Live Editing of Hypermedia Documents

Romualdo Monteiro de Resende Costa
Márcio Ferreira Moreno
Rogério Ferreira Rodrigues
Luiz Fernando Gomes Soares

Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225
Rio de Janeiro – 22453-900 – Brazil

{romualdo,mfmoreno,rogerio,lfgs}@inf.puc-rio.br

Live Editing of Hypermedia Documents

Romualdo Monteiro de Resende Costa
Márcio Ferreira Moreno
Rogério Ferreira Rodrigues
Luiz Fernando Gomes Soares

Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225
Rio de Janeiro – 22453-900 – Brazil

{romualdo,mfmoreno,rogerio,lfgs}@inf.puc-rio.br

ABSTRACT

In some hypermedia system applications, like interactive digital TV applications, authoring and presentation of documents may have to be done concomitantly. This is the case of live programs, where not only some contents are not known a priori, but also some temporal and spatial relationships, among program media objects, may have to be established after the unknown content definition. This paper proposes a method for hypermedia document live editing, preserving not only the presentation semantics but also the logical structure semantics defined by an author. To validate this proposal, an implementation has been done for the Brazilian Digital TV System, which is also presented.

Categories and Subject Descriptors

I.7.2 [Document Preparation]: Markup languages, Languages and systems, Hypertext/hypermedia.

General Terms

Standardization, Languages.

Keywords

NCL, Interactive Digital TV, SBTVD, declarative middleware, Ginga.

1. INTRODUCTION

Multimedia/hypermedia document authoring and presentation usually are independent phases. However, sometimes these phases have to be handled concomitantly. This is the case, for example, of non-linear TV programs produced from live events. In the context of this paper, non-linear programs are hypermedia documents conceived for interactive TV exhibition. They are usually composed by several media objects, besides the main audio and video, related in space and time. In live non-linear programs not only some contents (usually the main audio and video) are unknown a priori, but also some temporal and spatial relationships among their media objects may not be established yet. As an example, consider a soccer game live broadcast. An unexpected game pause for a player medical assistance may be used to add an advertisement, temporally related with the medical product in use at that exact moment.

Many authoring environments contain presentation functions to help in editing tasks. These functions are usually receptor (presentation environment) simulators allowing authors to visualize document presentation during the authoring phase. With these facilities, when documents are modified, the presentation generally needs to be restarted to allow the result visualization. Different from this scenario, live non-linear programs require concomitant authoring and presentation. Moreover, if the presentation environment (from now on called client) is different from the authoring one (from now on called provider), as is the case in TV broadcasting, the client usually will need authoring facilities besides its normal presentation services.

Hypermedia live editing must preserve at the client the same presentation semantics specified by the author in the provider. However, in scenarios like TV broadcasting, the client may be a retransmitter station where a recorded live program[1] should preserve not only the original presentation semantics but also the author logical intentions. Taking again the previous soccer game example into account, it should be clear the author intention of relating the medical advertisement with the product in use by the doctor, and not only with the time moment of its exhibition. This would give a hint to the retransmitter that the advertisement should not be substituted by another one that has no semantic relation with the content of the live-recorded video.

Usually, clients and providers have different goals and therefore their hypermedia documents are supported by distinct syntax notations. The authoring syntax notation normally has high-level constructs to aid document creation. It tries to capture not only the presentation flow, but also the logical organization of a hypermedia document, and it usually has constructs to favor the representation of relationships that exist among media objects in the author mind model (logical semantics). On the other hand, the presentation syntax notation normally has constructs that favor the

DocEng'06, October 10–13, 2006, Amsterdam, The Netherlands.

[1] One must distinguish between live programs and recorded live programs. Many events are advertised as being live, although they are often "live-recorded" (sometimes this is referred to as "live-to-tape"). This is particularly true, for example, in the case of war scenes repeatedly presented in news bulletin.

presentation engine to preserve the presentation flow specified by the author (presentation semantics).

As a result of their different goals, authoring and presentation syntax notations can be supported by different synchronization paradigms. While authoring syntax notations usually use high-level constructs based on the causality/restriction paradigm (event-based paradigm), some presentation syntax notations have constructs based on the timeline paradigm.

If the authoring and presentation syntax notations have a one-to-one correspondence, both logical and presentation semantics are preserved. However, this is not usually the case, and some loss occurs when syntax notations are converted. In general, when converting from the authoring to the presentation syntax notation, hypermedia systems preserve only the presentation semantics, thus losing some author logical intentions.

From the previous discussion, some conclusions can be derived. First, since live hypermedia editing requires concomitant authoring and presentation, the authoring syntax notation is generated on-the-fly and must be converted to the presentation syntax notation also on-the-fly. Thus, the authoring tool must provide means to define an initial syntax specification and means to change this notation as time goes by. This usually requires a syntax notation for document specifications and a syntax notation for commands that change these specifications.

Second, when a client acts also as a provider (as is the case of a broadcast retransmitter) and no semantics loss is allowed, two alternative solutions exist:

1) The client presentation syntax notation has a one-to-one correspondence with the original provider syntax notation; 2) The client works with two syntax notations: the first one identical to the original provider syntax notation (or having a one-to-one correspondence), and the other one (derived through conversion from the first one) for presentation. In this case, commands that modify the original syntax notation are also used to modify the client authoring syntax notation, whose modifications are converted on-the-fly on modifications of its presentation syntax notation.

Since syntax notation for authoring and presentation have different purposes and, as consequence, usually do not have one-to-one correspondence, the second solution is the choice proposed by this paper. Moreover, for not losing any semantics from the provider to the client, there will not be any different intermediary (transfer) syntax notation, thus avoiding any syntax translation. In other words, the transfer syntax notation, from the provider to the client, will be identical to the authoring syntax notation.

To validate the ideas of this paper, an implementation has been done for the Brazilian Digital TV System (SBTVD). The implementation was based on the declarative middleware named Ginga, which uses the Nested Context Language (NCL) [14] as its declarative language. The concepts can, however, be extended to other presentation engines.

The remainder of this paper is organized as follows. Section 2 contains a brief discussion of related work. Section 3 presents the live editing solution proposed for NCL. Section 4 discusses the details about the implementation carried out for the SBTVD. Finally, Section 5 contains the final remarks.

2. RELATED WORK

Most non-linear interactive digital TV systems allow live editing. For declarative applications, the main free-to-air broadcast systems use an authoring syntax notation based on XHTML-like languages [16], and a syntax notation for editing commands based on DSM-CC [9] like events. Besides XHTML modules and extensions, DVB-HTML [5], ACAP-X [2] and BML [8] (the declarative languages of the European, American, and Japanese Digital TV systems, respectively) incorporate other W3C (World Wide Web Consortium) related standards such Cascading Style Sheets (CSS) [15] and Document Object Model (DOM) [17]

In all these three languages, relationships among media objects are specified using XHTML entities and embedded ECMAScript [4] or Java[2] code. XHTML entities support simple user interactive relationships. Other spatial and temporal relationships must be specified through non-declarative (script or Java) code. This is one of the main weaknesses of these XHTML-based languages, since high-level constructions like those found in NCL, SMIL [18] and XMT-O [12], must be coded step by step in lower level abstractions of ECMAScript or Java. For a large document, this task can be difficult, tedious and error-prone. An alternative is to break the document, which represents the whole non-linear TV program, into several small documents triggered along the time by previously constructed editing commands, as if we were doing a live editing. In this case, the presentation semantics is preserved but the logical semantics can be lost, as explained in the next paragraph.

In the three systems, after receiving a declarative document, the presentation engine builds a DOM forest, whose nodes represent the XHTML-based tags. DOM allows dynamic content access and also content, style and structure updates. Thus, live editing can be done through external commands related with document editions. The commands may modify the DOM structure, preserving the presentation and also the author logical intentions, since there is a one-to-one correspondence between nodes in a DOM structure and elements in a document specification. Live editing can also be done through synchronization events that trigger a Java application or a XHTML-like document exhibition in a given instant of time, relative to the main video of a non-linear program. However, in this case, the preserved relationship is between the main video time instant and the new document exhibition. If the author's intention is to establish a relationship with any other object apart from the main video object, this semantics is lost. As an example of live editing in these TV systems, consider the aforementioned soccer game live broadcasting. When the transmission is interrupted for player medical help, we could modify the original document structure, or insert a new XHTML-based document to be launched immediately, presenting the advertisement.

MPEG-4 system standard [12] proposes another scenario where authoring and presentation may be simultaneously done. During MPEG-4 authoring, documents can be defined using the high-level abstractions of the XMT-O language [12]. In this language, relationships can be defined using compositions with temporal semantics or using events defined by media objects (start presentation, end presentation, selection etc.).

[2] http://java.sun.com.

The presentation syntax notation of MPEG-4 (that is usually its transfer syntax notation, as well) is different from XMT-O and is called BIFS (Binary Format for Scenes) [12]. BIFS can be synchronously multiplexed with document-object contents into a stream [7] that can be presented at the same time it arrives.

BIFS presentations are organized in scenes, each one structured as an object hierarchy. Modifications done in authoring subsystem could be actualized during document presentation through scene updates. MPEG-4 scenes could be granularly updated by inserting or deleting objects, or by changing object properties, such as their spatial positions. It would also be possible to insert or delete animation events and interactive events related to objects in a scene. Eventually, a scene could also be entirely replaced by another one. The only necessary support would be a tool that would allow specifying such modifications in exhibition time. Nevertheless, in general, MPEG-4 tools only allow modifications predicted and specified before the beginning of a document presentation [11].

The main problem, however, is that XMT-O and BIFS are often based on distinct synchronism paradigms[3]. The conversion from XMT-O to BIFS syntax notation preserves the presentation semantics but the relationships defined by authors using XMT-O are usually lost. Although the presentation engine can infer from BIFS the time positioning for each object, it is difficult or even impossible to infer the original logical structure defined by the author.

SMIL [18] is a W3C standard that enables authoring multimedia documents in the Web. Although SMIL documents allow live content media objects, SMIL relationships must be previously defined. The GRiNS SMIL player [3], for example, needs to receive complete SMIL document specifications before presentation starts. Modifications performed after presentation starts, which modify presentation or logical document semantics, are not reflected in the client.

Like XMT-O, SMIL relationships are specified using compositions with temporal semantics or using events defined by media objects[4]. SMIL compositions can be nested and define the presentation structure of a document. Although the techniques proposed by this paper for NCL can be applied to SMIL documents, one must be aware that at a certain presentation time, a composition may define relationships that have already occurred and relationships that are still going to occur, since compositions group into one entity a set of individual relationships. Therefore, some editing operations (in this case the removal of the composition) should be judiciously done. Hence, preserving SMIL semantics in a client, after a live editing, can be very hard. Since in NCL relationships are represented by links in isolation, the live editing is easier to be accomplished, as shown in the next sections.

[3] Besides timeline, BIFS optionally supports some temporal constructs using FlexTime model [12]. FlexTime allows defining synchronization relationships using a limited set of Allen relations [1].

[4] Actually, SMIL is older than XMT-O, which inherits several SMIL modules, including those related with temporal synchronization.

3. Live Editing Solution for NCL Language

Before presenting the proposed solution in Section 3.2, Section 3.1 gives a brief introduction to the NCL language and NCL formatter concepts.

3.1 Basic Concepts

As usual in XML based language, NCL follows a modular approach. Its Structure module defines the basic document structure. It defines the root element, called *ncl*, besides the *head* and the *body* elements, following the terminology adopted by other W3C standards.

The Media module defines basic media object (*media* node) types. The Context module defines *context* node entities. An NCL context node can contain a set of nodes and a set of links. Contexts do not have embedded temporal semantics like *seq* and *par* elements of SMIL and XMT-O. Instead, they can get these semantics and more sophisticated other ones, when used together with the Template module, as will be explained.

The Layout module specifies elements and attributes that define how objects will be presented inside *regions* of output devices. The Descriptor module allows decoupling a node from its presentation specification, allowing an object to have multiple different presentations. A *descriptor* element has temporal attributes defined by the Timing module, an attribute to identify the presentation tool (*media player*) to be used, and still another attribute referring to a region defined by elements of the Layout module. NCL Language Profile allows a descriptor to be associated with any media or context node, through *media* and *context* element attributes, through link endpoints, or through parent-context node attributes.

The TestRule module allows defining rules that, when satisfied, select alternatives for document presentation. The ContentControl module specifies the *switch* element, which groups alternative document nodes to be chosen during presentation time. The DescriptorControl module specifies the *descriptorSwitch* element, which contains a set of alternative descriptors to be associated with an object, which are also chosen during presentation time.

Events are used for spatial and temporal synchronization defined by links. Presentation and selection events (occurrences in time) can be defined over media object and context contents, using content-anchor defined by the MediaContentAnchor module and CompositeNodeInterface module, respectively. The latter also defines the *port* element, which exports a child-node interface (*port*, *attribute* or *anchor*), allowing relationships (defined by links) to go inside a composite node. AttributeAnchor module allows the definition of node *attributes* as node interfaces, used to define attribution events. SwitchInterface module allows the definition of special interfaces which will be mapped to a set of alternative interfaces of nodes contained in a switch.

The XConnector module allows defining connectors, which can be used to specify reference and spatio-temporal synchronization relations (causal and constraint relations). The Linking module is responsible for defining links (relationships) using connectors (relations). A link must refer to a connector and to events that will participate in the relationship.

NCL allows intensive reuse of its elements. Its Reuse functionality is partitioned into two modules: Import and

ReuseEntity. Import module allows importing connector bases, descriptor bases, region bases, cost function bases, test rule bases, and template bases from other NCL documents. Moreover, the module allows importing an entire NCL document to another NCL document. The ReuseEntity module allows an NCL element to be reused. An element to be reused must be referred by another element, and they are considered the same.

The XTemplate module allows the definition of composite-node templates. As aforementioned, a composite-node template may be used to give temporal or spatial semantics to a context element and are very important in easing the authoring of NCL documents. Templates of arbitrary different semantics may be created and then easily used in the document authoring. For example, a template base may include the definition of templates with the same semantics of the *par*, *seq*, and *excl* SMIL elements. Importing this base, an NCL document can be authored as if the language had these elements acting just like SMIL.

NCL formatter is in charge of receiving the NCL document and controlling its presentation, trying to guarantee that specified relationships among media objects are respected. The formatter deals with NCL documents that are collected inside a data structure known as *private base*. Documents in a private base can be started, paused, resumed, stopped and can reference each other.

Figure 1 shows part of an NCL document syntax notation for the soccer game used as example in Section 1. The live video (soccer game) is modeled by media node *V*. Since the video is live transmitted in the broadcast flow, the node source (*src*) attribute identifies the elementary stream PID in the MPEG-TS [10] flow. In other scenarios, as IPTV environments, for example, this location could be replaced by a URI with another streaming schema. A context node *C* contains the broadcast station logo and a short jingle. *V* and *C* are synchronized by a link, named *L*. The link uses the *onBeginStart* connector, defined inside a connector base imported in the document header, and defines that the video start must trigger the logo and the jingle presentation.

```
<?xml version="1.0" encoding="UTF-8"?>
<ncl id="soccerApplication" ...>
<head>
 ...
 <connectorBase>
  <importBase alias="connBase"
                documentURI="soccerApp.conn"/>
 </connectorBase>
</head>
<body id="soccerBody">
 <port id="appEntry" component="V"/>
 <media type="video" id="V" src="x-mpeg2ts:205"/>
 <context id="C">
  <port id="cEntry" component="logo"/>
  <media type="img" id="logo" src="logo.jpg"/>
  <media type= "audio" id="jingle" src="jn.mp3"/>
  <link id="cLink"
            xconnector="connBase#onBeginStart">
   <bind component="logo" role="onBegin"/>
   <bind component="audio" role="start"/>
  </link>
 </context>
 <link id="L" xconnector="connBase#onBeginStart">
  <bind component="V" role="onBegin"/>
  <bind component="C" interface="CEntry"
                                  role="start"/>
 </link>
</body>
</ncl>
```

Figure 1. Part of the "Soccer Game" NCL document.

3.2 Live Editing NCL Documents

From the discussion presented in Section 1, the procedure proposed in this paper is based on the following assumptions:

1) The client authoring syntax notation has a one-to-one correspondence with the provider authoring syntax notation.

As a consequence, the transfer syntax notation (from the provider to the client) must have a one-to-one correspondence with the client and provider authoring syntax notations.

2) A command that alters the provider authoring syntax notation must have a similar command, in the client side, which changes the client authoring syntax notation without losing the one-to-one correspondence between the syntaxes.

No assumption is made about the presentation syntax notation in the provider.

In order to satisfy the first premise, XML files based on NCL are chosen for the client and provider authoring syntax notation, as well as for the transfer syntax notation. In order to satisfy the second premise, commands are defined and hypermedia entity parameters have the same syntax notation of the authoring syntax notation in clients and providers. The commands can insert, delete or alter NCL entities.

NCL formatter (client) presentation syntax notation is based on temporal graphs [13]. This data structure favors the formatter to preserve the presentation flow specified by the author (presentation semantics). As usual, there is not a one-to-one correspondence, but a many-to-one correspondence, between the NCL syntax notation and the temporal graph notation. Therefore, the initial NCL specification must be converted in temporal chains of the temporal graph before the presentation starts. Once the presentation begins, commands sent to the client modify the data structure representing the authoring syntax, generating events that will change the presentation syntax notation.

Table 1 shows some commands defined for remotely editing NCL documents. For each NCL entity, *add* and *remove* commands are defined. If an entity already exists, the *add* command has the update (altering) semantic. Besides commands to alter the NCL specification, there are also commands to open, close and save private bases, that is, commands to deal with the data context of the NCL formatter; and commands to start, pause, resume and stop document presentations. It is worth mentioning that Table 1 shows only some of the commands, relevant for this paper, that alter the content of *head* and *body* elements of NCL documents.

Table 1. Some live editing commands for NCL language and formatter.

Commands	Description
openBase(baseId)	Open a new private base. If the private base does not exist, a new base is created.
saveBase(baseId)	Save all private base content into a persistent storage device available in the client-side.
closeBase(baseId)	Close the private base and dispose all private base content.
addDocument(baseId, document)	Add a document in a private base.
removeDocument(baseId, documentId)	Remove a document from a private base.
addNode(baseId, documentId, compositeId, node)	Add a node in a composite node of a private base document.
removeNode(baseId, documentId, compositeId, nodeId)	Remove a node from a composite node of a private base document.
addDescriptor(baseId, documentId, descriptor)	Add a descriptor in the descriptor base of a private base document.
removeDescriptor(baseId, documentId, descriptorId)	Remove a descriptor from the descriptor base of a private base document.
addInterface(baseId, documentId, nodeId, interface)	Add an interface (port/anchor/attribute/switchPort) in a node of a private base document.
removeInterface(baseId, documentId, nodeId, interfaceId)	Remove an interface from a node of a private base document.
addLink(baseId, documentId, compositeId, link)	Add a link in a composite node of a private base document.
removeLink(baseId, documentId, compositeId, linkId)	Remove a link from a composite node of a private base document.
startDocument(baseId, documentId, interfaceId)	Start playing a private base document beginning the presentation from a specific document interface.
stopDocument(baseId, documentId)	Stop the presentation of a private base document. All document events that are occurring should be stopped.
pauseDocument(baseId, documentId)	Pause the presentation of a private base document. All document events that are occurring should be paused.
resumeDocument(baseId, documentId)	Resume the presentation of a private base document. All previously paused document events should be resumed.

Add commands always have NCL entities as their arguments. These entities are defined using a syntax notation identical to that used by the NCL schemas[5]. If the specified entity already exists or not, document consistency must be maintained by the NCL formatter, in the sense that all entity attributes classified as required must be defined.

Coming back to the game example of Figure 1, when initiating the program transmission, the commands depicted in Table 2 should be sent joint with the NCL document partially presented in Figure 1.

Table 2. Initializing commands for the "Soccer Game" example.

Commands	Description
openBase (FlaAndAMG)	Creates a new private base for "Flamengo x Atletico-MG" soccer game.
addDocument (FlaAndAMG, soccerApp.ncl)	Add the NCL document presented in Figure 1 into the previously created base.
startDocument(FlaAndAMG, soccerApplication, appEntry)	Start presenting the NCL document in parallel with the game transmission (main audio and video streams).

To add an advertisement, temporally related with the medical product used for the player medical assistance, in the live editing commands described in Table 3 could be delivered.

Table 3. Live editing commands applied in the "Soccer Game" example.

Commands	Description
addInterface(FlaAndAMG, soccerApplication, V, NCLfile0.xml)	Add an anchor in the node entity whose id is V. The command also identifies the document containing V and the private base containing the document. The anchor description is done inside the NCLfile0.xml file.
addNode (FlaAndAMG, soccerApplication, soccerBody, NCLfile1.xml)	Add a node into the context whose id is soccerBody. The node to be added is specified in the NCLfile1.xml file.
addLink (FlaAndAMG, soccerApplication, soccerBody, NCLfile2.xml)	Add a link in the context whose id is soccerBody. The link is described in the NCLfile2.xml file.

The first command shown in Table 3 adds an anchor in node *V* whose content is defined in NCL file named NCLfile0.xml (Figure 2). This anchor defines a temporal instant relative to node *V* content presentation that the pause for medical care occurs (in the example, ten minutes after the game beginning).

[5] http://www.telemidia.puc-rio.br/specs/xml/NCL23/profiles/

```
<?xml version="1.0" encoding="UTF-8"?>
<area begin="600s" id="medicalAssistAnchor" />
```

Figure 2. Temporal anchor described in NCL syntax notation.

The second command inserts the node described in NCLfile1.xml (Figure 3) in the document body.

```
<?xml version="1.0" encoding="UTF-8"?>
<media type="video" id="P" src="medAd.mpg"/>
```

Figure 3. Content node described in NCL syntax notation.

Finally, the last command inserts a link described in file NCLfile2.xml (Figure 4) relating the video and the recently added node.

```
<?xml version="1.0" encoding="UTF-8"?>
<link id="I" xconnector="connBase#onBeginStart">
 <bind component="V"
       interface="medicalAssistAnchor"
       role="onBegin"/>
 <bind component="P" role="start"/>
</link>
```

Figure 4. Link described in NCL syntax notation to introduce a synchronous relationship between two video nodes.

After applying the commands specified in Figure 2 in the authoring syntax notation, a new NCL document is formed, as illustrated in Figure 5. The modifications are highlighted using bold style.

```
<?xml version="1.0" encoding="UTF-8"?>
<ncl id="soccerApplication" ...>
<head>
 ...
 <connectorBase>
  <importBase alias="connBase"
                documentURI="soccerApp.conn"/>
 </connectorBase>
</head>
<body id="soccerBody">
 <port id="appEntry" component="V"/>
 <media type="video" id="V" src="x-mpeg2ts:205">
  <area begin="600s" id="medicalAssistAnchor" />
 </media>
 <context id="C">
  <port id="cEntry" component="logo"/>
  <media type="img" id="logo" src="logo.jpg"/>
  <media type= "audio" id="jingle" src="jn.mp3"/>
  <link id="cLink"
             xconnector="connBase#onBeginStart">
   <bind component="logo" role="onBegin"/>
   <bind component="audio" role="start"/>
  </link>
 </context>
 <link id="L" xconnector="connBase#onBeginStart">
  <bind component="V" role="onBegin"/>
  <bind component="C" interface="CEntry"
                                  role="start"/>
 </link>
 <media type="video" id="P" src="medAd.mpg"/>
 <link id="I" xconnector="connBase#onBeginStart">
  <bind component="V"
        interface="medicalAssistAnchor"
        role="onBegin"/>
  <bind component="P" role="start"/>
 </link>
</body>
</ncl>
```

Figure 5. Final NCL document after live editing.

Note that commands have an order so as to maintain the document consistency. In the previous example, if the link is created before the commands to create the referenced node and anchor are launched, an invalid document may be produced in the client. In order to avoid this problem, when entities referenced by command parameters are not found, they should be created using a standard definition (sometimes only the identifier is used). Later on, these entities may be substituted, if they arrive in time, or the relationship will be ignored by the formatter.

Since different clients may experiment different delays, mainly for processing commands, the provider may opt to send a temporal anchor using the string "now". In this case, client formatters are in charge of determining the exact synchronization instant, which is given by the instant they receive the relationship (link entity) associated with the anchor. To take profit of this feature, in the soccer game, the first command parameter (Figure 2), which describes the anchor content, should be replaced by the NCL code depicted in Figure 6.

```
<?xml version="1.0" encoding="UTF-8"?>
<area begin="now" id="medicalAssistAnchor" />
```

Figure 6. Optional temporal anchor described in NCL syntax notation.

4. IMPLEMENTATION ISSUES

The live editing solution proposed in Section 3 was implemented as part of the Ginga middleware, which is the name of the middleware proposal for the Brazilian Digital TV System (SBTVD). As other digital TV middlewares [2] [5] [8], Ginga is composed by a declarative environment integrated with a procedural one. The procedural environment is Java-based and follows the GEM recommendation [6]. The declarative environment is based on the proposal described in this paper and is supported by a presentation engine based on the NCL language.

The core of Ginga presentation engine is constituted by an NCL formatter and a private base manager. The private base manager is in charge of receiving the editing commands and maintaining the active NCL documents, dealing with the authoring/transfer syntax notations at the client side. The NCL formatter is responsible for controlling the document presentations, working with the

presentation syntax notation. Besides the NCL formatter and the private base manager, the presentation engine has other modules: software components that handle content transport; media players for rendering media object content presentation; and converters to translate other hypermedia languages (like SMIL and XMT-O) into NCL syntax notation.

One instance of content transport handler was implemented to run over MPEG-2 transport streams [10]. In MPEG2-TS, the main audio and video are encoded as elementary streams and can be multiplexed with other elementary streams carrying auxiliary data. The DSM-CC [9] was also adopted in Ginga for carrying the editing commands in MPEG-2 elementary streams. Hence, a DSM-CC data stream demultiplexer was implemented as part of Ginga content transport handler.

DSM-CC stream events and DSM-CC object carousel protocol are the basis for the live editing of the non-linear TV programs in Ginga presentation engine.

DSM-CC stream events must have a structure basically composed of an id (identification), a time reference and a payload. The identification uniquely identifies each stream event. The time reference indicates the exact moment to trigger the event. A time reference equals to zero informs that the event must be triggered immediately after being received (events carrying this type of time-reference are commonly known as "do it now" events). The payload allows carrying event parameters.

The DSM-CC object carousel protocol allows the cyclical transmission of event objects and file systems. Event objects are used to map stream event names into stream event ids. Event objects are used by providers to inform the client-middleware about DSM-CC stream events that can be received. Event names allow providers specifying types of events, offering a higher abstraction level for middleware applications when registering and handling DSM-CC stream events. As an example, an event name "goal" can be defined for soccer applications. A soccer application can then register itself only as a "goal" listener and thus is free of knowing the id used to specify each stream event announcing a new goal occurred in the play. Besides event objects, the DSM-CC object carousel protocol is also used to transport files organized in directories. The Ginga DSM-CC demultiplexer is responsible for mounting the file system organization at the client side.

At the provider side, the contents of live media objects are codified as main audio and video MPEG-2 elementary streams, multiplexed in an MPEG-2 transport stream, and broadcasted to the system clients.

The NCL document files, the XML command parameters (Figures 2, 3 and 4) and the content of NCL document media objects that have minor sizes (e.g. the logo and the jingle in Figure 1) are organized in a file system structure. The editing commands are codified as DSM-CC stream events. A DSM-CC carousel generator is used to join the file system and the stream events into another elementary stream. After individually elementary streams have been created, all streams are multiplexed in an MPEG-2 transport stream, which is broadcasted to clients.

When a command needs to be sent, for example, the *addDocument* command of Table 1, a DSM-CC carousel generator is used in the provider. A DSM-CC event object is created, mapping the string "*gingaEditingCommand*" into a selected stream event id (for instance, 133), and put as the first object in the DSM-CC carousel. Afterwards, the XML command parameter, in this example the original NCL document to be added (Figure 1), is put in a file system, together with minor size media object contents, as the logo and the jingle content files. Then, the file system is encoded into the object carousel and the resultant elementary stream output by the DSM-CC generator is multiplexed in the MPEG-2 transport stream. After sending the data carousel, a DSM-CC stream event with the previous id is created and sent as another MPEG-2 TS elementary stream. The stream event is assigned a time reference equals to zero. The stream event payload receives the complete command string, in this example "*addDocument(FlaAndAMG, soccerApp.ncl)*".

At the client side, the Ginga content transport handler is responsible for receiving the files transported in the DSM-CC object carousel and placing them in the set-top box file system structure. The content transport handler also processes the DSM-CC event objects and maintains the name-id mapping information. The private base manager registers itself as a "*gingaEditingCommand*" listener and is notified about this kind of stream event. The stream event payload is then used by the private base manager to interpret the command semantics.

When an editing command needs to alter the presentation, the private base manager calls the NCL formatter. At this moment, the translation between the authoring syntax notation and the presentation syntax notation occurs.

For each media type supported by Ginga (jpeg, gif, png, html, mp3 etc.), there is a specific media player module implemented. Besides content rendering, a player monitors the media object anchors. User interactions as the beginning or end of temporal anchors are notified to the NCL formatter. When these events are conditions on NCL links the corresponding action is triggered. A special player monitors the main audio and video presentation allowing NCL media objects to be related with the main TV program being presented.

Language translators allow extending Ginga to control other hypermedia document formats. Most of SMIL and XMT-O elements and attributes are possible to be translated to NCL representation [14].

In order to simplify the private base manager task, only one private base can be opened in a certain time. The Ginga current version associates a private base with a TV channel. Thus if the user changes the selected channel, the current private base is closed. As future work, we intend to offer more flexibility in private base handling.

Two set-top platforms have been used in Ginga implementation. One is the reference architecture proposed for the SBTVD, which is based on a 700 MHz CPU with 128 MB of RAM. Another one, more restrict, with a 233 MHz AMD GEODE processor and 64 MB.

5. FINAL REMARKS AND FUTURE WORK

This paper proposes a live editing solution for declarative multimedia/hypermedia documents. The proposal aims at maintaining the authoring logical semantics while modifying the

document presentation on the fly. As a consequence, we assume a distributed scenario where providers and clients share the same authoring syntax notation. NCL was chosen as the authoring and transfer syntax notation because of its relationship specification based on links, which favors updates during document presentation.

Another advantage of NCL is that its documents are not restricted to only one entry point. The language defines that each port element in a composition can be an entrance start point. Therefore, if the author needs going to a document specific part during document presentation, the document can be stopped, a new port can be created using live editing commands and finally the author can restart the presentation, using the start command with the new port entity created as parameter.

One of the main motivations of this work was to use structured hypermedia models to improve the capabilities of authoring interactive TV programs, mainly the live and live-recorded ones. Nevertheless, the proposed solution can be adapted to a web-based (IPTV) scenario.

The ideas described in this paper have been implemented as part of the Ginga middleware, which is the middleware proposal for the Brazilian Digital TV System. More specifically, the NCL formatter, together with a private base manager, forms the core of the Ginga presentation engine. NCL documents, as well as the live editing commands, are transmitted from providers to clients using a DSM-CC multiplexer. XML files that are sent as editing command parameters are encoded within object carousel, while the commands themselves are transmitted as DSM-CC stream events.

Although the solution proposed in this paper is focused on the NCL language, it is not limited to a specific language or conceptual model. As a future work we intend to define live editing commands for SMIL. Different from NCL, where relationships can often be individually edited through actions (remove, add or update) over links and related entities, relationships in SMIL, when defined using compositions, should be judiciously done, since compositions group into one entity a set of individual relationships. However, after version 2.0, the SMIL temporal relationships may be also defined in a more individually form, using media and composition events. Hence, to preserve the SMIL semantics in a client after a live editing it may be necessary to convert compositions specifications to events. This approach may also be generically applied to XMT-O that inherits several SMIL modules.

We are now working on the interoperability of Ginga with other interactive TV declarative middlewares. The idea is to implement NCL formatter media players for each different language currently adopted: BML, DVB-HTML and ACAP-X. Thus, documents specified in these languages can be inserted into NCL documents as new NCL media node type.

6. REFERENCES

[1] Allen, J. F. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM,* 26, 5 (Nov. 1983), 832-843.

[2] ATSC Advanced Television Systems Committee. *ATSC Data Broadcasting Standard - A/90*, (July 2000).

[3] Bulterman, D., Hardman L., Jansen, J., Mullender, K., Rutledge, L. GRiNS: A GRaphical INterface for creating and playing SMIL documents. *Proceedings of WWW7 Conference, Computer Networks and ISDN Systems*, v. 30, (April 1998), 519-529.

[4] ECMA International - European Association for Standardizing Information and Communication Systems. *ECMA – 262 – ECMAScript Language Specification*, 3º Edition, (Dec. 1999).

[5] ETSI European Telecommunications Standard Institute. *DVB Specification for Data Broadcasting – ETSI EN 301 192,* (June 2004).

[6] ETSI European Telecommunications Standard Institute. Globally Executable MHP version 1.0.2 (GEM 1.0.2) – *ETSI TS102 819,* (Oct. 2005).

[7] Herpel, C. Elementary Stream Management in MPEG-4. *IEEE Transactions on Circuits and System for Video Technology*, v. 9, n. 2, (March 1999), 315-324.

[8] ISDB Integrated Service Digital Broadcasting. *Data Coding and Transmission Specification for Digital Broadcasting – ARIB STB B.24*, (2002).

[9] ISO/IEC International Organization for Standardization. *13818-6:1998, Generic Coding of Moving Pictures and Associated Audio Information – Part 6: Extensions for DSM-CC*, (1998).

[10] ISO/IEC International Organization for Standardization. *13818-1:2000, Generic Coding of Moving Pictures and Associated Audio Information – Part 1: System*, (2000).

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

[12] ISO/IEC International Organization for Standardization. *14496-1:2004, Coding of Audio-Visual Objects – Part 1: Systems*, 3º Edition, (2004).

[13] Rodrigues, R.F., Soares, L.F.G. Inter and Intra Media-Object QoS Provisioning in Adaptive Formatters. *Proceedings of ACM Symposium on Document Engineering 2003*, (Nov. 2003), p. 78-87.

[14] Silva, H.V.O., Muchaluat-Saade, D.C., Rodrigues, R.F., Soares, L.F.G. NCL 2.0 Integrating New Concepts to XML Modular Languages. *Proceedings of ACM Symposium on Document Engineering 2004*, (Oct. 2004), 188-197.

[15] W3C World-Wide Web Consortium. *Cascading Style Sheet, level 2 – CSS*, (May 1998).

[16] W3C World-Wide Web Consortium. *Extensible HyperText Markup Language – XHTML 1.0*, 2º Edition, (Aug. 2002).

[17] W3C World-Wide Web Consortium. *Document Object Model – DOM Level 3 Specification*, (April 2004).

[18] W3C World-Wide Web Consortium. *Synchronized Multimedia Integration Language – SMIL 2.1 Specification,* (Dec. 2005).