New Facilities for Presenting and Authoring MPEG-4 Documents

Romualdo Monteiro de Resende Costa, Rogério Ferreira Rodrigues,
Luiz Fernando Gomes Soares
Departamento de Informática – PUC-Rio
Caixa Postal 38.097 –22.453-900 – Rio de Janeiro – RJ – Brasil
romualdo@telemidia.puc-rio.br, rogerio@telemidia.puc-rio.br, lfgs@inf.puc-rio.br

New Facilities for Presenting and Authoring MPEG-4 Documents

Romualdo Monteiro de Resende Costa, Rogério Ferreira Rodrigues,
Luiz Fernando Gomes Soares
Departamento de Informática – PUC-Rio
Caixa Postal 38.097 –22.453-900 – Rio de Janeiro – RJ – Brasil
romualdo@telemidia.puc-rio.br, rogerio@telemidia.puc-rio.br, lfgs@inf.puc-rio.br

Abstract

This paper proposes new features for presenting and authoring MPEG-4 programs using the NCL language. On one hand, the language allows to define relationships among objects inside an MPEG-4 scene and external media objects, including objects in other MPEG-4 scenes. Relationships with several different semantics will also be allowed other than those defined by the standard. An NCL hypermedia formatter integrated with MPEG-4 players supports these new introduced features. On the other hand, the paper also takes advantage of the NCL composite-node template concept, to add new authoring facilities when specifying MPEG-4 documents using the XMT-O language.

1. Introduction

MPEG (*Moving Picture Experts Group*) defines a family of standards for digital encoding audiovisual information. The first standards, MPEG-1 [1] and MPEG-2 [2], focus primarily on aspects related to the encoding and decoding of audio and video media. Different from the audiovisual linear encoding of these previous standards, the MPEG-4 [3] encoding is object-based. Audiovisual scenes are defined as a set of objects (*video, audio, text*, *image*) that are encoded separately. Spatial-temporal relationships among objects are specified defining their position within a scene in a given presentation time.

The object-based audiovisual encoding approach has some advantages: it allows authors to reuse audiovisual material; objects can be encoded using different spatial and temporal resolutions; compression techniques can be individually applied, according to each media characteristics; synthetic objects can be part of a presentation together with other natural media objects[1]; user interactive actions can be supported; and alternative content for adaptive presentation can be specified.

The MPEG-4 information about spatial-temporal and interactive relationships among objects, are encoded in a binary format called BIFS. (*BInary Format for Scenes*) [4]. Specifications based on BIFS are synchronized and multiplexed with object contents, in a data structure formed by streams [5].

BIFS structures a scene as a hierarchy of objects. A scene can be granularly updated by inserting or deleting objects, or by changing object properties, such as their spatial positions. It is also possible to insert or delete animation events and interactive events related to objects in a scene. A scene can also be entirely replaced by another one.

Although there is a flexible support for defining relationships among objects in a same scene, relationships among objects defined in different scenes are limited. This indeed comes from the fact that, although MPEG-4 documents[2] may contain many scenes, only one of them can be played at a time.

Document specification using BIFS is not trivial, hence the MPEG-4 standard proposes two alternative XML-based languages: XMT-A and XMT-O [4]. All existing binary format (BIFS) expressions can be defined using XMT-A without loss of representativeness. As a result, the language has become excessively complex and large. XMT-O has been developed trying to overcome this complexity. Based on SMIL 2.0 language [6], XMT-O uses

[1] MPEG-4 defines natural objects as those captured by digital equipment (*cameras, microphones, etc*). Complementarily, synthetic objects are produced by computer units [3], and may vary from two-dimensional simple objects, such as lines and points, to three-dimensional complex objects, such as facial and body animations.

[2] In this paper, document and program are used with the same meaning, denoting one or more related scenes, composed by objects.

compositions to structure a document. XMT-O compositions, which can contain media objects and other nested compositions, support temporal synchronization semantics, making the authoring task easier. However, this benefit is limited by the simple set of available compositions (*par*, *seq* and *excl*, as in SMIL) [4]. As a consequence, the definition of complex relationships may require compositions with several nesting levels.

This paper proposes extensions to XMT-O through NCL facilities. NCL (*Nested Context Language*) [7] is a modular XML-based language for authoring hypermedia documents that has introduced a number of new concepts to overcome some limitations of existing XML modular languages.

In order to allow generic relationships among MPEG-4 scenes, MPEG-4 documents are modeled as NCL nodes, which can be related with other NCL nodes using NCL links [7]. This approach not only allows the definition of new relationships between MPEG-4 scenes, but also supports multipoint relationships (among multiple objects), as well as relationships among objects in MPEG-4 scenes and objects outside MPEG-4 documents. Aiming at presenting the related MPEG-4 scenes, MPEG-4 players have been incorporated to the set of display tools of the *NCL formatter* (the software component responsible for controlling NCL document presentations).

NCL facilities are also used to define new presentation semantics to XMT-O compositions, expanding the MPEG-4 expressiveness. In order to extend the presentation semantics of the XMT-O set of compositions, semi-complete hypermedia structures, called composite-node templates, are used. In this direction, NCL XTemplate module [7] has been incorporated to the XMT-O modules, originating a new profile for this language, called in this paper XT-XMT-O (XMT-O+XTemplate).

The remainder of this paper is organized as follows. Section 2 discusses the specification of MPEG-4 content as NCL document nodes, and how the presentation of these nodes can be orchestrated. Section 3 introduces XT-XMT-O profile. Section 4 contains the final remarks with a brief discussion of related and future work.

2. Using NCL to Define Relationships among MPEG-4 Contents

The audiovisual-presentation example of Figure 1-2 illustrates some desirable new relationships to be introduced in a MPEG-4 presentation besides those allowed in the MPEG-4 standard. The presentation is composed by three main objects: an MPEG-4 scene, an MPEG-4 program (video) and another non-MPEG-4 video. The MPEG-4 scene is composed by three internal objects (Figure 1-1): the background image (VOP1) and two video objects (VOP2 and VOP3). Assume in the example that, when a user clicks over one of the video objects (VOP2 or VOP3), the corresponding video (VIDEO1 or VIDEO2), defined externally to the MPEG-4 scene, must be displayed. Simultaneously, the MPEG-4 scene presentation must be paused. The MPEG-4 scene is resumed only when the selected external video ends.

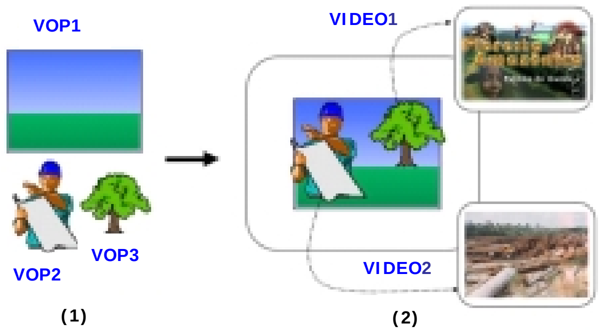

Figure 1. Relationships with MPEG-4 content

Figure 1 thus illustrates a relationship between MPEG-4 scenes that maintains both scenes in exhibition, as well as a relationship between an MPEG-4 scene and an external object not defined in the MPEG-4 document.

Based on the example presented in Figure 1, the following subsections propose two solutions for supporting the new relationships through embedding MPEG-4 content in NCL documents.

2.1. MPEG-4 Scene Objects Modeled as NCL Anchors

The first solution models MPEG-4 content as a continuous media NCL object (NCL video object). Internal objects in a MPEG4 scene are modeled as NCL spatial anchors.

Figure 2 shows the main parts of an NCL document representing the presentation example depicted in Figure 1. Lines 12 to 15 define the video node representing the MPEG-4 scene. Anchors are defined for internal objects subjected to relationships: in this case, VOP2 and VOP3. These anchors are defined through NCL *area* elements.

Two other video objects are specified in lines 16 and 17 of Figure 2, representing videos VIDEO1 (MPEG-4) and VIDEO2 (MPEG-1), respectively.

```
1. <ncl id="NCL" xmlns="Schema" ...>
2.   <head>
3.     <layout>
4.         …….
5.     </layout>
6.     <descriptorBase>
7.        <descriptor id="descMP4" region="mpeg4"
                                   player="MPEG4PlayerAdapter"/>
8.        ……..
9.     </descriptorBase>
10.  </head>
11.  <body>
12.     <video id="mpeg4" src="scene.mp4" descriptor="descMP4”>
13.       <area id="vop2" coords="10,10,50,250"/>
14.       <area id="vop3" coords="80,0,75,200"/>
15.     </video>
16.     <video id="video1" src="video1.mp4"
    descriptor="descVideo1"/>
17.    <video id="video2" src="video2.mpg"
    descriptor="descVideo2"/>
18.     <linkBase>
19.        <link id="link1" xconnector="hyperlinkPauseStart.xml">
20.           <bind component="mpeg4" port="vop3" role="onClick"/>
21.           <bind component="mpeg4" role="pause"/>
22.           <bind component="video1" role="show"/>
23.        </link>
24.        <link id="link2" xconnector="hyperlinkPauseStart.xml">
25.           <bind component="mpeg4" port="vop2" role="onClick"/>
26.           <bind component="mpeg4" role="pause"/>
27.           <bind component="video2" role="show"/>
28.        </link>
29.        <link id="link3" xconnector="finishesMP.xml">
30.           <bind component="video1" role="onPresentationEnd"/>
31.           <bind component="video2" role="onPresentationEnd"/>
32.           <bind component="mpeg4" role="resume"/>
33.        </link>
34.     </linkBase>
35.  </body>
36.</ncl>
```

Figure 2. NCL document embedding MPEG-4 content

NCL links represent relationships among NCL nodes (objects). Lines 19 to 23 specify a multipoint relationship. The *xconnector* attribute of the *link* element (line 19) defines the relation type: in this case, when the node anchor performing the role “onClick” is selected the node performing the role “pause” must have its presentation paused and the node associated to the role “show” must have its presentation started. NCL *bind* elements define the nodes (and in some situations their anchor - through bind *port* attributes) that participate in the relationship, relating them to the connector roles. Thus, this link specifies that when spatial anchor VOP2 is selected, the MPEG-4 scene must be paused and VIDEO1 started.

In NCL, descriptor elements specify how media objects should be presented: defining their exhibition regions in the display device and which tool (player) should be used for their presentation. Line 7 defines the descriptor for the MPEG-4 scene presentation. An MPEG-4 player, discussed afterward, is specified as well as a region “mpeg4” that must be defined in the NCL layout element (not detailed in the figure, but similar to XMT-O layout elements).

In the example, the MPEG4PlayerAdapter player (line 7) represents the MPEG-4 adapter implemented in order to integrate IBM´s M4Play player [8] with the NCL formatter. The NCL formatter implements an extensible architecture where display tools are considered as plug-ins. Thus, an MPEG-4 player becomes just one more formatter display tool, specialized in the presentation of MPEG-4 content.

Some MPEG-4 players are specified in the set of “reference implementations for application development” by the standard [9]. However, these players deal with audiovisual scenes alone. They do not offer interfaces allowing a coordinated integration among multiple player instances, aiming at defining more sophisticated relationships among different MPEG-4 contents. Considering the available MPEG-4 players, IBM´s M4Play has been chosen because of the JAVA API it offers. In order to make the interface provided by the player compatible with the NCL formatter API [10], an adapter was implemented. Among its main functions, the adapter takes care of monitoring MPEG-4 content anchors used in relationships. Selection (mouse click) and presentation events regarding these anchors are signaled to the formatter, while actions related to the content presentation and commanded by the formatter, such as suspend, resume, change-rate, etc., are applied over the MPEG-4 content.

As illustrated in the class diagram of Figure 3, every NCL formatter player (FormatterPlayer class) defines one interface to command the presentation it is in charge of. The NCL formatter allows the same player to control more than one object presentation, each object being modeled by the PlayerObject generic class. The MPEG-4 adapter (MPEG4PlayerAdapter class) has been implemented as a specialization of the generic player class, and therefore it can control one or more MPEG-4 content objects (instances of MPEG4PlayerObject class).

During an NCL document presentation, when an object shall be presented, the player manager is asked to return the proper player. In Figure 2 (line 7), in order to display the video object “mpeg4”, the player manager would return one instance of the MPEG4PlayerAdapter class. Then the NCL formatter would ask the returned adapter to prepare the video object presentation, by calling its prepare() method. Among other actions [10], this method causes the creation of an instance of the MPEG-4 display object (instance of MPEG4PlayerObject class).

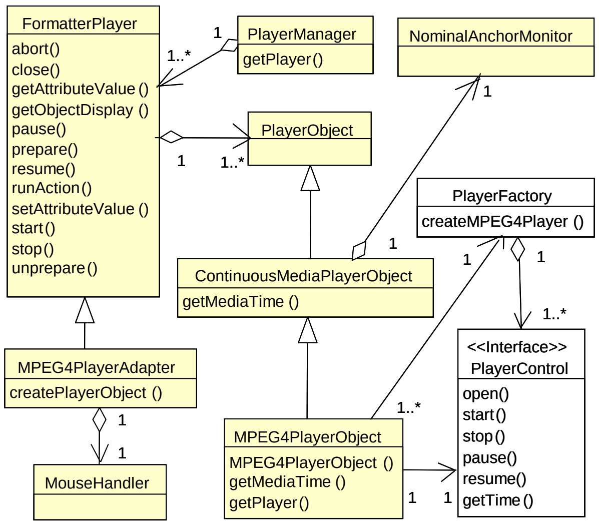

Figure 3. Class diagram for integrating an MPEG-4 player with the NCL formatter

Initially the MPEG4PlayerObject class constructor asks the player constructor implemented in the IBM´s package (PlayerFactory class) for the creation of an MPEG-4 player (M4Play instance). The next step of the preparation consists in passing, to the created MPEG-4 player, the URI of the MPEG-4 file content to be displayed. This is carried out by a call to the open() method of the PlayerControl interface (also defined in the IBM packet).

After preparing the display, presentation commands coming from the formatter, like start, pause, resume and close, are directly mapped from the MPEG-4 adapter to the MPEG-4 player.

The MPEG-4 adapter monitors mouse events (using MouseHandler class), in order to have, allow the NCL formatter to maintain the relationships between MPEG-4 objects and other objects (MPEG-4 or not) specified by the author. If a specific spatial area of the MPEG-4 object is selected, the formatter is notified and evaluates the links that depend on such events. Concurrently, the display object instance monitors the content presentation (using NominalAnchorMonitor class). *Threads* are instanced to observe and signal the occurrence of temporal anchors (a time track of the MPEG-4 object). Through calls to the getMediaTime() method of the media display object, the monitor knows if the display of a specific track has begun or ended, notifying the event to the formatter.

2.2. NCL Modeling of MPEG-4 Sensors and Routes

The solution proposed in the previous section is appropriate when anchors are defined over static objects in the MPEG-4 scene, since anchors are created without any knowledge about the internal data structure of the MPEG-4 content. When objects internal to a scene change their position during presentation the solution may still work, but with great authoring effort. In order to overcome this limitation a second solution is proposed, as follows.

Synchronization relationships in BIFS are generally described using the *timeline* paradigm[3]. Using timeline, the semantics of a relationship (defined by an author using a higher-level language, for example, XMT-O) is lost, although the time positioning for each object is known.

Relations involving user interaction cannot be specified using the timeline paradigm. Thus, in order to define these kinds of relations and to preserve them in MPEG-4 components, routes and sensors [4] are adopted.

An MPEG-4 audiovisual scene is constructed as an acyclic directed graph. In this graph sensor type nodes represent the interaction, and child type nodes represent objects [11]. Nodes classified as sensors generate events, which can be launched by the user interaction with a specific child type node or, occasionally, by scene changes [3]. Events generated in sensors alter their output event type fields (*eventOut*) values. On its turn, changes in these fields may affect the input event field (*eventIn*) or the exposed field (*exposedField*) of other node in the audiovisual scene, changing its value and, consequently, the node properties.

In order to associate the sensor fields to the fields of other nodes, a route must be established between them. In these routes, the sensor scope is limited to its audiovisual scene, justifying why routes can only propagate events between elements belonging to the same scene.

Sensors and routes defined in MPEG-4 components are atypical elements. In order to represent them in NCL documents, sensors can be modeled as attributes of the node representing the MPEG-4 scene, and exported as NCL node attribute-anchors.

As explained in Section 2.1, in order to specify relationships between MPEG-4 components and objects external to a scene, NCL links need to be established. The node anchor encapsulating the attribute, which represents the sensor in the audiovisual scene, and the referred external objects must then be bound to the link connector. NCL anchors representing MPEG-4 sensors can be used either as NCL link

[3] Besides *timeline*, the MPEG-4 optionally supports the FlexTime model [4]. FlexTime allows defining synchronization relationships using Allen relations [12].

conditions as NCL link actions. When used as a condition, sensor changes may trigger document presentation actions (for instance, pause another object presentation). Inversely, when bound to an action, presentation events may cause a value to be assigned to the attribute of the NCL node, reflecting on changes in sensors and, consequently, on the MPEG-4 audiovisual scene.

Figure 4 shows this second proposed solution for the same example of Figure 2.

```
1. <ncl id="NCL" xmlns="Schema" ...>
2.  <head>
3.    <layout>
4.      …….
5.    </layout>
6.    <descriptorBase>
7.        <descriptor id="descMP4" region="mpeg4"
                                 player="MPEG4BIFSAdapter"/>
8.       …….
9.    </descriptorBase>
10. </head>
11. <body>
12.   <video id="mpeg4" src="scene.mp4" descriptor="descMP4">
13.     <attribute name="sensorVop2" />
14.     <attribute name="sensorVop3" />
15.   </video>
16.   <video id="video1" src="video1.mp4" descriptor="descVideo1"/>
17.   <video id="video2" src="video2.mpg" descriptor="descVideo2"/>
18.   <linkBase>
19.     <link id="link1" xconnector="hyperlinkAttributePauseStart.xml">
20.     <bind component="mpeg4" port=" sensorVop3" role="onTrue"/>
21.       <bind component="mpeg4" role="pause"/>
22.       <bind component="video1" role="show"/>
23.    </link>
24.     <link id="link2" xconnector="hyperlinkAttributePauseStart.xml">
25.      <bind component="mpeg4" port="sensorVop2" role="onTrue"/>
26.       <bind component="mpeg4" role="pause"/>
27.       <bind component="video2" role="show"/>
28.     </link>
29.     <link id="link3" xconnector="finishesMP.xml">
30.       <bind component="video1" role="onPresentationEnd"/>
31.       <bind component="video2" role="onPresentationEnd"/>
32.       <bind component="mpeg4" role="resume"/>
33.     </link>
34.   </linkBase>
35. </body>
36.</ncl>
```

Figure 4. NCL document with anchors mapped to MPEG-4 scene objects

As in Figure 2, the MPEG-4 scene is defined in lines 12 to 15, encapsulated as an NCL video node. Node interface points (*attribute* element) are specified and related to MPEG-4 internal sensor nodes. At this point it is important to mention that, in order to work, the NCL attribute name must contain the same string used to identify the object inside the MPEG-4 scene, in this case, scene sensors.

Lines 19 to 23 specify a multipoint relationship corresponding to that of the same lines in Figure 2. However, the *xconnector* attribute of this link element points to a different relation type, which specifies that: when the node attribute performing the role "onTrue" has its value changed to "true", the presentation of the node performing the role "pause" must be suspended and the presentation of the node bound to the role "show" must be initiated.

In order to notify the NCL formatter of events generated internally by MPEG-4 components, modifications in the existent MPEG-4 display tools are required. As the IBM´s M4Play display tool does not provide access to nodes and sensors of an MPEG-4 scene and due to the fact that its license of use does not allow changes in its code, the open-source OSMO4 (or OSMOSE) player [13] was chosen to be altered[4].

Table 1 summarizes the main implementation files of OSMO4 that have to be modified in order to have sensors and routes mapped to NCL events.

Table 1. OSMO4 files related to MPEG-4 scene events

Files	Description
m4_esm_dev.h	Define the region structure for presentation of MPEG-4 documents.
m4_scenegraph _dev.h	Define the audiovisual structure of MPEG-4 scenes, including routes.
m4_nodes.h	Define the node structure in the MPEG-4 scenes.
m4Term.c	Procedures for access using the audiovisual display device.
route.c	Procedures for activating routes.
render2d.c	Procedures for controlling the presentation region of MPEG-4 documents.
sensor_stacks.c	Procedures for controlling sensors.

3. Templates for MPEG-4 Authoring

The previous section presented how MPEG-4 scenes can be embedded in NCL objects to improve the temporal and spatial relationships defined by the standard. However, MPEG-4 authoring can also be improved without needing to have its data structure as part of an NCL document. Although not taking into account all NCL facilities, the improvements proposed in this section do not need changes in MPEG-4 players or the use of NCL formatters and player adapters. Only minor changes to XMT-O are needed to give this language new composition semantics through NCL composite-node template use.

NCL brought the concept of architectural styles to the hypermedia domain, introducing the concept of hypermedia composite-node templates. Using a composite-node template, generic structures for hypermedia documents can be defined, allowing reuse of these structures by several distinct documents. A

4 OSMO4 has a C++ implementation that is smaller, more understandable and more flexible [13] than the two-dimensional IM1 player [9] of MPEG-4 standard.

composite-node template specifies types of components, types of relations, components and relationships that a composition has or may have, without identifying who all components and relationships are. This definition is a responsibility of hypermedia compositions that inherits (uses) the template.

NCL XTemplate is an XML-based language for the definition of hypermedia composite-node templates. XTemplate is composed by modules, allowing their combination for defining language profiles. This paper proposes a specific profile for XMT-O.

As aforementioned the definition of a hypermedia composite-node template is similar to the definition of an ADL architectural style, adding constraints to the vocabulary of types and allowing the definition of specific instances of components and connectors, which is very useful when authoring hypermedia documents. Hypermedia composite-node templates are divided in two distinct parts, and only the first one is required:

1. a vocabulary, which defines types of components and types of relations;
2. a set of constraints on the vocabulary elements, on the inclusion of instances of components, on the inclusion of instances of relations (representing relationships among components), and on the inclusion of a component inside a composite node component.

Through XTemplates, compositions with different semantics can be created, and authors do not need to be limited to the set of compositions predefined in the host language. Conceptually, templates for composite-nodes can specify any type of relationship among the composite-node components, such as composite-node nesting and spatial-temporal relationships.

In order to illustrate how a composite-node template is created, assume a composition representing the presentation of a sequence of an arbitrary number of images in parallel with a text explaining each one of them; presentation that has one logo image and an audio as background during all the sequence exhibition. Figure 5 presents an XMT-O template for this example.

A template is composed by a head and a body element. The head defines the template vocabulary, constraints, and resources that can be defined without the use of XSLT [14]. The body specifies resources and relationships among template components using XSLT. The definition of relationships in the body element gives behavior semantics to a composite-node template.

In the vocabulary, template-component types are specified. Components can be media nodes (lines 4 and 5) or composite nodes, containing other components, recursively (lines 6 to 11).

Rules for the elements specified are also defined in the vocabulary. In the example of Figure 5, constraints are defined concerning component element cardinalities, specifying the minimal and maximal number of instances each type can have.

Inclusion relations are also defined in the header (lines 6 to 11): *seq* type contains *par* types, which contains one *image* and one *text* types.

Finally, the head may contain component instances, defined by the element *resources* (line 14).

```
 1. <xtemplate … id="imagewithsubtitles" >
 2.   <head>
 3.     <vocabulary>
 4.        <component  type="audio" ctype="audio"
                minOccurs = 1 maxOccurs =1 />
 5.        <component  type=" backdrop " ctype="img" minOccurs = 1
                maxOccurs =1 />
 6.        <component  type="seq" ctype="seq">
 7.           <component  type="par" ctype="par">
 8.              <component  type="image" ctype="img" minOccurs = 1
                 maxOccurs = 1 />
 9.              <component  type="text"  minOccurs = 1 maxOccurs =
                 1/>
10.           </component >
11.        </component >
12.     </vocabulary>
13.     <resources>
14.     <resource type="backdrop"  label="logo" src="BackDrop.jpg" />
15.     </resources>
16.   </head>
17.   <body>
18.      <xsl:stylesheet>
19.        <xsl:template match="/*/*/*">
20.          <xsl:if test="not(./@type='image') and not(./@type='text')">
21.            <xsl:copy-of select="." />
22.          </xsl:if>
23.        </xsl:template>

24.        <xsl:template name="imageElement">
25.          <xsl:param name="i">
26.          </xsl:param>
27.            <xsl:for-each select="/*/*/child::*[@type='image'][$i]" >
28.                <xsl:copy>
29.                     <xsl:for-each select="text()|@*">
30.                         <xsl:copy/>
31.                     </xsl:for-each>
32.                     <xsl:copy-of select="*"/>
33.                     <xsl:apply-templates/>
34.              </xsl:copy>
35.            </xsl:for-each>
36.        </xsl:template>

37.        <xsl:template name="textElement">
38.            <xsl:param name="i">
39.            </xsl:param>
40.            <xsl:for-each select="/*/*/child::*[@type='text'][$i]" >
41.                <xsl:copy>
42.                     <xsl:for-each select="text()|@*">
43.                         <xsl:copy/>
44.                     </xsl:for-each>
45.                     <xsl:copy-of select="*"/>
46.                     <xsl:apply-templates/>
47.              </xsl:copy>
48.            </xsl:for-each>
49.        </xsl:template>

50.        <xsl:resource type="seq">
51.           <xsl:for-each select="child::*[@type='image']" >
```

```
52.             <xsl:resource type="par">
53.                 <xsl:variable name="i" select="position()"/>
54.                 <xsl:attribute name="id">par<xsl:value-of
                      select="$i"/>
55.                 </xsl:attribute>
56.                    <xsl:call-template name="imageElement">
57.                         <xsl:with-param name="i" select="$i">
58.                         </xsl:with-param>
59.                   </xsl:call-template>
60.                   <xsl:call-template name="textElement">
61.                        <xsl:with-param name="i" select="$i">
62.                        </xsl:with-param>
63.                   </xsl:call-template>
64.             </xsl:resource>
65.         </xsl:for-each>
66.      </xsl:resource>

67.    </xsl:stylesheet>
68.  </body>
69. </xtemplate>
```

Figure 5. Template for the XT-XMT-O

As aforementioned, relationships defined in the body element can be specified using XSLT [14]. In order to be able to use XSLT facilities, NCL XTemplate extends the XSLT standard. Elements of this extension are defined in a namespace refered to *xsl* in the figure.

XSLT transforms, used for specifying relationships, are defined in the *xsl:stylesheet* element. The transformation must be applied to the composite node content that inherits the *template* specification.

Inclusion operations in components following the composite-node types defined in the head, are implemented adding the *xsl:stylesheet* element as a body child. This element will define a transform where new resources must be declared using the *xsl:resource* element. In Figure 5 (lines 50 to 66), the *seq* type component is instanced by the *resource* element, whose content are instances of the *par* type component. Each instanced *par* type component receives a different value for its *id* attribute, representing the order of its creation. Inside each *par* type component, media objects, related to image and text type components, are inserted.

XMT-O templates, such as the one shown in Figure 5, can be inherited by XT-XMT-O compositions, by means of the *xtemplate* attribute. The composition will then have the semantics specified in the relationships defined by the inherited template. As a consequence, the original *par* and *seq* semantics are lost[5]. In order to use a template, the author does not need to know XSLT constructions.

Figure 6 shows an XMT-O composition (*par*), which inherits the semantic structure of the template shown in Figure 5. The *type* attributes in the XT-XMT-O document elements make references to the component types defined by the template.

```
<XMT-O …>
<head> … </head>
<body>
<par id="cityShow" xtemplate="imagewithsubtitles.xml">
<audio type="audio" id="samba" src="aquarela.mp3"/>
<img type="image" id="sld1" src="riojaneiro.jpg" dur="10"> …</img>
<img type="image" id="sld2" src="saopaulo.jpg" dur="10"> …</img>
<img type="image" id="sld3" src="belohzte.jpg" dur="10"> … </img>
<string type="text" id="txt1" textLines="Rio de Janeiro"> …  </string>
<string type="text" id="txt2" textLines="Sao Paulo">  …     </string>
<string type="text" id="txt3" textLines="Belo Horizonte">... </string>
</par>
</body>
</XMT-O>
```

Figure 6. XT-XMT-O profile document

In order to present XT-XMT-O documents, templates must be handled by a template processor, implemented extending an XSLT processor. The template processor output is a new pure XMT-O document, incorporating the directives of the referenced templates. Figure 7 shows XMT-O document obtained after processing the document in Figure 6.

```
<XMT-O …>
<head> … </head>
<body>
<par id="cityShow">
   <audio  id="samba" src="aquarela.mp3"/>
   <img id="logo" src="img/BackDrop.jpg"/>
   <seq>
      <par id="par1">
         <img id="sld1" src="riojaneiro.jpg" dur="10"> … </img>
         <string id="txt1" textLines="Rio de Janeiro"> … </string>
      </par>
      <par id="par2">
         <img id="sld2" src="saopaulo.jpg" dur="10"> …  </img>
         <string id="txt2" textLines="Sao Paulo">  …     </string>
      </par>
      <par id="par3">
         <img id="sld3" src="belohzte.jpg" dur="10">… </img>
         <string id="txt3" textLines="Belo Horizonte">... </string>
      </par>
   </seq>
</par>
</body>
</XMT-O>
```

Figure 7. XMT-O document outputted by the template processor

4. Related Work and Final Remarks

The XMT-O link (element *a*), specified in Linking Module [4], can be used to define relationships among different MPEG-4 documents. XLink [15] extensions are defined to extend the semantics of XMT-O link element. Using the XLink *actuate* attribute it is possible, for example, to define relations with temporal synchronization semantics (*onLoad* value).

[5] Note that these semantics are already lost when using XMT-O events, by means of begin and end element attributes defined by the language.

However, XLink extension does not allow to define relationships among objects within an MPEG-4 document and other media objects outside this document. This characteristic forbids the simultaneous display of all related objects, as proposed in this paper. Besides, using NCL it is possible to define many other relationships [7], as exemplified in Figures 2 and 3 (pause, resume, assign values to attributes, etc.).

In the specific case of relationships among MPEG-4 scenes, NCL formatter allows the simultaneous display of related scenes, extending the original presentation facilities. When applied to an interactive digital TV environment, MPEG-4 scenes can correspond to the encoding of audiovisual content existing in different channels of a main program. In this environment, relationships between scenes in different channels may be common during a non-linear interactive TV program exhibition.

As discussed in Section 2, an MPEG-4 player was added to the set of tools of the NCL formatter. In this player, the internal components of an MPEG-4 scene are ignored by the formatter. However, a second player was also extended, externalizing BIFS events internal to components, specified using sensors and routes. As a future work we plan to capture other MPEG-4 internal events, such as transition events established by temporal relationships specified using the FlexTime model.

Templates for multimedia presentations are proposed in [16] using a relational base as repository. The templates are created in a graphical environment, but the model for defining them is more restricted than XMT-O templates using XSLT.

Templates specific for MPEG-4 are presented in [11] using BIFS constructs. However, these kinds of templates focus on the reuse of presentation design, mainly the audiovisual components formed from various synthetic objects, and not on the reuse of complex relationship specifications, as proposed in this paper.

Considering the authoring of MPEG-4 documents, the paper shows how semi-complete hypermedia structures can be used to establish new presentation semantics for XMT-O compositions. These structures, defined as composite-node templates, allow the specification of inclusion relationships and relationships defined by events. As a future work, we intend to add other NCL facilities to XMT-O, like the XConnector [7] module. XConnector shall give more expressiveness for defining XMT-O relationships. We believe that XConnector can also be used in the definition of more expressive XMT-O templates.

5. References

[1] L. Chiariglione, “Short MPEG-1 Description”, ISO/IEC JTC1/SC29/WG11 – NMPEG96, 1996.

[2] L. Chiariglione, “Short MPEG-2 Description”, ISO/IEC JTC1/SC29/WG11 – NMPEG00, 2000.

[3] R. Koenen, “MPEG-4 Overview”, ISO/IEC JTC1/SC29/WG11 - N4668, 2002.

[4] ISO/IEC International Organization for Standardization, “14496-1:2001, Coding of Audio-Visual Objects – Part 1: Systems**”,** 2º Edition, 2001.

[5] C. Herpel, “Elementary Stream Management in MPEG-4”, IEEE Transactions on Circuits and System for Video Technology, v. 9, n. 2, p. 315-324, 1999.

[6] W3C World-Wide Web Consortium, “Synchronized Multimedia Integration Language – SMIL 2.0 Specification Second Edition”, W3C Recommendation, 2005.

[7] H.V.O. Silva, D.C Muchaluat-Saade, R.F. Rodrigues, L.F.G. Soares, “NCL 2.0: Integrating New Concepts to XML Modular Languages”, DocEng’04, p. 188-197, 2004.

.

[8] IBM AlphaWorks Emerging Technologies, Toolkit for MPEG-4 - M4Play, 2003.

[9] ISO/IEC International Organization for Standardization, “14496-5:2000, Coding of Audio-Visual Objects – Part 5: Reference Software**”,** 2º Edition, 2000.

[10] R. F. Rodrigues, L. M. Rodrigues, L.F.G. Soares, “A Framework for Event-Driven Hypermedia Presentation Systems”, 7º Multimedia Modeling Conference, p. 169-185, 2001

[11] F. Bouilhaguet, J.C. Dufourd, S. Bouilhaguet, “MPEG-Pro, an Authoring System for MPEG-4 with Temporal Constraints and Template Guided Editing”, ICME, 2000.

[12] J.F. Allen, “Maintaining Knowledge about Temporal Intervals”, Communication of the ACM, 26, p. 832-843, 1983.

[13] GPAC Project on Advanced Content, OSMOSE Player for MPEG-4 - OSMO4, 2002.

[14] W3C World-Wide Web Consortium, “XSL Transformations XSLT 2.0”, W3C Recommendation, 2005.

[15] W3C World-Wide Web Consortium, “XML Linking Language XLink 1.0”, W3C Recommendation, 2005.

[16] A. Celentano, O. Gaggi, “Template-Based Generation of Multimedia Presentations”, Journal of Software Engineering and Knowledge Engineering, v. 13, n. 4, p. 419-445, 2003.