International Journal of Software Engineering and Knowledge Engineering
Vol. 13, No. 4 (2003) 419–445
© World Scientific Publishing Company

TEMPLATE-BASED GENERATION OF MULTIMEDIA PRESENTATIONS

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

Data-centered approaches to multimedia presentation design and implementation can be developed by extending methodologies and technologies common in text-based applications. A multimedia report is a multimedia presentation built on a set of data returned by one or more queries to multimedia repositories, integrated according to a template with appropriate spatial layout and temporal synchronization, and coherently delivered to a user for browsing. We discuss the problem of defining templates for such multimedia reports with a focus on media coordination and synchronization. Multimedia presentations can be automatically generated according to the template by instantiating it on actual data instances. An XML language describes the spatial layout and the temporal constraints of the media objects. An authoring system and a player have been implemented.

Keywords: Multimedia presentation, schema modelling, XML, data integration, automatic generation.

1. Introduction

Data-centered approaches to multimedia application design and development are growing and broadening due to the rapid progress in technology for display, creation, storage and transfer of multimedia documents, which gives the user new possibilities to access and retrieve information of different kinds.

Often the user interface to information is based on web clients, a standard and platform independent solution for displaying dynamically assembled documents with data selected and retrieved from databases and framed in templates. Web-based applications receive increasing attention, and models, methodologies and tools exist to support designers and programmers in defining, prototyping, testing and deploying such applications.

When shifting from mostly static to mostly dynamic media the term *presentation* better describes the type of application which integrates and displays information. A broad range of applications like distance learning, web advertising and e-business, virtual tourism, cultural heritage, news delivery and entertainment are based on multimedia presentations where continuous media like audio and video play a fundamental role.

Continuous media add a time dimension to the integration of different information items, and introduce coordination and synchronization constraints in the design of data-centered applications.

The authors of multimedia presentations must design the coordinated playback of different media and the consistent interpretation of user interactions. If information that is displayed comes from a data repository, its identification and extraction requires additional care. Re-use of media for different purposes, or adaption to user profile and history, are requirements asking for the design of presentations according to well defined models and schemas.

In such a scenario the automatic generation of standard multimedia presentations with data extracted from a repository is a valuable goal that allows authors to build with limited effort several variants on one template schema without redesigning the whole application from scratch.

We aim at automatically generating multimedia presentations by defining templates based on recurring patterns, focusing the discussion on coordination and synchronization of continuous media. In previous works we have defined a synchronization model and developed an authoring tool for multimedia presentations [7, 8]. In this paper we extend both the model and the authoring tool in order to model template schemas for automatically building multimedia presentations. With a schema authoring system we aim at giving the author the possibility to define the layout and the behavior of the presentation, the characteristics and attributes of the objects involved without knowing the instances that will be used to fill the template.

The paper is organized as follows: Section 2 introduces multimedia reports as a class of multimedia presentations. Section 3 reviews the relevant literature. Section 4 discusses authoring of multimedia presentations in terms of synchronization among the media objects. Section 5 defines the structure and the components of a report template definition, while in Section 6 an XML language suitable to describe the spatial layout and the temporal constraints of multimedia presentations and report templates is presented. Section 7 presents the algorithms for integrating data into a report template, in order to generate a complete presentation. Section 8 presents an authoring environment based on the model. Section 9 discusses handling of missing data, and Section 10 comments about consistency issues and draws the concluding remarks.

2. Automatic generation of multimedia presentations

The automatic generation of multimedia presentations is based on two phases.

In the first phase the template of a presentation is defined, in which the multimedia items can be placed in a coordinated way according to the desired dynamics. The template describes the general synchronization structure of the presentation by defining temporal and synchronization constraints among media items. One or more queries are also defined which intensionally describe the presentation contents, i.e., the media items such as video and audio files, images and text files. This design phase is executed once for each presentation type.

In the second phase, which is executed for each new presentation, data is retrieved from a data repository according to the queries defined in the template

design phase, the template is filled with the retrieved data, the spatial and dynamic relationships are instantiated, and the presentation is played.

The two phases are tightly interconnected during early development stages. The template instantiation on a data sample could suggest changes in the template in order to improve the final presentation. In some cases such automatic generation can be a first prototyping step of a more refined presentation, especially if the data repository is quite stable.

Globally, the two phases build a continuous presentation in which data extracted from a multimedia data repository is located, connected, synchronized and coherently presented to the user. We call this activity *multimedia reporting*, i.e, the automatic generation of multimedia documents modelled with respect to a template, whose content is retrieved according to selection parameters.

In the most general case multimedia reporting would require the designer to approach and solve many problems about data selection, e.g., how to coherently integrate data coming from one or several databases. In Section 10 we shall briefly discuss some issues about this problem, we note here that too much generality prevents from satisfactory solutions and is in some way in contrast with the idea of "reporting", an activity based on standardization. Therefore we assume the following scenario for our work:

1. The presentation collects data into groups, like in a text report, such that in each group data of different types exists (video, audio, text, image), whose instances are related like in a relational table. More precisely, we assume that each group is structurally equivalent to a relational table where columns identify the media types and rows are instances. Some values can be NULL values, denoting that in some instances some media can be missing.

2. Apart from groups, "background" data exists which are associated to the presentation as a whole, or to parts of it identified by a group or a sequence of groups, such as a continuous soundtrack, a permanent title, a background image, and so on.

3. The whole presentation consists of the coordinated (e.g., sequential) playback of the groups, taking care of user actions like pause, stop, rewind, and so on.

4. No *a priori* constraint is put on the time properties of continuous data items, but the system should be able to coordinate the execution by synchronizing the beginning and end of the data group components.

Conceptually, defining multimedia reports is not different from building text-only reports or dynamic web pages: the author must define the structure of the report or of the web page, i.e., the data layout, and the query to select and retrieve relevant data. In the case of multimedia reporting, data items collected have a temporal behavior, which increases the complexity of the structure definition by adding a new dimension to the task: the author should deal with synchronization

problems and temporal sequencing of objects. If the spatial layout definition could be trivial, this is not true for the temporal dimension.

As an example, an author could design a news-on-demand service based on a database of articles stored as related multimedia document items: video documentaries, audio and text comments, images, and so on. A multimedia report is built from the selection of the appropriate news, by presenting them as in a synchronized sequence. Each article has a video story, an audio comment, and a text, which must be synchronized. The articles are normally played one after the other, but the user can interact with the presentation, thus changing its linear behavior, e.g., the user could skip forward or backward, or could stop or pause the playback of a medium item. In such case the whole presentation must be re-synchronized, therefore the report definition must be supported by a model of media synchronization which handles events generated by user interaction.

3. Related work

The problem of automatic generation of synchronized multimedia presentations with variable data has been approached in recent years. There are two main approaches, which represent two different points of views of the problem:

1. the author defines a template which intensionally describes the structure and the behavior of a presentation which contains multiple instances of a repetitive pattern, selected from a data repository;

2. a multimedia presentation is defined as a collection of constraints on media items. The generated presentation is a solution of the constraints set which can also consider some additional parameters imposed by the user.

The first approach allows the re-use of the template of the presentation for other multimedia reports, and does not require the author to set the same synchronization relationships for all the instances. The second approach is less suitable for generating reports, but can be better used to adapt one presentation to different user profiles or to different contents. The use of a template-based approach is more suitable to deal with a set of multimedia data of unknown cardinality, following a constant schema.

The first approach is used by SQL+D [4, 5], an extension to SQL which allows users to retrieve multimedia documents as result of querying a multimedia database. An SQL+D query specifies all presentation properties, from screen layout to its temporal behavior. In addition to SELECT-FROM clauses, the user can define DISPLAY-WITH clauses to describe screen areas (called *panels*), in which groups of retrieved media items are placed with specified relative positions. A SHOW clause defines the temporal behavior in terms of timed sequences of returned instances display. SQL+D requires the authors to have specific skills. Even if a user interface helps to build the query, they must know the structure of the database from which they retrieve multimedia data. Moreover, queries on multiple databases are not

allowed. Differently from our approach, which is based on synchronization events, temporal constraints are defined by arranging multimedia objects along a timeline. This solution is less flexible, and can require to know in advance the data contained in the database in order to obtain a complex dynamic behavior. Moreover, SQL+D does not allow the author to include in the presentation media items which do not depend on the query performed.

A high level specification based on constraints is proposed within the project *Dynamo*, *Semi-automatic Hypermedia Presentation Generation* project, aimed at "increasing the level of automated adaptation of varying user and system characteristics during the process of creating hypermedia presentations" [15]. A presentation can be configured to adapt to a number of user-related parameters such as the current state of knowledge of the user, the task he/she is involved with, his/her preferences, and environment-related parameters concerning the available resources. In the framework of the *Dynamo* project, Geurts et al. [9] present a formalism to construct multimedia documents by defining semantic relations between media objects. Differently from the model presented here, spatial layout and temporal dynamics can be described through the use of both quantitative and qualitative constraints. Qualitative constraints facilitate high-level reasoning, but they are often not sufficient because they do not define a precise design (e.g., the author states that figure A is on the left of another object, but is not interested in specifying the number of pixels between them). The author does not design the template of the presentation with specific layout and behavior, but defines a set of constraints; the system builds a multimedia presentation which obeys the constraints. A prototype is developed called Cuypers [16, 17], which is a transformation environment supporting semi-automated assembling of multimedia documents according to a rich structural and semantic annotation based on XML. The annotation allows for the specification of different processing steps concerning semantic structure, constraints satisfaction and final form presentation, which occur in multimedia authoring, to be integrated in a single execution stream.

Delaunay^{MM} [6] is a framework for querying data stored in distributed data repositories, including the Web. Delaunay^{MM} does not allow an author to build a new multimedia document containing retrieved data, but it offers a number of functionalities for presenting multimedia data retrieved by a query to a multimedia repository. Query answers are organized into presentations, and profiles are used to generate user-defined layout of a document and ad hoc querying capabilities to search each type of media item. Delaunay^{MM} addresses the specification of spatial layout, but it does not address the problem of the temporal synchronization of media objects.

In [1] Adali et al. present a process algebra for querying multimedia presentation databases. The algebra can be used to locate presentations with specific properties but also for combining portion of different presentations by retrieving objects from them. A multimedia document is represented by a tree, whose branches describe all the possible presentation sequences. Differently from our approach, the authors

cannot create a presentation from scratch, but can only select a path in the tree of an existing presentation and derive from it a new document. In our approach the author queries a database of media and combines the retrieved media into a new presentation defined by a schema.

In [3], André presents a completely different approach to the problem of automatic generation of multimedia documents, based on concepts already developed in the context of natural language processing. The author considers the generation of multimedia presentations as a goal-directed activity. The input is a communicative goal with a set of parameters, like target audience and language, resource limitations and so on. The planning component of the system selects a multimedia presentation structure on the base of some communicative rules, and retrieves elementary objects like text, graphics or animations. The temporal behavior is expressed by temporal relations similar to the ones defined by Allen [2] and by metric (in)equalities.

4. Dynamics definition in multimedia presentations

A graph is a visual representation commonly used for describing the temporal behavior of a multimedia presentation. In [7] we have defined an event-based synchronization model among continuous and non continuous media in a multimedia presentation. The model is oriented to designing and prototyping multimedia presentations rather than to providing an execution language like, e.g., SMIL [14]. It is a good trade-off between expressiveness and simplicity, and is targeted to a class of multimedia presentations we have called "video-centered" presentations, where one or more continuous media set the time base for synchronizing other static and dynamic media.

The reader is referred to the cited work for a discussion of motivations and details about the model, whose main properties only will be recalled in this paper.

A multimedia presentation is a collection of *media objects* whose behaviors are described by a set of synchronization relationships established by the author. Each medium requires some device to be rendered or played, such as a window, a frame, an audio channel, or a combination of audio and video resources (as required by a video file with integrated audio). Such a virtual device is called *channel* in the model. It is used by the medium for the whole duration of its playback.

Five synchronization primitives define object reactions to events, both *internal* (e.g., the natural end of a media item) and *external*, like user interactions.

The relation "*a plays with b*", written $a \Leftrightarrow b$, models the parallel composition of media objects a and b: it states that if one of the two objects is activated by the user or by some other event, the two objects play together. Relation "*plays with*" ($a \Leftrightarrow b$) is asymmetric, object a acts as a "master": when it ends, object b is terminated too, if it is still active.

The relation "*a activates b*", written $a \Rightarrow b$, models the sequential composition of two objects: when object a naturally ends, object b begins its playback. These two relations are similar to the tags <`par`> and <`seq`> of SMIL but some differences exist that are detailed in [7]. In particular, we distinguish between *internal* events

which are generated by components of the presentation and *external* events, which are generated by the user, separating the *natural termination* of an object, occurring when it reaches its ending point, from its *forced end*, occurring when the user stops it. In the relation $a \Rightarrow b$, if the user stops object a, object b is not activated. In the same situation, if the relation $a \Leftrightarrow b$ holds, the object b is not terminated.

The relation "*a is replaced by b*", denoted by $a \rightleftharpoons b$, is mainly used with static objects whose time duration is potentially infinite. It states that starting object b forces a to end, so its channel is released and can be used by b.

Two other relations model object reactions to user interactions. The relation "*a terminates b*", written $a \Downarrow b$, terminates two objects at the same time as a consequence of the forced termination of object a. The relation "*a has priority over b with behavior* α", symbolically written $a \overset{\alpha}{>} b$, means that object b is paused (if $\alpha = p$) or stopped (if $\alpha = s$) when object a is activated; a is supposed to be the target of a hyperlink that moves the user focus from the current document (b) to another document or to another presentation.

Besides stopping a medium item or following hyperlinks, the user can interact with a multimedia presentation in other ways. If the user moves back or forward along the presentation timeline, the model handles two separate events: a *stop* event at the current point of the presentation playback and a *start* event in the new position. All the synchronization relationships involved with stopping the current medium and starting the target one are activated. A *pause* event simply "freezes" the running media (therefore pauses the presentation playback) until the user issues a *resume* event.

It must be noted that our model describes media synchronization based on discrete events; fine-grain synchronization (like lip-synchronization) cannot be defined in such a way, bust must be built into a compound media stream which is then handled as a single medium item. For the same reason intra-medium events are generally not supported, but a continuous media stream can be divided in sequential fragments (called *scenes*) in order to define intermediate synchronization points during playback.

A visual authoring tool supports the author in defining the temporal relationships between media by drawing a graph where the nodes are the media objects and the edges the synchronization relationships [8]. A player interprets the synchronization schema and runs the presentation.

A comparison with other multimedia synchronization models would go beyond the goal of this paper, and can be found in [7]. We comment briefly here about two popular and standard models, Allen's relations and SMIL, to give the reader a glance of the main differences which justify our approach.

Allen [2] defines a set of thirteen relationships between temporal intervals of known length. In our model the length of a media object is the time span from its beginning to its natural end, but its actual duration is known only at run-time, since synchronization relationships can modify the object behavior with respect to its natural playback. Allen's model captures the relationships between two media

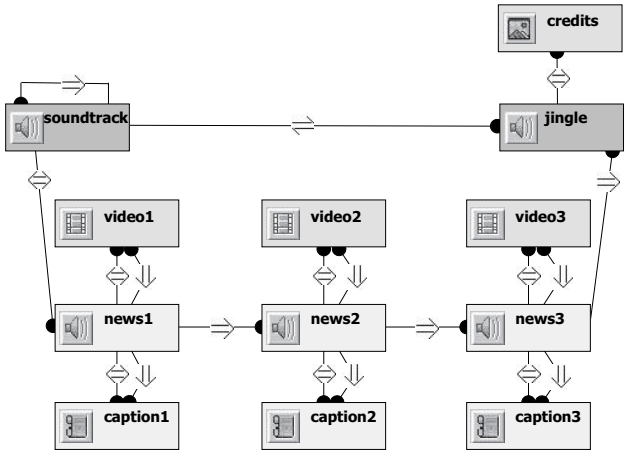


Fig. 1. A simplified synchronization graph for a news-on-demand presentation.

items when their execution is known, therefore cannot be used as a design aid.

The main difference between SMIL [14] and our model concerns the lack of a reference model for the data structure in SMIL. Our model organizes media objects into a hierarchical structure, which is useful to design complex presentation. The XML language which will be presented in Section 6, clearly separates spatial and temporal relations from references to media objects in three separate sections. A media item can be referenced several times without redundancy by addressing its *id*. Thus, an author can easily reuse the structure of another presentation, a single media item, an entire document or a part if it. In SMIL, instead, the two types of information are interleaved in the document, possibly generating redundancy.

Other differences between SMIL and our model concern the way actions directed to end media executions are managed. Like Allen's relationships, SMIL native features do not distinguish between natural and forced termination of a media, therefore the effects of a user interaction on a single media component cannot in general be described.

Figure 1 illustrates a graph showing the synchronization schema of a simplified news-on-demand cover made of three articles, in a graphic style very close to the one used by the authoring system.[a] A background soundtrack plays continuously (when it ends it is activated again by virtue of relation $soundtrack \Rightarrow soundtrack$). The articles are played in sequence, and each article is made of a spoken narration ($news_i$), a video ($video_i$) and a text caption ($caption_i$). All the components of an article play in parallel as described by relationships $news_i \Leftrightarrow video_i$ and $news_i \Leftrightarrow caption_i$ (the dot at the end of the edge connecting two media denotes the dependent medium). The length of each article is controlled by the length of the narration, which is the *master medium* ruling the parallel play of the other two media. At the

[a]Small differences are introduced to enhance the readability, and concern mainly the composites which will be discussed in the next Section

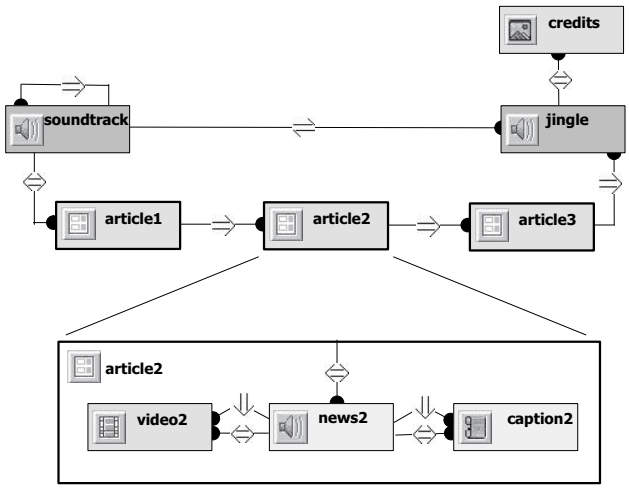


Fig. 2. A modular news-on-demand presentation.

end of each spoken narration, the relation $news_i \Rightarrow news_{i+1}$ activates the next one.

If the user stops the news playback, he or she stops the master medium, i.e., $news_i$, which also stops the video and text caption due to the relations $news_i \Downarrow video_i$ and $news_i \Downarrow caption_i$. When the last article ends, the soundtrack is replaced by a jingle ($soundtrack \rightleftharpoons jingle$) and a credits screen is displayed ($jingle \Leftrightarrow credits$).

Each media object is associated to a *channel* represented with a distinctive color in the graph.

5. Defining multimedia reports

In the graph of Figure 1 a recurring pattern is immediately perceivable: each article has the same components, i.e., a spoken comment, a video and a text caption, and the three articles have a common behavior. Figure 2 makes the recurrence more visible by introducing a compound media item for each article, which we call a *composite*, whose details can be hidden at a high level of specification. A composite, drawn with a thick border in order to distinguish it from atomic media items, is a kind of *envelope* enclosing several media items mutually synchronized which behaves at a high level of observation as a compound media item, starting and ending according to the same synchronization rules which hold for atomic media items. In particular, the composite ends when all the media items enclosed are no longer active. More formally, if *Media* is the set of media items contained in a composite and *Active* is the set of active media, the composite is ended if $Media \cap Active = \emptyset$. If the last event occurred is $end(m)$ where m is an atomic media item, $m \in Media$, the composite ends naturally, otherwise it is assumed to be stopped.

From the synchronization schema of Figures 1 and 2 a *template* for a multimedia

report which displays selected news in sequence can be derived straightforwardly.[b] The template of a multimedia report can therefore be defined by linking nodes of a graph (the object placeholders) with labelled edges (the temporal relations).

The template gives an intensional definition of the presentation. It does not detail the retrieved objects involved in the presentation, since the cardinality of the media set returned by querying the repository is unknown till execution. Therefore some nodes of the graph are placeholders for a collection of concrete media items with the same characteristics, while other nodes denote media items which do not depend on queried data. The drawing notation must make evident which are the items which build up the repeated media groups, and the schema editor must provide the author a means to draw the structure of a report specifying which part of the structure is a replicated group, the relations inside a group and between different instances of the replication. The concept of composite is used to specify such repeated groups: to distinguish between report templates and synchronization schemas describing presentations, a composite denoting a repeated group in a report template is called a *stencil*.

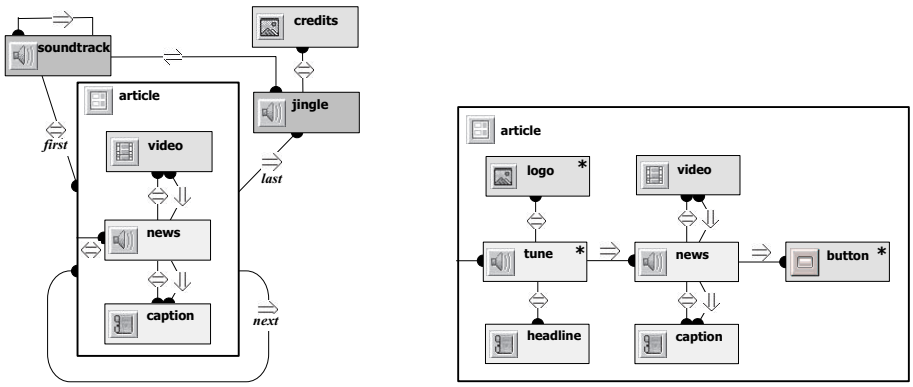
Figure 3a shows a report template for the news-on-demand example of Figure 2. The stencil encloses the media placeholders which make up a repeated element (i.e., an *article*), specifying which events are generated and which synchronization relationships are obeyed. A stencil may contain also media items which do not depend on query results, but are simply replicated once for each tuple returned: such items are denoted with a star in the upper right corner. In Figure 3b a richer article structure is shown: while retaining the synchronization between the voice comment, the video and the text of Figure 3a, each article instance is preceded by the article headline together with the TV channel logo and musical tune, and is followed by a "*next*" button which allows the user to step through the news. The button is a dynamic media item which ends when the user clicks on it (see [7] for details). The logo, the musical tune and the button are repeated for all the news but do not change their content.

The stencil is used to instantiate replicated groups. Relations which involve the stencil can be labelled with a value denoting which tuple of the result is affected by the relation: the first, the last, the next, or all the tuples if the relation is unlabelled.

The execution of the replicated instances of a stencil is subjected to the following rules:

1. The first instance is executed according to the synchronization relationship labelled with the label *first* (in Figure 3a, the *plays with*($\Leftrightarrow$) relation with the soundtrack, which means that the media items enclosed in the stencil start playing with the soundtrack), and the composed media are synchronized as described by the stencil details.

[b] From now on we shall use consistently the term *schema* for denoting synchronization schemas of presentations, and the term *template* for denoting synchronization schemas for report templates. Even if both represent the synchronization among media items (or placeholders), such a distinction in the terminology will help the reader to focus the proper context.



(a) (b)

Fig. 3. (a) A visual template for a news report, (b) An article with placeholders and media items.

2. The instance execution ends according to the synchronization schema described in the stencil, and the end event is propagated out of the stencil according to the relationship which links the stencil instance to the next one, labelled with the label *next*. Then the next instance is started according to the same synchronization schema; in Figure 3a each of the following narrations starts with the related video and caption.

3. When the last instance of the stencil ends, the end event is propagated as described by the relationship labelled with the label *last*. In the example of Figure 3a, the soundtrack is replaced by a jingle, the credits screen is displayed, and the presentation ends.

Since a stencil (as well as a composite) masks the details of the internal media and placeholders, synchronization relationships cannot be established between media items outside the stencil and media items and placeholders inside.

Such a representation is used by the visual authoring system and is supported by an XML-based language for defining the data structure and relationships in a more suited machine processable representation.

6. An XML-based schema definition for multimedia reports

The structure and the temporal behavior of a multimedia presentation are described by an XML schema, based on the same synchronization model underlying the graph representation.

In order to support content independent processing of the presentation structure the schema, besides keeping multimedia data separated from the structure definition (which is quite obvious), also divides structure related information from spatio-temporal information. In other models such information is often mixed. For

example, in SMIL spatial information is declared separately in the `head` section, but the synchronization definition includes the declaration of the media objects. Such integrated definition does not encourage object re-use, mainly in complex documents where it would be especially useful. Redundancy is generated, which requires cross-checking between different document sections.

Multimedia presentations and report templates have many features in common, they only differ for the presence of stencils and placeholders. Therefore, the XML schema supports both types of documents. An XML source document contains three types of specifications: the spatial layout of the document, the media involved in the presentation and their temporal behavior. Data is organized in three sections: the *layout section*, the *components section* and the *relationships section*. This solution enables the definition of spatio-temporal relationships among media objects without knowing any information about their location or duration, and makes it simpler to draw a report template which can be instantiated with minimal modifications of the XML file.

The author defines the temporal behavior by addressing abstract media object identifiers (i.e., placeholders of actual data) rather than actual instances. The system will bind object identifiers, defined in the component section, to actual media objects after retrieving the data. The final presentation is rendered by processing the XML file and accessing media objects which are located elsewhere.

6.1. *The Layout section*

The `layout` section contains the definition of the spatial layout of media in the presentation window. The presentation layout is organized in channels, which are a combination of a portion of the user screen, hosting some media, and audio devices to play soundtracks or other audio files.

Figure 4 shows the XML template (i.e. the XML specification describing a template) of a news-on-demand multimedia report modelled according to Figure 3a. The channels `video` and `caption` are rectangular areas of the user screen delimited by the corner coordinates `SupX`, `SupY`, `InfX` and `InfY`. The captions contain textual information about the news report. `Voice` and `sound` are audio channels, therefore they have no layout. Each channel has a unique `name`.

6.2. *The Components section*

The `components` section contains the description of the media objects involved in the presentation, their types, links to media files, channels used, etc. Continuous media objects, i.e., videos, animations and audio files, are called `clips`; static media objects, i.e., text pages and images, are called `pages`.

Each element has a unique identifier `id`, which is used to reference the object from the other sections of the document, a `type` and a `channel` in which it is played.

If the XML specification describes a presentation, the clips and the pages have an attribute `file` whose value is the path of the media files. If the XML specification

```
<presentation xmlns="report.xsd">
 <layout width="500" height="400" >
  <channel name="video" SupX="19" SupY="13" InfX="471" InfY="321"/>
  <channel name="caption" SupX="19" SupY="324" InfX="471" InfY="391"/>
  <channel name= "voice"/>
  <channel name= "sound"/>
 </layout>

 <components>
  <module id="news_report">
   <clip id="soundtrack" file="sound.wav" channel="sound" type="audio"/>
   <stencil id="article">
     <clip id="video" channel="video" type="video" />
     <clip id="news" channel="voice" type="audio" />
     <page id="caption" channel="caption" type="text" />
   </stencil>
   <clip id="jingle" file="jingle.wav" channel="sound" type="audio"/>
   <page id="credits" file="credits.txt" channel="video" type="image"/>
  </module>
 </components>

 <relationships>
  <play>
   <master><cont_object id="soundtrack"/></master>
   <slave><ref_stencil id="article" num="first"/></slave>
   <master><ref_stencil id="article"/></master>
   <slave><cont_object id="news"/></slave>
   <master><cont_object id="news"/></master>
   <slave><object id="video"/></slave>
   <master><cont_object id="news"/></master>
   <slave><object id="caption"/></slave>
   <master><cont_object id="jingle"/></master>
   <slave><object id="credits"/></slave>
  </play>
  <act>
    <ended><cont_object id="soundtrack"/></ended>
    <activated><cont_object id="soundtrack"/></activated>
    <ended><ref_stencil id="article"/></ended>
    <activated> <ref_stencil id="article" num="next"/> </activated>
    <ended> <ref_stencil id="article" num="last"/> </ended>
    <activated> <object id="jingle"/> </activated>
  </act>
  <repl>
      <before> <object id="soundtrack"> </before>
      <after> <object id="jingle"> </after>
  </repl>
  <stop>
     <first><object id="news"/></first>
     <second><object id="video"/></second>
     <first><object id="news"/></first>
     <second><object id="caption"/></second>
  </stop>
 </relationships>
</presentation>
```

Fig. 4. XML schema for a news report

```
<presentation xmlns="model.xsd">
 <layout width="500" height="400" >
  <channel name="video" SupX="19" SupY="13" InfX="471" InfY="321"/>
  <channel name="caption" SupX="19" SupY="324" InfX="471" InfY="391"/>
  <channel name= "voice"/>
  <channel name= "sound"/>
 </layout>

 <components>
  <module id="news_report">
   <clip id="soundtrack" file="sound.wav" channel="sound" type="audio"/>
   <composite id="article1">
    <clip id="video1" channel="video" type="video" />
    <clip id="news1" channel="voice" type="audio" />
    <page id="caption1" channel="caption" type="text" />
   </composite>
   <composite id = "article2"> ....</composite>
   <composite id = "article3"> ....</composite>
   <clip id="jingle" file="jingle.wav" channel="sound" type="audio"/>
   <page id="credits" file="credits.txt" channel="video" type="image"/>
  </module>
 </components>

 <relationships>
  <play>
   <master><cont_object id="soundtrack"/></master>
   <slave><cont_object id="article1" num="first"/></slave>
   <master><cont_object id="article1"/></master>
   <slave><cont_object id="news1"/></slave>
    ...
   <master><cont_object id="news1"/></master>
   <slave><object id="caption1"/></slave>
    ...
   <master><cont_object id="jingle"/></master>
   <slave><object id="credits"/></slave>
  </play>
  <act>
   <ended><cont_object id="soundtrack"/></ended>
   <activated><cont_object id="soundtrack"/></activated>
   <ended><cont_object id="article1"/></ended>
   <activated> <cont_object id="article2"/></ended>
   ...
  </act>
  <stop>
    <first><object id="news1"/></first>
    <second><object id="video1"/></second>
    <first><object id="news1"/></first>
    <second><object id="caption1"/></second>
    ...
  </stop>
  <repl>
     <before> <object id="soundtrack"> </before>
     <after> <object id="jingle"> </after>
  </repl>
 </relationships>
</presentation>
```

Fig. 5. The generated presentation in XML

describes a report template, pages and clips definitions are placeholders for retrieved items, therefore the attribute `file` will be added at presentation instantiation time.

The media can be defined inside a tag `stencil`, which represents a stencil which builds up a report repeated item. In Figure 4, the clips `soundtrack` and `jingle`, and the page `credits` refer to media objects which do not depend on the report instantiation (therefore the attribute `file` is defined), while the `stencil` section represents the thick rectangle of Figure 3a, which will be instantiated on the results of the query execution. In Figure 4, the clip `video` represents the set of videos returned for the selected news, `news` is the set of voice comments and `caption` is the set of text pages related to the same news.

6.3. *The Relationships section*

The `relationships` section describes the temporal behavior of objects through a list of synchronization primitives needed for the correct playback of the presentation.

The tags `play` and `act` define the basic relationships of parallel and sequential synchronization of media objects that in the visual representation are denoted by the symbols $\Leftrightarrow$ ("*plays with*") and $\Rightarrow$ ("*activates*"). The tag `play` defines the parallel execution of a `master` and a `slave` object. The tag `act` defines the sequential composition of objects `ended` and `activated`. Both the `master` and the `ended` objects of the relationships `play` and `act` must be continuous media, since static media have no defined duration.

The relationship "*is replaced by*" ($\rightleftharpoons$) is encoded with tag `repl` which defines sharing of a same channel between two objects which are active at different times in cases different from simple sequencing. The tags `before` and `after` define respectively the replaced and the replacing objects in the channel usage. In the example the jingle replaces the soundtrack in the same audio channel.

Relationships "*terminates*" ($\Downarrow$) and "*has priority with behavior* α" ($\overset{\alpha}{>}$) are respectively translated with tags `stop` and `link`. Tag `stop` models the synchronous stop of the second object in the relation (`second`) when the other object (`first`) is forced to end. Tag `link` defines the behavior (coded in the attribute `behaviour`) of the source object (tag `from`) when the destination object (tag `to`) starts playing.

In an XML template, relationships can be established between media objects and stencils. In a such case, the tag `ref_stencil` is used. Relationships between stencil instances must be carefully evaluated during the template and data integration phase. The attribute `num` identifies which instance of the stencil is referred: the `next` instance, the `first` or the `last` instance.

7. Template and data integration

Once media objects are collected from query results, the template is instantiated on the objects retrieved in order to generate the actual multimedia report. Figure 5 shows the XML description of the presentation described in Figure 1. The transformation of a report template into a presentation is performed by the procedure **FILL** which reads the XML file of the report *template* and writes the resulting

presentation into *report*. For simplicity the code assumes a correct XML template of the report, therefore it lacks any error checking and diagnosing feature.

```
FILL(template, report: file, RS: mapping function)
// template: file which contains the XML template of the report,
// report: XML file which will contain the presentation computed from the template,
// RS: function which returns ∀ stencil the set of data returned by the query

begin
  line = template.readline();                  // first line is namespace
  line = replace(line, "report.xds", "model.xds");
  while line ≠ "</layout>" do                  // first section is layout
    begin                                      // copy without changes
      copy(line, report);
      line = template.readline()
    end;
  copy(line, report);                          // copy "</layout>"
  line = template.readline();                  // read next line
  while line ≠ "</components>" do              // next section is components
    begin                                      // process components section
      if line.contains("stencil") then
        begin
          id_stencil = attribute(line,"id");
                                               // replicate a stencil for all data tuples
          FILLSTENCIL(template, report, line, RS(id_stencil), num, stencil)
        end
      else copy(line, report);                 // copy data item outside the stencil
      line = template.readline()
    end;
  copy(line, report);                          // copy "</components>"
  line = template.readline();                  // read next line
  while line ≠ "</relationships>" do           // last section is relationships
    begin                                      // process relationships section
      if beginRelation(line) then              // copy or replicate the relationships
          FILLRELATIONS(template, report, line, num, stencil)
      else copy(line, report);                 // copy the lines with the relation type
      line = template.readline()
    end;
  ... copy to end of template
end
```

The layout section of the XML document is not affected by report instantiation, which involves the objects and their relationships, but does not modify the layout. Only the reference to the `namespace` needs to be modified, since the XML template refers to the namespace defined for report templates, while the XML presentation addresses the one defined for multimedia presentations.

The `components` section must be extended to address the objects retrieved. Media outside the stencils remain unchanged, while each stencil is replaced by a composite which contains the concrete media objects returned by the query. In our example the returned set is

$$RS(id_{article}) = \{(video_i, news_i, caption_i) \mid 0 < i \leq |RS(id_{article})|\}$$

The `components` section is completed by replacing the stencils with a composite which contains such objects. The composite is replicated $|RS(id_{article})|$ times, and the instances are distinguished by systematically changing the object name placeholders in the template. The attribute `id` is instantiated by appending a sequence number i, and the attribute `file` (if missing) is added to each object, referring the actual media locations. This behavior is described by the procedure **FILLSTENCIL**.

FILLSTENCIL(*template*, *report*: file, *line*: string, *RS*: set of tuples,
num, *stencil*: mapping function)
// *template*: file which contains the XML template of the report,
// *report*: XML file which will contain the presentation computed from the template,
// *line*: string which contains the last line read from file *template*,
// *RS*: the set of returned media for this stencil,
// *num*: function returning for each stencil the number of replications,
// *stencil*: function returning for each element the containing stencil

```
begin
  elem = 0; num_c = 1; id = attribute(line,"id");
  composite ="";
  line = readStencil(template, line);        // read the whole stencil
  while RS ≠ ∅ do                            // while the query returns some data
    begin
      pick next tuple from RS;               // each stencil becomes a composite
      composite = replace(line,"stencil","composite");
                                 // append to attribute "id" a sequence number
      composite = append(composite,"id", num_c);
      for all element in composite do        // insert attribute "file" if missing
                                             // if element depends on query results
          if attribute(element,"file") = null then
              composite = add(element,"file", tuple(elem++));
      copy(composite, report);
      elem = 0; num_c++
    end;
  for all element in line do
      stencil(element) = id;                 // element is contained in stencil id
  stencil(id) = id;                          // id is itself a stencil
  num(id) = |RS|                             // num(id) is the number of replications
end
```

The `relationships` section is processed similarly to the `components` section. Each single relationship is processed by the procedure **FILLRELATIONS** which controls if the media involved are stencils, media placeholders or actual media items. Relationships between objects outside the stencil are copied unchanged in the new file, while relationships between objects inside a stencil are replicated by the procedure **SAME**.

```
SAME(report: file, relation: string, iter: integer)
// report: file which contains the XML report,
// relation: string containing the whole relation,
// iter: number of iteration

  begin
    for i = 1 to iter
      begin
        line = append(relation, "id", i);      // append sequence number to attribute "id"
        copy(relation, report)                 // copy line with correct indexes
      end
  end
```

The management of relationships which involve stencils and other objects is a bit more complex. With reference to Figure 3a, a stencil can be both the origin and the end of a dynamic synchronization relation with a media item of a placeholder. The attribute `num` of the template definition points out which tuple of the resulting set is affected by the relationship. If `num = next` the relationship must be defined between each resulting composite and its successor as described by the procedure **NEXT**. If it is not present, the relationship must be replicated for any tuple of the set, otherwise it involves only the selected tuple.

```
NEXT(report: file, relation: string, iter: integer)
// report: file which contains the XML report,
// relation: string containing the whole relation,
// iter: number of iteration

  begin
    for i = 1 to iter − 1                  // for each instance
      begin                                // instantiate relation with next instance
        relation = appendN(relation, "id", i, 1);
        relation = appendN(relation, "id", i + 1, 2);
        copy(relation, report)
      end
  end
```

```
FILLRELATIONS(template, report: file, line: string,
                        num, stencil: mapping function)
// template: file which contains the XML template of the report,
// report: XML file which will contain the report after the computation,
// line: string which contains the last line read from file template,
// num: function which returns for each stencil the number of replications,
// stencil: function which returns for each element the containing stencil
  begin
    relation = readRelation(template, line); // read the whole relation
    id_A = attributeN(line, "id", 1);        // attribute "id" of the first object
    id_B = attributeN(line, "id", 2);        // attribute "id" of the second object
                  // replace all occurrences of "ref_stencil" with "cont_object"
    relation = replace(relation, "ref_stencil", "cont_object" );
```

```
if stencil(id_A) = null then                    // A is not in a stencil
  begin
    if stencil(id_B) = null then                // B is not in a stencil
        copy(relation, report)
    else                                        // B is a stencil, stencil(id_B) = id_B
      begin                                     // remove attribute "num"
        attr_num = attribute(relation, "num");
        relation = remove(relation, "num");
        case attr_num ="first":
          begin                                 // append 1 to 2nd attribute "id"
            relation = append_n(relation,"id", 1, 2);
            copy(relation, report)
          end;
        case attr_num ="last":
          begin                                 // append # of iterations to 2nd attribute "id"
            relation = append_n(relation,"id", num(stencil(id_B)), 2);
            copy(relation, report)
          end;
                                                // attr_num ≠ null here
      end                                       // end B is a stencil
  end;                                          // end A is not in a stencil

if stencil(id_A) ≠ null and stencil(id_A) ≠ id_A then
  begin                                         // A is a placeholder in a stencil
    if stencil(id_B) ≠ null and stencil(id_B) ≠ id_A then    // B is in a stencil
                                                // copy relation for all stencil instances
        SAME(report, relation, num(stencil(id_A)))
    else                                        // B is a stencil, stencil(id_B) = id_B
      begin                                     // remove attribute "num"
        attr_num = attribute(relation, "num');
        relation = remove(relation, "num');
        case attr_num ="next":                  // copy relation for all stencil instances
            NEXT(report, relation, num(stencil(id_A)));
        case attr_num = null:                   // copy relation for all stencil instances
            SAME(report, relation, num(stencil(id_A)))
      end                                       // end B is a stencil
  end;                                          // end A is a placeholder in a stencil

if stencil(id_A) = id_A then                    // A is a stencil
  begin
    if stencil(id_B) = null then                // B is not in a stencil
        ... symmetric to A not in a stencil and B stencil
    else                                        // B is a stencil or inside a stencyl
      begin                                     // attribute "num" of the 1st object
        attr_num_A = attribute(relation, "num", 1);
                                                // attribute "num" of the 2nd object
        attr_num_B = attribute(relation, "num", 2);

        if attr_num_A ≠ null and attr_num_B ≠ null then
          begin                                 // instantiate attribute "id" as above
            relation = remove(relation, "num");
```

```
            case attr_numA ="first":
                relation = append_n(relation,"id",1,1);
            case attr_numA ="last":
                relation = append_n(relation,"id",num(stencil(id_A)),1);
            case attr_numB ="first":
                relation = append_n(relation,"id",1,2);
            case attr_numB ="last":
                relation = append_n(relation,"id",num(stencil(id_A)),2);
            copy(relation, report)
          end

        else                                  // attr_numA or attr_numB = "next",
                                              // or both A and B are stencils
        ...equal to A in a stencil and B is a stencil
      end                                     // end B is a stencil or in a stencil
    end
  end
```

In the example described in Section 4, the soundtrack play starts the execution of the first stencil instance. Since the attribute `num` is present with value `first`, the relationship must be evaluated only once, and refers to the first instance of the stencil, therefore to the first composite $article_1$. Each stencil instance starts playing the audio file instantiated for the `news` item, since the attribute `num` is not defined in the relation `play` between `article` and `news`. The object `news` plays the role of a master, since it starts the video and the caption, and stops their playback when ending. Its ending coincides also with the ending of the composite.

Relationships between the objects inside the stencil are replicated for all the instances.

At the end of the last stencil instance, the relation `act` between `article` and `jingle` is translated only once between `article3` and `jingle`, according to the value of the `num` attribute.

An `act` relationship exists between the two stencil instances, specifying at both ends a stencil *article*. The relation must be replicated for all the tuples of the resulting set, i.e., for all composites, since the attribute `num` is not defined in the first element of the presentation, but assumes the value `next` in the second element. The relations instantiated are therefore

$$article_i \Rightarrow article_{i+1}, 1 \leq i \leq 2$$

making the generated presentation to play sequentially all the articles.

8. The authoring environment

Presentations defined using the model addressed in this paper are supported by an authoring environment called *LAMP* (LAboratory for Multimedia presentations Prototyping) which allows an author to set up, test and execute a complex multimedia presentation by specifying the media items involved and the synchronization relationships among them.

The authoring system components are a visual editor, an execution simulator to test the presentation behavior on different media-related and user-related events, a

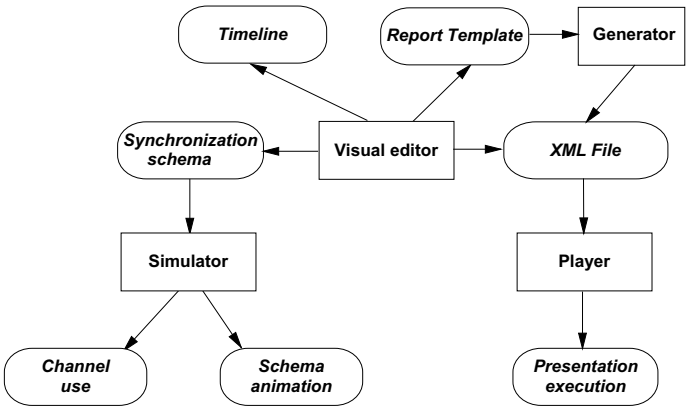


Figure 6: The *LAMP* authoring environment

generator for integrating variable data in a report template, and a player for the execution of the final presentation (Figure 6). The generator of multimedia reports implements the algorithms presented in Section 7. A detailed description of the other components of *LAMP* is in [8].

The editor allows the authors to build both multimedia presentation schemas and report templates by adding media objects (nodes) and synchronization relations (edges) to a graph. Screen layout and playback channels are visually arranged by drawing rectangles inside the presentation window.

An execution simulator is very useful to test a synchronization schema, since it allows the author to check the temporal behavior of the presentation without requiring the actual media file to be available (e.g., in case of report templates): placeholders are allocated into the channels the corresponding media would use in the real execution. Then, without being compelled to follow a real time scale, the author can generate all media related events, both internal (e.g., the end of an object play), and external (e.g., a user-executed stop or a hyperlink activation) to see how the synchronization relationships are fired and how the presentation evolves.

In order to help the author to understand the relationships between events and media, the simulator animates also the synchronization graph: when the user activates an object, the corresponding node in the graph is highlighted. The relations triggered by the activation of the object are also highlighted, and their effect is propagated to the related objects, giving the author, in a short animation, the visual perception of how media are synchronized.

Figure 7 shows the interface of the authoring tool when executing a simulation step during the development of the example presentation illustrated in Section 4. In Figure 7a the presentation is playing the first news $news_1$, with the associated media $video_1$ and $caption_1$. Active objects are highlighted with thick borders, and in the preview window a static view of the media is displayed, with their names. The author can simulate the occurrence of a media-related event by selecting it in a list. In Figure 7a the end of medium $news_1$ is fired, and Figure 7b shows the result: the graph highlights the second instance of the news, and the preview panel

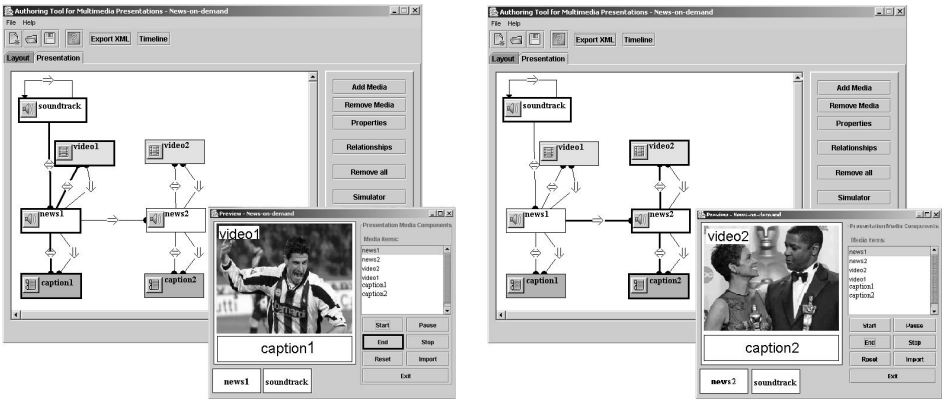


Figure 7: The visual interface of the *LAMP* authoring tool

is updated.

With such a visual representation the author can always check which part of the presentation he/she is simulating, which are the active media, why the presentation behaves as observed, and how the end-user interface is organized.

The visual editor generates the XML-based description which is used by the player for the presentation playback. This representation can be translated into a SMIL file even if this translation is not always possible (details are described in [8]). SMIL does not cover all the temporal constraints imposed by the five synchronization primitives defined in our model. In particular, SMIL does not deal with the forced termination of an object; most commonly used SMIL players allow the user to start, stop, pause and resume a presentation, and not the component media items alone. Even if the translation of a single primitive is quite easy, it is not trivial to define a general rule for the translation of a complex presentation. For example, the relationship *soundtrack* $\Rightarrow$ *soundtrack* cannot be translated using the SMIL tag `seq` (which naturally maps the relationship *act*) since the soundtrack should repeat continuously, and not only twice. The translation rule must consider the context in which the relationship is set, discover that it defines a media loop, and generate consequently a different SMIL code. Due to the nested structure of a generic presentation (and of a report template) such a context can be very broad; a step by step translation based on nesting could unfeasible in the general case.

If the author defines a report template, the corresponding XML file can be elaborated by a generator to produce the final presentation. We assume, at the current stage of implementation of our system, that the retrieval system stores the URLs of the media objects returned by the query in a text file. The generator integrates the report template with the query results according to the algorithm described in Section 7, producing the XML file of the report.

LAMP also provides a player which is able to read the XML file of a presentation and to deliver it to final users. It can be used as a stand-alone application or during the authoring phase, since it can interact with the simulator by visually relating the execution evolution with an animation of the synchronization schema.

The LAMP environment is implemented in Java. Two libraries supported the development of specific components:

- the Java Media Framework API (JMF) [12] 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.

- the Swing library [13], a fully-featured library to implement windowing functionalities, was used to implement the graphical user interface.

9. Handling incomplete results

As discussed in Section 2, the ordered set of tuples retrieved by the query $RS(id_{article}) = \{(video_i, news_i, caption_i) \mid 0 < i \leq |RS(id_{article})|\}$ can contain NULL values, denoting that in some tuple some media item can be missing. This value requires attention, particularly if the missing media item is an object which rules the behavior of the whole presentation. As an example, if the media object $caption_2$ is missing in the second tuple returned by the query, the presentation can continue its playback, simply, the channel assigned to $caption_2$ remains empty. However, if the missing object is $news_2$, the presentation cannot continue after $news_1$ ends.

Recalling a concept we have introduced in Section 4 we call *master* objects of a presentation the items which rule the behavior of the presentation, i.e., the items whose time properties define the presentation timing and advancing. The absence or unavailability of a master object stops the presentation playback. In an automatic generation framework such a behavior is not admissible, therefore master objects must be clearly identified, and their unavailability in a stencil instance must be overcome. Such a problem can be solved in two ways.

First of all, we could modify the XML language for report templates to allow the author to define which media items are required and which are optional, therefore can be missing. Otherwise, we could recognize this type of object by analyzing the synchronization relationships.

If we call C the set of media placeholders defined inside a stencil c, the media placeholder m is the stencil *master object* iff $m \in C$, the synchronization rule $c \Leftrightarrow m$ exists, and $\exists a \in C | m \Leftrightarrow a$. The *master objects* of a presentation are all the media items which instantiate a master object of a stencil.

Once we have identified the master objects, either by looking into the XML file or analyzing the synchronization relationships, we can filter the set of tuples returned by a query according to the following rules:

1. if the tuple does not contain any NULL value or the media item corresponding to the NULL value is not a master object, then the tuple is accepted;

2. if the NULL value corresponds to a master object the tuple is discarded.

A more conservative approach could be to present to the user all data returned by the query. In this case, the NULL value could be replaced by a *timer*, a continuous object with a constant duration [7]. Such a solution plays the available media

objects for a defined time interval, during which the channel associated to the NULL instance is empty, but the presentation runs. Variants of this schema can be implemented, e.g., if the tuple contains other continuous media, the timer duration can be set equal to the longest duration by the player, or a default value can be provided at template design time.

10. Conclusion

We have not discussed in this paper issues related to the query formulation and execution. This is of course a problem of crucial importance, and we do not claim it is easy to formulate formally and to solve. However, effective solutions can be found in the database area where models and technology for dealing with multimedia data exist.

A number of questions must be answered, which however do not interfere with the schema model we have discussed here. We have assumed that data comes from one multimedia database, therefore media instances are naturally related to each other much as in a relational table. What if several data repositories are accessed? This situation seems desirable due to the large number of available media sources. However the problem may become hard to approach for several reasons:

1. Different data repositories can hold data items which are semantically close but very far in their physical properties, e.g., different in video size, or in image resolution, or in audio fidelity. A coherent presentation including elements from all the repositories can be very hard or even impossible.

2. Different repositories can require different query languages, or queries with different parameters, due to the differences in DB schemas. What about result integration?

3. We assume that different types of media be returned. How are different elements related if they come from different databases, so that in principle only the interpretation of content can relate them? How can we "link", say, a video instance to an instance of a text related to the same article but coming from a different repository?

Current technology can help us in approaching some of the points above. Wrappers and mediators [11] can be used to approach the problem of querying and integrating several data repositories with different schemas. Semantic attributes and metadata can be used to identify relevant information in multimedia objects, e.g., according to the MPEG-7 standard [10]. In a multimedia report, however, it is plausible to assume a high degree of homogeneity in the returned data, due to the iterated nature of the media presented to the user.

In approaching automatic generation of presentations therefore we are bound to a set of constraints which make our initial assumptions realistic and effective.

1. We must be able to select coherent data, i.e., data that is semantically related and that can be put in a presentation which is recognizable by the reader as a meaningful document. This problem is present in all the automatic presentation construction systems, and is assumed implicitly.

2. Data can be linked by external keys or equivalent cross-reference information which assure that we can identify related data by testing such information.

3. Data is coherent with respect to physical playback properties.

These requirements are satisfied if we have only one multimedia database. They can be guaranteed to some extent by filtering data coming from different databases using wrappers and mediators, even if it could be hard to assure the needed physical homogeneity in the resulting presentation. We should in this case assume that the report produces a presentation prototype that has to be refined by hand in its visual aspects.

Applications that can be satisfied with these requirements are wide: news-on-demand, that we have used as a scenario in a very simplified view, is a good case, since the assumption that the same database holds news video with associated texts and audio is realistic. Advertisement is another good case, since it is plausible that a set of advertised items can be described each by a picture or a video, a spoken text, a jingle, and so on, related by well identifiable keys. In all cases the multimedia report can be completed with purely aesthetic media such as a background soundtrack, decorative frames, contour images, and so on, which can be described in the report template or added in a subsequent refinement phase.

Acknowledgements

An anonymous referee has made many comments which helped us to improve noticeably this paper. This work has been partially supported by Italian Ministry of Education, University and Research (MIUR) in the framework of the National Project *Specification, Design and Development of Visual Interactive Systems*, and of grants for young researchers.

References

1. S. Adali, M. L. Sapino, and V. S. Subrahmanian. An algebra for creating and querying multimedia presentations. *Multimedia Systems*, 8(3):212–230, 2000.
2. J. F. Allen. Maintaining knowledge about temporal intervals. *Comm. ACM*, 26(11):832–843, November 1983.
3. E. Andrè. *A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text*, chapter The Generation of Multimedia Documents, pages 305–327. Marcel Dekker Inc., 2000.
4. C. Baral, G. Gonzalez, and A. Nandigam. SQL+D: extended display capabilities for multimedia database queries. In *ACM Multimedia 1998*, pages 109–114, Bristol, UK, September 1998.
5. C. Baral, G. Gonzalez, and T. Son. A Multimedia display extension to SQL: Language and Design Architecture. In *International Conference in Data Engineering*, Orlando, FL, USA, February 1998.
6. I.F. Cruz and W.T. Lucas. A Visual Approach to Multimedia Querying and Presentation. In *The Fifth ACM International Conference on Multimedia '97*, pages 109–120, Seattle, WA, USA, November 1997.
7. O. Gaggi and A. Celentano. Modeling Synchronized Hypermedia Presentations. To appear in *Multimedia Tools and Applications*, Kluwer Academic Publ., 2003.

8. O. Gaggi and A. Celentano. A Visual Authoring Environment for Multimedia Presentations on the World Wide Web. In *IEEE International Symposium on Multimedia Software Engineering (MSE2002)*, pages 206–213, Newport Beach, California, December 2002.
9. J. Geurts, J. van Ossenbruggen, and L. Hardman. Application-Specific Constraints for Multimedia Presentation Generation. In *International Conference on Multimedia Modeling 2001 (MMM01)*, pages 247–266, CWI, Amsterdam, The Netherlands, November 5-7 2001.
10. ISO/MPEG. MPEG-7 Standard Overview. *ISO/IEC JTC1/SC29/WG11*, N4980, 2002. `http://mpeg.telecomitalialab.com/standards/mpeg-7/mpeg-7.htm`.
11. Project MIX. The MIX (Mediation of Information using XML) Home Page. `http://www.db.ucsd.edu/Projects/MIX/`.
12. Java Media API. Java Media Framework API. `http://java.sun.com/products/java-media/jmf/`.
13. Java 2 Platform, Standard Edition (J2SE). J2SE Technology. `http://java.sun.com/j2se/`.
14. Synchronized Multimedia Working Group of W3C. Synchronized Multimedia Integration Language (SMIL) 2.0 Specification, August 2001.
15. Dynamo Project. Semi-automatic Hypermedia Presentation Generation (Dynamo) `http://db.cwi.nl/projecten/project.php4?prjnr=74`.
16. L. Rutledge, B. Bailey, J. van Ossenbruggen, L. Hardman, and J. Geurts. Generating Presentation Constraints from Rethorical Structure. In *11th ACM Conference on Hypertext and Hypermedia*, San Antonio, Texas, USA, May 30–June 3 2000.
17. J. van Ossenbruggen, J. Geurts, F. Cornelissen, L. Hardman, and L. Rutledge. Towards Second and Third Generation Web-based Multimedia. In *The Tenth International World Wide Web Conference*, pages 479–488, Toulouse, France, May 1–5 2001.