

This consideration shows that the use of events to represent the synchronization among the media items is more flexible than representations based on media temporal properties such as, e.g., a timeline. In fact, the graph shown in Fig. 3 represents a *family of executions*, where the exact duration of each medium (hence the exact sequence of media related events) is not defined, but the mutual occurrences are visible.

For example, assuming that the video and the audio are longer than two seconds (the delay for the title display), hence that the title is displayed while the video and audio are still playing, two different sequences of media related events can be observed for each story: (1) the *video* ends before the *audio*, and (2) the *audio* ends before the *video*. In the first case when the video ends, its last frame is displayed until the end of the audio, while in the second case the video is stopped when the audio ends. A strict interpretation of the synchronization schema in Fig. 3 would require also to analyze the case in which the audio or video end before the title display, but this situation is not meaningful from the television news point of view, even if it is considered in the presentation schema.

Generally speaking, the different behaviors subsumed by the synchronization schema of Fig. 3 depend on the order of the media items natural end, and can be deduced from the number of permutations of *end* events associated to the dynamic media.

The number of different reactions to synchronization events can also be deduced from the number of different paths in the automaton that formally describes the presentation evolution (ignoring the different executions of the *tune* loop), which is illustrated in Fig. 7, in Section 5.1.

4. A Visual Environment for Authoring Multimedia Presentations

We have developed a complete visual authoring environment for prototyping multimedia presentations using the Java language. The choice of Java, despite of performance problems, comes from the availability of platform-independent GUI and media player packages which allowed us to speed-up the prototype implementation.

Authoring consists mainly in drawing the synchronization graph of the presentation; all the temporal aspects of the presentation are defined by manipulating the graph. The visual editor provides also facilities to design the screen layout and to define the playback channels, to simulate the presentation dynamics, and to generate the XML description for the player.

The authoring environment does not provide functions for editing the media components, but only to assemble the corresponding files; this is not a limitation, since a wide choice of good programs for digital media manipulation is available.

The *LAMP* environment has been tested on MS-Windows, Linux and Mac OS X environments. It uses the Abstract Windows Toolkit (AWT) for the graphical user interface, which relies on the native GUI toolkit, thus preserving the look and feel of each platform. The figures included in this article show the interface of the Mac OS X environment.

Three other libraries supported the development of the visual interface:

- the Swing library[17], a fully-featured library implementing windowing functionalities,
- the JGraph library[20], a library for graphs visualization and manipulation, used to implement the synchronization graph, and
- the EZD library[11], a generic graphics drawing library used in the simulator interface.

The Java Media Framework (JMF) API[19] was used for implementing the player. JMF enables audio, video and other time-based media to be added to Java applications, providing a simple architecture to synchronize and control several media objects. Even if a platform-independent version of the JMF libraries is delivered by Sun Microsystems, which should guarantee media compatibility, we experienced some differences in media support on different platforms, and different video and audio codecs are supported by optimized, platform-dedicated versions, such as the one for MS-Windows. For example, Quicktime MOV videos are fully supported, while MPEG-1 files are supported by the Windows optimized version, not by the platform-independent version on Mac OS X. The configuration of codecs installed on the machine also influences the JMF behavior.

The authoring system exports an XML description of the presentation, that can be further processed, through the JAXB library[18] (Java API for XML processing).

The XML description is structured in three sections: the `layout` section contains the description of the channels used by the media items, which are defined in the `components` section; in the `relationships` section, the synchronization relationships of the presentation are defined. The `components` section contains both the description of the hierarchical structure of the document, in terms of composites, and the information about the media items, like the channel used and the location of the corresponding resource (attributes `channel` and `file`). Details about the XML description are given in Refs. 8 and 13.

A second XML file is maintained by the editor to store all the information needed to manage the visual properties of the presentation schema, such as the position of the media items on the screen, the coordinates of the connecting edges, the channel colors, and so on, describing the presentation in terms of the design user interface, discussed in the next section.

4.1. *The user interface*

The visual editor provides two views: one to define the channels and the layout of the presentation, and one to define the temporal behavior.

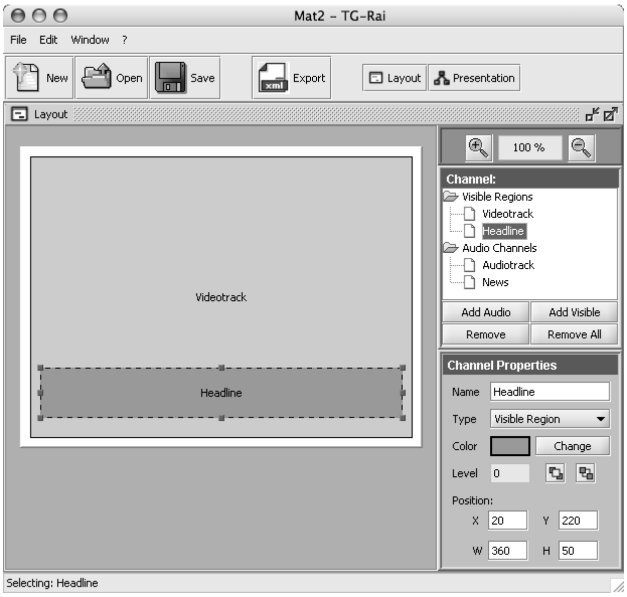
Channels can be of two types: regions, i.e., screen areas, and audio channels. The user defines the size of the multimedia presentation window, and creates the regions by drawing rectangles on it; each region has a unique name. The audio channels are defined only by assigning them a name as illustrated in Fig. 4. Colors distinguish the different regions, and are used consistently in the authoring process[b]. Regions can be moved and resized by direct manipulation.

With reference to the example, two regions are defined: *videotrack* for the video component of each story, and *headline* for the story title. The opening titles, the break animation and the index of articles are also displayed in the *videotrack* region. Two audio channels are defined, *audiotrack* for the tunes and *news* for the speaker's voice reading the stories.

The temporal behavior view provides the author with a panel on which to draw the graph which describes the presentation dynamics in terms of synchronization among media. Each media object is drawn as a rectangle with an icon which identifies the type. The rectangle is colored like the associated region (hollow for audio objects) to show the channel used. Consistency between channel definition and usage is checked by the editor.

New media objects and composites are inserted by selecting the corresponding

[b]In the figures distinguishable shades of grey are used to convey color information.



Channel Properties
Name: Headline
Type: Visible Region
Level: 0
Position:
X: 20 Y: 220
W: 360 H: 50

Selecting: Headline

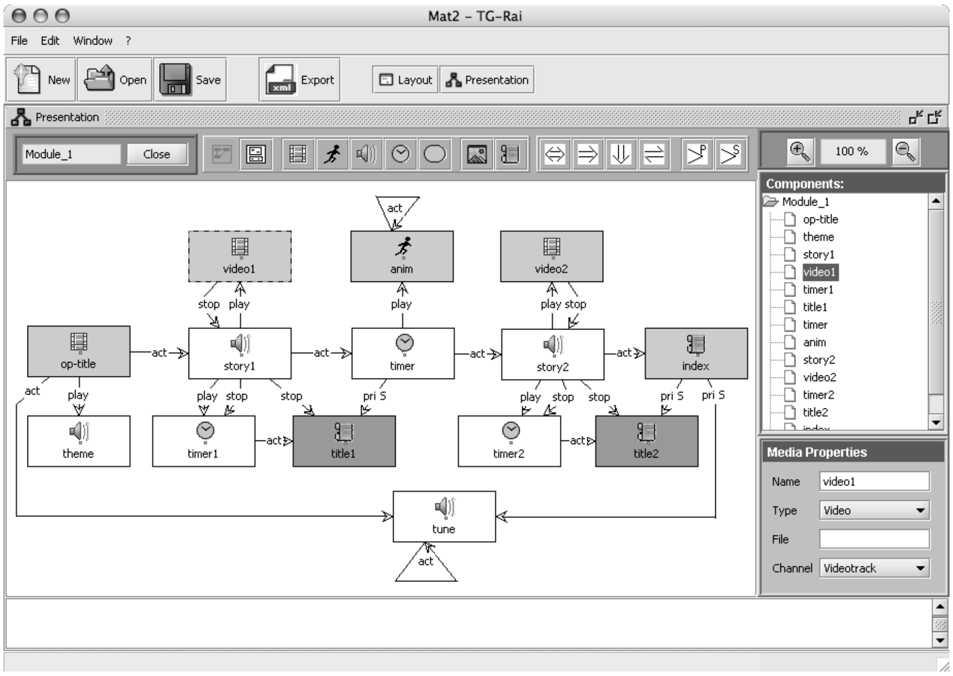
Fig. 4. The layout and channel definition of a news-on-demand presentation

icon from the media toolbox, and placing them in the drawing area. Properties are set or changed trough a panel which specifies the media name, its type, the associated resource (file name or URL) and the channel used for its playback. The resource can be specified at a later time without preventing the author from testing the presentation, as discussed in Section 5.

Synchronization relations are established by selecting a relation type from the relation toolbox and by drawing a connecting edge from the first to the second object, which is then labeled with the name of the relation type.

Figure 5 shows the appearance in the visual editor of the synchronization structure depicted in Fig. 3. Additional synchronization relations have been added to manage the situation in which the user stops a story. Assuming that the user can control the video region with VCR-style controls, stopping the video must also stop the audio, which in turn stops the timer and the title; since the time at which the stop command is not known, it must be directed to both media, and will be effective for the one active at that time.

In Fig. 5, the presentation is limited to two stories for simplicity. Indeed, it is a typical case of instance of a more general schema, identified by a recurrent pattern, easy to recognize, that contains a single news highlight (i.e., the video and the audio files and the associated timer and title) together with the animation and the timer which compose the break between the stories. The definition of presentation templates which can be instantiated with different data at run-time is an extension of this model, which is discussed at length in Ref. 8.



Name video1
Type Video
File
Channel Videotrack

Fig. 5. The synchronization graph of a news-on-demand presentation

5. Testing the Presentation Execution

An execution simulator allows the author to test the temporal behavior of the presentation. The simulator uses media placeholders to show the channels' usage. The author, without being compelled to follow a real time scale, can trigger all possible events, both internal (e.g., the end of an object play) and external (e.g., a user-executed stop, a hyperlink activation or the start of a dynamic medium under direct user control) to see how the events are propagated and how the presentation evolves. The simulator checks the triggered events against the current presentation state, detecting inconsistent situations such as the stop of a non-active medium, or the start of a medium whose playback channel is not free, and issuing proper diagnostic messages.

The simulator provides two interfaces for two different simulation styles. According to the first style, the simulation of channel use, media placeholders are used to show the presentation dynamics as perceived by the final user; a second simulation style, the animation of the synchronization graph, shows how the synchronization relations affect the presentation execution.

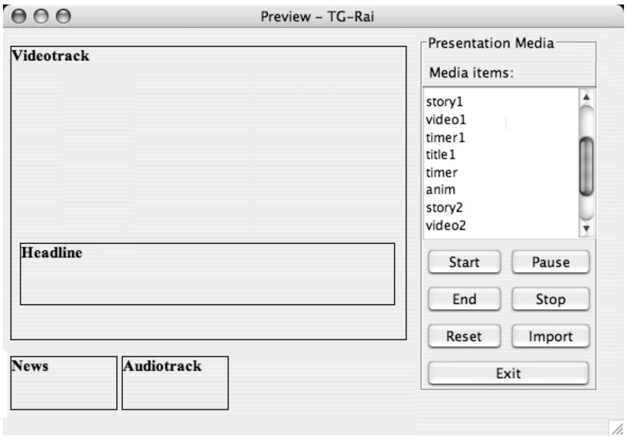


Fig. 6. The simulation of channel use

5.1. *Simulation of channel use*

The simulator opens a new window which lists the media components of the presentation and displays the channels as designed by the author in the layout view. Audio channels, which do not have a layout, are represented as small rectangular areas out of the presentation window (Fig. 6).

The author can select a medium from the list and can send events to it: *start*, *stop* and *end*, thus simulating its beginning, its forced stop or its natural end. Otherwise, the author can select a file of pre-generated events, and see the presentation evolution step by step.

The simulator execution follows step by step the evolution of the automaton which formally describes the presentation dynamics. Figure 7 shows the automaton of the sample presentation: each state S_i contains the set of active media, while the transitions are labeled with the media related events.

At each step, the simulator window contains all the information about the state of the presentation. The user selects the event to simulate and the simulator executes the transition function of the automaton, displaying the new state. The mapping between the channels and the media items currently using them is represented by the content of the regions in the simulator window, while the sets of active media objects contains all the objects which are using a channel.

At the beginning of the simulation all the channels are free, therefore they are drawn as empty, uncolored areas. Each channel is marked with its name. When the author starts a media item, the channel it would use in the real execution is filled with a media placeholder which is an image or a text taken from the media file, or is colored with the color set in the layout view for that channel if the media file is not available (audio channels are in white); the media item name is displayed in the channel area. A number of incoherencies are checked and reported, e.g., if the

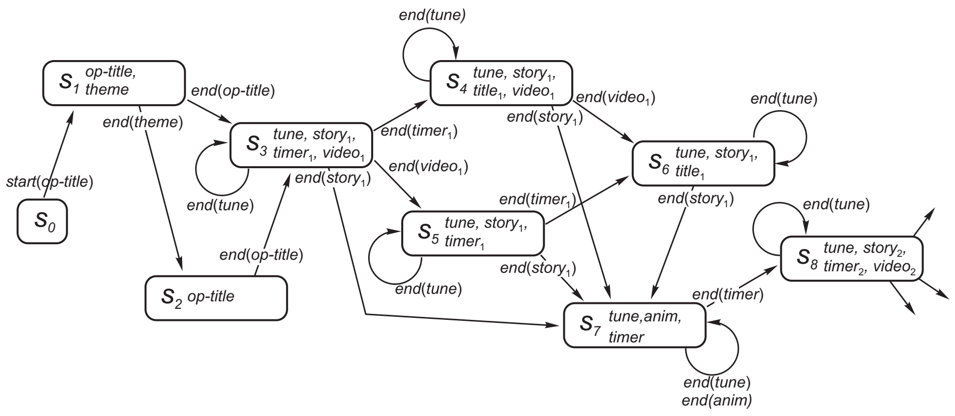


Fig. 7. A fragment of the television news presentation automaton

channel is busy because it was used by another media item and not released, an error message is displayed.

The simulator checks the relationships defined by the author looking for the ones which involve the selected media item (let us call it a):

- if a is a paused media, it is restarted;
- for each object b such that the relation $a \Leftrightarrow b$ or the relation $b \Leftrightarrow a$ exists, the simulator starts object b, i.e., it fills the corresponding channel with the media placeholder or with the channel color, and with its name;
- for each object b such that the relation $a \rightleftharpoons b$ exists, the simulator first stops object b, releasing the channel, then activates object a;
- for each object b such that the relation $a \overset{\alpha}{>} b$ exists, the simulator first stops (if $\alpha = s$) or pauses (if $\alpha = p$) object b and then activates object a.

A trace of the events triggered and media activated is logged for tuning and debugging purposes.

The author can end or stop an active media. In both cases, the simulator releases the corresponding channel by removing the media placeholder and replacing the object name with the channel name. Then, as for media start, it analyzes the relationships between the ended and stopped media item and the other media items. If the author ends media item a:

- for each object b such that the relation $a \Leftrightarrow b$ exists, the simulator stops object b;
- for each object b such that the relation $a \Rightarrow b$ exists, the simulator activates object b;
- for each object b such that the relation $a \overset{p}{>} b$ exists, the simulator resumes object b.

If the author stops a media object a, the simulator forces the termination of all objects b for which the relation $a \Downarrow b$ holds. If the author pauses a media object a, the simulator pauses also all media objects b if a relation $a \Leftrightarrow b$ exists. In such a way the author can see in every moment which channels are busy and which media are using them.

The simulator can also load a list of pre-recorded events, giving the author a complete unassisted animation of specific and different presentation behaviors in a simulated time scale.

5.2. *Animation of the synchronization graph*

The interface described above is a simulation of media layout and dynamics during the presentation playback. However, understanding the reasons for the observed behavior, i.e., which part of the synchronization graph is currently involved in the execution, is not simple. Except for the name of the media active in the channels, no other information is visible. In order to improve the user perception of the events and their relationships with the media, the simulation animates also the graph of the presentation: when an object is activated, the corresponding node in the synchronization graph is highlighted. Then the relations triggered by the activation of the object are also highlighted, and their effect is propagated to the related objects, giving the author the visual perception of how media synchronization is achieved.

For checking the natural evolution of the presentation, the user sends *end* events to the dynamic media, experiencing different combinations of relative timing; for checking the behavior in presence of user interaction, the user can send *start*, *stop*, *pause* and *resume* events. After any event, the graph is animated by showing how this event is propagated to the target medium and to other media objects through the synchronization relations. Therefore, the simulator allows the author to spot (and possibly remove or change) the relations which induce an unexpected behavior.

Figure 8 shows some steps of the simulation of the presentation illustrated in Fig. 5[c]. Active media objects and relations are highlighted with thick lines. In step (a) the presentation is playing the opening titles: the user has started the presentation by sending the *start* event to the *op-title* medium. The *theme* tune is also activated due to the relation *op-title* $\Leftrightarrow$ *theme*.

The transition from step (a) to step (b) is caused by the end of the theme tune; *op-title* is still playing because it is the master medium in the *op-title* $\Leftrightarrow$ *theme* relation, therefore it is not affected by the other medium's end. In step (c) the user sends an *end* event to *op-title*, and the first story is activated due to the relation *op-title* $\Rightarrow$ $story_1$, which plays together with $video_1$ and $timer_1$. The soundtrack *tune* is also activated. The simulation continues as illustrated in steps (d)–(e) of

[c]The graphical aspect is different from that of the visual editor because a different graphic library has been used, but the representation of the synchronization graph is conceptually the same.

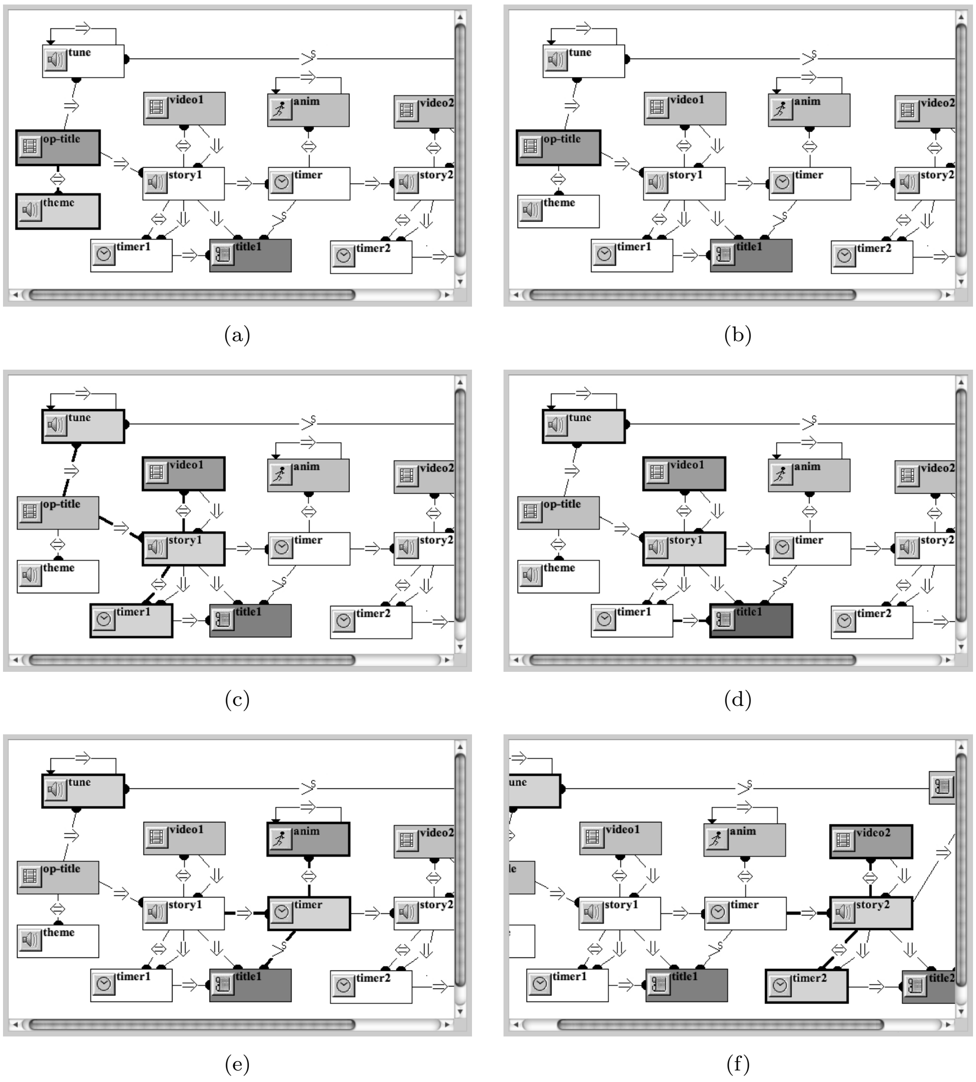


Fig. 8. The simulation of a news-on-demand presentation

Fig. 8 due to the following sequence of events: $end(timer_1)$, $end(story_1)$, $end(timer)$. The soundtrack *tune* continues playing since no *end* event has been directed to it. Figure 9 shows the state of the channel occupation after triggering the end of the title timer delay during the play of the first story; it corresponds to the step (d) in Fig. 8.

Since the simulation is independent from the actual media file duration, the author can simulate any combination of object duration. For example, in any of the situations illustrated in steps (c)–(f) of Fig. 8, the author can trigger the end of

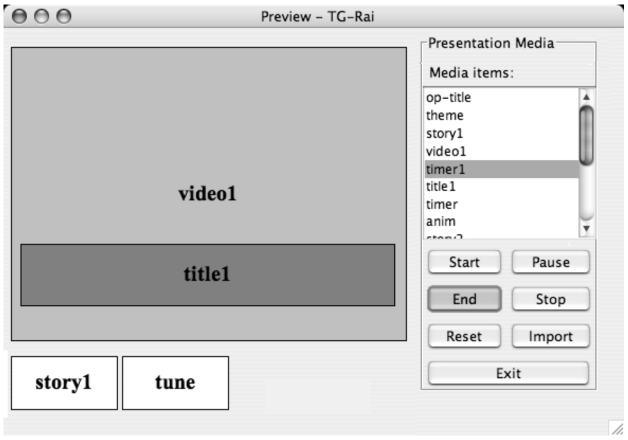


Fig. 9. The channel occupation corresponding to step (d) of Fig. 8

the background *tune*, but since it loops the presentation state does not change, as justified by the self loops in the automaton of Fig. 7; the *tune* ⇒ *tune* relation in the synchronization graph is highlighted to show the reason for such a behavior.

5.3. *Test data selection*

The simulator can be used in two ways: the user can interactively select the events to simulate, triggering them one by one, or import a file which contains a sequence of events, running the simulator in automatic playback mode. In the latter case, the simulator shows the evolution of the presentation step by step triggering on a conventional time scale the events in the sequence. In both cases, the sequence of events is test data, and the animation of the graph, shown by the simulator, constitutes, together with the input data, a test case of the presentation.

The main problem of running the simulator with predefined data is the choice of event sequences which are coherent with the presentation execution: e.g., the events in the test data should not try to trigger the stop of a non active media, and should not activate media out of their natural ordering, unless explicit hyperlinks are provided in the presentation schema.

The solution to this problem relies on the automaton itself, which allows test data to be validated. From each automaton's state an edge starts for any event that may occur during that state, and brings to a new state which captures all the consequences of that particular event. Therefore, test data can be checked against the automaton by following the events along a path from the initial to the final state.

The automaton can also be used to generate test data for a complete and systematic test of a presentation according to coverage testing techniques. As discussed in the literature on software testing, different perspectives on what is "covered" by a

Table 3. A set of test data for path coverage testing

#	Event sequence	State sequence
1	*start*(*op-title*), *end*(*theme*), *end*(*op-title*), *end*($timer_1$), *end*($video_1$), *end*($story_1$), *end*(*timer*)	$S_0, S_1, S_2, S_3, S_4, S_6, S_7, S_8$
2	*start*(*op-title*), *end*(*theme*), *end*(*op-title*), *end*($timer_1$), *end*($story_1$), *end*(*timer*)	$S_0, S_1, S_2, S_3, S_4, S_7, S_8$
3	*start*(*op-title*), *end*(*theme*), *end*(*op-title*), *end*($video_1$), *end*($timer_1$), *end*($story_1$), *end*(*timer*)	$S_0, S_1, S_2, S_3, S_5, S_6, S_7, S_8$
4	*start*(*op-title*), *end*(*theme*), *end*(*op-title*), *end*($video_1$), *end*($story_1$), *end*(*timer*)	$S_0, S_1, S_2, S_3, S_5, S_7, S_8$
5	*start*(*op-title*), *end*(*theme*), *end*(*op-title*), *end*($story_1$), *end*(*timer*)	$S_0, S_1, S_2, S_3, S_7, S_8$
6	*start*(*op-title*), *end*(*op-title*), *end*($timer_1$), *end*($video_1$), *end*($story_1$), *end*(*timer*)	$S_0, S_1, S_3, S_4, S_6, S_7, S_8$
7	*start*(*op-title*), *end*(*op-title*), *end*($video_1$), *end*($timer_1$), *end*($story_1$), *end*(*timer*)	$S_0, S_1, S_3, S_5, S_6, S_7, S_8$
8	*start*(*op-title*), *end*(*op-title*), *end*($story_1$), *end*(*timer*)	S_0, S_1, S_3, S_7, S_8
9	*start*(*op-title*), *end*(*op-title*), *end*($timer_1$), *end*($story_1$), *end*(*timer*)	$S_0, S_1, S_3, S_4, S_7, S_8$
10	*start*(*op-title*), *end*(*op-title*), *end*($video_1$), *end*($story_1$), *end*(*timer*)	$S_0, S_1, S_3, S_5, S_7, S_8$

set of test data can be taken. In our scenario, *path coverage* seems the most suitable technique; in fact, the execution of all the media without considering their dependencies, which roughly corresponds to the so called *instruction coverage* in software testing, is not meaningful since we need to test the mutual relationships between media. *Branch coverage* is also suitable; it means to execute the presentation by reaching each state from every possible previous state; or, in other words, executing the presentation by leaving each state under every possible event occurring in that state. However, branch testing does not test all the different combinations of events along time, leading to a weaker way to exercise the presentation.

According to path coverage, test data can be automatically generated by following all the distinct paths (all the distinct edges for branch coverage) in the automaton. In principle there are paths of infinite length due to looping media, but in practice loop traversal can be limited to a finite number of iterations without affecting the significance of test.

With reference to the automaton of Fig. 7, a set of test data covering all the paths from state S_0 to state S_8, without considering the loops induced by the repeating *tune*, contains 10 different event sequences, shown in Table 3. To cover all the edges of the automaton (still ignoring the *tune* loops), only 5 sequences are needed, numbered 1 and 7–10 in Table 3.

6. The Player

Besides the authoring tool and the simulator, *LAMP* contains also a player of the complete presentation designed. The player reads the XML file produced by the authoring tool and displays the channels according to the `layout` section. Then, like the simulator, it builds step by step the automaton of the presentation from the `relationships` section and finds the media item which starts the presentation. As the user starts playback, the player starts the first media object and computes the transition function of the automaton to find out which media items are active in the next state. Media objects are located through the `file` attribute in the `components` section.

The player could work jointly with the simulator to improve the prototyping activity of multimedia presentations[d]. If the author feels uncomfortable with the simulator schematic interface, he or she could play the presentation with actual files on a true time scale, viewing the synchronization evolution in real time according to the relations triggered. The coordination of all the components of the *LAMP* environment could then ensure that the author has a clear perception of presentation playback and easy tracing of any design error of the authored presentation.

Although the visualization offered by the simulator is limited, it has, however, some advantages with respect to the player. Simulation allows the author to prototype the presentation in a "what if" style, and is also worth when the media components are long-lasting files, since the author can trigger the end of a long video sequence without waiting for the actual duration, to see the subsequent evolution. In the same way, the absence of media embedded timing allows the author to model and check the behavior of sections of the presentation whose structure is repeated several times along the presentation time span, possibly with different media instances.

7. Multimedia Presentation Translation in SMIL

The presentation designed and tested can be played as a stand-alone presentation in the LAMP player, that interprets the XML file generated by the editor. The presentation can also be translated into another language, more widely available or more suitable for delivery over a network with full resource control mechanisms.We discuss here the translation into SMIL 2.0, the W3C standard for multimedia documents[29].

The presentation layout raises no problems: each visual channel can be translated into a SMIL region, a screen area hosting media to be played, with the same spatial arrangement. Audio files do not require a specific channel in SMIL, therefore audio channels are ignored.

The main differences between the two models concern the lack of a reference model for the data structure in SMIL, and the type of media synchronization definition. Our model organizes media objects into a hierarchical structure useful to design complex presentation with recurring patterns. It can be used to infer some temporal relationships between media without defining them explicitly: e.g., all

[d]Work is in progress for fully integrating the two components, which are subject to separate version and library updates.

the media objects contained in a composite structure must be stopped when the structure itself is stopped.

As far as media synchronization is concerned, our model is fully event-based, while SMIL adopts a timeline metaphor on which events can be defined to alter the linear flow of media. We must note that SMIL does not cover all the synchronization constraints defined in our model. In particular the two models differ in the way actions directed to end the media execution affect the subsequent execution of the presentation. Like Allen's relationships, SMIL's native features do not distinguish between the natural and the forced termination of a medium; therefore, the effects induced on a presentation component by a user act or by the behavior of another component cannot be easily distinguished by the effects induced by the natural evolution of the presentation. This limitation narrows the set of behaviors defined in our model, which can be represented using SMIL. Other differences concern the ability of the user to interact directly with the single media components to change the presentation behavior.

As a consequence, even if it is sometimes possible to provide rules for translating the synchronization relations into SMIL tags, they do not consider the overall synchronization schema of the presentation, failing to generate the correct behavior. As an example, the $a \Rightarrow b$ relation can in most cases be translated with the `seq` tag, but applying this correspondence with a local view can lead to wrong translation: e.g., the relation $a \Rightarrow a$ does not map into a sequence but plays media a in a loop, and must be translated with the SMIL attribute `repeat="indefinitely"`. A more subtle example is the set of relations $a \Rightarrow b$, $b \Rightarrow c$, $c \Rightarrow a$, which interpreted separately define media sequencing, but taken as a whole define a loop spanning three media objects.

Therefore, a systematic translation of the multimedia presentation into SMIL cannot be provided by a general algorithm working only at the level of the individual synchronization relations. However, it is possible to design an algorithm for defining with SMIL's tags and attributes a set of presentation behaviors expressed with our model, by analyzing the behavior of the presentation at a higher level, as defined by its automaton. The algorithm covers most cases of practical interest, and is based on the results of a work co-authored by one of the authors of this article[6].

In that work the authors propose an algorithm to extract a SMIL script from an automaton describing the behavior of a continuous multimedia presentation in a context very similar to the one discussed in this article. Given the automaton derived from a presentation designed with LAMP, such as the one depicted in Fig. 7, the shortest path among the paths whose states cumulatively contain all the media of the presentation contains all the information needed to build a corresponding SMIL script[e]. Along the path, each state represents the media items playing in parallel, and two consecutive states define the sets of media items which are played in sequence. The algorithm is indeed more complex, and considers not only the

[e]This property is justified in Ref. 6.

set of active media in the automaton states, but also the events which step the automaton to the next state. Without entering into details here, from the sample presentation depicted in Fig. 5, whose automaton is partially illustrated in Fig. 7, the algorithm finds the path traversing the states $S_0, S_1, S_3, S_4, S_7, S_8$, and extracts the following SMIL code, which correctly translates the presentation behavior[f]:

```
<seq>
    <par end="op-title.end">
        <video id="op-title" />
        <audio id="theme" />
    </par>
    <par end="seq1.end">
        <audio id="tune" repeat="indefinite"/>
        <seq id="seq1">
            <par end="story1.end">
                <video id="video1" />
                <audio id="story1" />
                <text id="title" begin="2s" />
            </par>
            <anim id="anim1" repeatdur="2s"/>
            <par end="story2.end">...</par>
        </seq>
    </par>
    <text id="index" />
</seq>
```

However, even if a general translation scheme can be derived from the analysis of the presentation behavior described by the automaton, the scheme does not apply consistently to all types of synchronization structures. The reader is referred to the cited article[6] for a discussion about this algorithm and of its limitations. In particular, our model can define presentations composed of independent flow paths, selected by the events occurring to the continuous media. In the automaton, such paths share an initial sequence of states, but then split in distinct states, in which new different media items start their playback. In such cases, the translation into SMIL must follow a different way.

Consider for example, the parallel playback of two continuous media, *a* and *b*. When one of the two objects terminates its execution without interruption, the other one is stopped, and one out of two images is displayed: *c* if *a* ends first (i.e., *b* is stopped by *a*'s end), *d* if *b* ends first. Figure 10 shows the synchronization schema and the automaton of this presentation fragment according to our model.

This synchronization schema must be represented in SMIL with an `excl` structure selecting one of the two media, *c* or *d*, according to the mutual time length of media *a* and *b*, as illustrated by the following fragment:

[f]Non relevant parameters are omitted

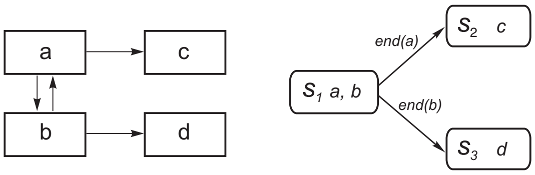


Fig. 10. A presentation requiring specific SMIL translation

```
<par>
    <par end="a.end;b.end">
        <video id="a" .../>
        <audio id="b" .../>
    </par>
    <excl>
      <priorityClass peers="never">
        <img id="c" begin="a.end" .../>
        <img id="d" begin="b.end" .../>
      </priorityClass>
    </excl>
</par>
```

This example fails to fit the general translation schema; in this case, the reason for such a difference is the inability of SMIL to individually handle the natural end of a media item and the premature termination due to a synchronization act.

Generally speaking, the presentations designed with LAMP can be translated straightforwardly into SMIL only in some cases, characterized by a substantial linearity of media execution. Complex presentations, that trigger different behaviors according to more sophisticated media execution control, require more complex translation schemas that, as far as we have investigated the problem, do not fall into a homogeneous processing schema.

8. Conclusion

In this article we have described the *LAMP* system, a laboratory for rapid prototyping of multimedia presentations. Its components are a visual authoring environment based on a graph representation of a multimedia synchronization model, an execution simulator and a player to play the authored presentation to the final users.

LAMP provides facilities to define the layout of a hypermedia presentation, to set up the synchronization relationships, to examine the parallel and sequential execution of media, and to save an XML description of the presentation. The simulator allows the author to test the presentation dynamic behavior by triggering events related to media start and end, and to user interactions like pause, stop and hyperlink activation. The possibility to generate test data in a systematic way for coverage testing, archiving them in a file for later reuse, greatly improves the presentation's test phase.

The authoring environment and the underlying multimedia presentation model were proposed to the students of the "Hypermedia Systems" course at the Ca' Foscari University of Venice in Fall 2004. The students were asked, as part of their examination task, to design a multimedia presentation using different models, among which the model presented here and the SMIL language. This experiment revealed that, approaching the problem directly with the SMIL language, the students had some difficulties in translating the desired behavior into a correct set of synchronization tags, mainly when dealing with user interaction or media of unknown length. Moreover, they were often not able to relate the SMIL tags used to the actual media visualization in a cause/effect relationship.

A frequent misleading situation was the need to distinguish between a natural end of a media item play and its forced stop as a reaction to another event, associating different presentation behaviors to the two cases, a situation common in interactive presentations. The students found difficult to understand and implement such a distinction using models in which there is no conceptual difference between the two situations. They were thus applying the same synchronization constraints to both cases, leading to a discrepancy between the expected and the obtained behavior, without finding in an evident way the reasons for such a mistake.

Such problems were better approached using the LAMP authoring environment, since the underlying synchronization model clearly distinguishes the reactions to different events, and the simulator execution highlights which synchronization relation is responsible for the observed behavior.

9. Acknowledgments

The authors acknowledge the contribution of Diego Medici to the development of the authoring tool, and Daniele Dal Mas and Alessandro De Faveri for the player implementation. One reviewer made many detailed comments that helped us to improve the article.

References

1. Adobe Systems Inc. Adobe Premiere. http://www.adobe.com/premiere.
2. J. F. Allen. Maintaining knowledge about temporal intervals. *Comm. ACM*, 26(11):832–843, November 1983.
3. Apple Computer Inc. Apple iMovie 3. http://www.apple.com/imovie/.
4. A. F. Ates, M. Bilgic, S. Saito, and B. Sarikaya. Using timed CSP for specification, verification and simulation of multimedia synchronization. *IEEE Journal on Selected Areas in Communications*, 14(1):126–137, 1996.
5. B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, 1990.
6. P. Bertolotti and O. Gaggi. A study on multimedia documents behavior: a notion of equivalence. *Multimedia Tools and Applications*, to appear, preliminary version: Techn. Rep. 82/05, Dipartimento di Informatica, Università di Torino, January 2005, http://www.di.unito.it/~bertolot/tech-report-mtap04.pdf.
7. D.C.A. Bulterman, L. Hardman, J. Jansen, K.S. Mullender, and L. Rutledge. GRiNS: A GRaphical INterface for creating and playing SMIL documents. In *WWW7 Con-*

ference, Computer Networks and ISDN Systems, volume 30(1-7), pages 519–529, Brisbane, Australia, April 1998.

8. A. Celentano and O. Gaggi. Template-based generation of multimedia presentations. *International Journal of Software Engineering and Knowledge Engineering.*
9. A. Celentano, O. Gaggi, and M. L. Sapino. Retrieval in multimedia presentations. *Multimedia Systems Journal*, 10(1):72–82, 2004.
10. W. K. Chan, M. Y. Cheng, S. C. Cheung, and T. H. Tse. Automatic goal-oriented classification of failure behaviors for testing XML-based multimedia software applications: an experimental case study. Technical report, HKU CS Tech. Report TR-2005-04, 2005.
11. Compaq Research. The Ezd library. http://www.research.compaq.com/wrl/projects/Ezd.
12. Envivio Inc. Envivio Broadcast Studio. http://www.envivio.com/products/ebs.html.
13. O. Gaggi and A. Celentano. Modelling synchronized hypermedia presentations. *Multimedia Tools and Applications*, 27:53–78, 2005.
14. A. Guercio, T. Arndt, and S.-K. Chang. A visual editor for multimedia application development. In *International Conference on Distributed Computing Systems Workshops (ICDCSW '02)*, pages 296–304, 2002.
15. ISO/MPEG. Overview of the MPEG-4 Standard, ISO/IEC JTC1/SC29/WG11 N2725. mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm, 1999.
16. iVAST Inc. iVAST Studio Author. http://www.ivast.com/products/studioauthor.html.
17. Java 2 Platform, Standard Edition (J2SE). J2SE Technology. http://java.sun.com/j2se/.
18. Java Architecture for XML Binding (JAXB). The JAXB library. http://java.sun.com/xml/jaxb/.
19. Java Media API. Java Media Framework API. http://java.sun.com/products/java-media/jmf/.
20. JGraph.com site. The Java Graph Visualization Library. http://www.jgraph.com/.
21. M. Jourdan, N. Layaïda, C. Roisin, L. Sabry-Ismail, and L. Tardif. Madeus, an authoring environment for interactive multimedia documents. In *ACM Multimedia 1998*, pages 267–272, Bristol, UK, September 1998.
22. Macromedia Inc. Macromedia Authorware. http://www.macromedia.com/authorware.
23. Macromedia Inc. Macromedia Director. http://www.macromedia.com/director.
24. Microsoft. Microsof Producer for PowerPoint 2002. http://www.microsoft.com/office/powerpoint/producer/.
25. G. J. Myers. *The Art of Software Testing.* John Wiley & Sons, 2004.
26. RealNetworks, Inc. Accordent's PresenterOne. http://www.realnetworks.com/products/presenterone/index.html.
27. C. A. S. Santos, L. F. G. Soares, G. L. de Souza, and J.-P. Courtiat. Design methodology and formal validation of hypermedia documents. In *Proceedings of the 6th ACM International Conference on Multimedia*, pages 39–48. ACM Press, 1998.
28. L. F. G. Soares, R. F. Rodrigues, and D. C. Muchaluat Saade. Modeling, authoring and formatting hypermedia documents in the HyperProp system. *Multimedia Systems*, 8(2):118–134, 2000.
29. Synchronized Multimedia Working Group of W3C. Synchronized Multimedia Integration Language (SMIL) 2.0 Specification, August 2001.
30. T. Tran-Thuong and C. Roisin. Multimedia modeling using MPEG-7 for authoring multimedia integration. In *ACM SIGMM International Workshop on Multimedia In-*

formation Retrieval, pages 171–178. ACM Press, 2003.
31. *Workshop on MPEG-4 Authoring, 14th ACTS Concertation Meeting*, Bruxelles, May 1999. http://www.cordis.lu/infowin/acts/analysys/concertation/multimedia/reports/mpeg.htm.
32. C. Yang and Y.Yang. SMILAuthor: An authoring system for SMIL-based multimedia presentations. *Multimedia Tools and Applications, Kluwer Publ. Co.*, 21(3):243–260, 2003.
33. Chun-Chuan Yang. Detection of the time conflicts for SMIL-based multimedia presentations. In *Proc. of 2000 International Computer Symposium (ICS2000) - Workshop on Computer Networks, Internet, and Multimedia*, pages 57–63, 2000.
34. J. Yu. A simple, intuitive hypermedia synchronization model and its realization in the Browser/Java environment. In *Asia Pacific Web Conference*, pages 209–218, September 1998.
35. J. Zhang and S. C. Cheung. Automated test case generation for the stress testing of multimedia systems. *Software: Practice and Experience*, 32(15):1411–1435, 2002.