Embedding 3D Objects into NCL Multimedia Presentations

Roberto Gerson de Albuquerque Azevedo
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, RJ, Brazil
robertogerson@telemidia.puc-rio.br

Luiz Fernando Gomes Soares
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, RJ, Brazil
lfgs@inf.puc-rio.br

© ACM, 2012. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in Web3D'12, http://dx.doi.org/10.1145/2338714.2338739.
Embedding 3D Objects into NCL Multimedia Presentations

Roberto Gerson de Albuquerque Azevedo  
Pontifical Catholic University of Rio de Janeiro  
Rio de Janeiro, RJ, Brazil  
rrobertogerson@telemidia.puc-rio.br

Luiz Fernando Gomes Soares  
Pontifical Catholic University of Rio de Janeiro  
Rio de Janeiro, RJ, Brazil  
lfgs@inf.puc-rio.br

Abstract

This paper discusses how 3D objects, in special X3D documents, can be embedded into Digital TV middleware, aiming at providing 3D interactive content for both IPTV and terrestrial DTV systems. Particularly, the paper focuses on embedding 3D objects into Ginga-NCL, the declarative environment of the Japanese-Brazilian ISDB-Tb terrestrial DTV middleware, and ITU-T Recommendation for IPTV services. Thus, we propose a well-defined interface that allows 3D objects to be declaratively embedded into NCL— the declarative language of Ginga-NCL— multimedia applications. Moreover, the proposed solution will allow for 3D objects to take advantage of the high-level abstractions of NCL to specify temporal behavior of 3D scenes, and contributes to the convergence process between different multimedia (2D and 3D) technologies.


Keywords: Declarative Languages, NCL, X3D, Digital TV.

1. Introduction

Long ago, Digital TV application designers wonder when (and how) they are able to use 3D content in his/her applications. Even so, no currently IPTV and terrestrial digital TV system in use has a standard support to interactive 3D content. Although some systems can support (auto-)stereoscopic video coding, they say nothing about interactive 3D applications.

The main explanation for this fact is usually the low-end solution used in TV receivers, which do not have sufficient power to render 3D contents. However, this is changing rapidly. More and more set-top boxes—the hardware responsible to receive and present the main broadcaster video and the interactive applications—are coming with graphic cards enabled for 3D content, thanks to the decreasing price of these graphics cards.

Moreover, the convergent use of different receiving platforms is also a trend in DTV systems. Powerful media-centers and portable receivers (like cell-phones, notepads, etc.) have begun to receive the same broadcasted/multicast signals. For instance, in the Japanese-Brazilian ISDB-Tb (Integrated Services for Digital Broadcast-Terrestrial) [ABNT 2011] system, portable devices are able to receive digital TV signals, and, in addition, work together with other fixed receivers (like a TV set), in a distributed multiple exhibition environment [Soares et al. 2009]. These portable devices usually come with 3D-enabled graphic support, making able 3D interaction even when using low-end fixed receivers.

Digital TV system can benefit through innumerable ways by supporting 3D interactive applications, if there are standard solutions. Educational games, showcase of products, virtual reality environments for collaboration activities, and virtual field trips are only few examples of applications that could be developed to improve the user immersion and the visual appealing of interactive digital TV (iDTV) applications.

In contrast with terrestrial and IPTV iDTV systems, the Web community has introduced in the last decades various proprietary and some standard solutions for delivering interactive 3D-content, although we still do not have a single widely adopted technology [Behr et al. 2010]. X3D [WEB3D 2009] is maybe the most promising one to become widely adopted in the Web [W3C 2009(b)]. This brings the question: why not embed and try X3D solutions in iDTV standards? This is part of the proposal of this paper.

However, seeing TV is very different from surfing in the Web—and, consequently, the support required by application developers. For instance, the Web—until today, and even with HTML5 [W3C 2009] new features—is mainly a text-based medium; while TV is mainly a video-based medium. For that reason, and some other well-known specific features of TV environments [Soares et al. 2010], support to structured-based spatiotemporal synchronization [Bulterman and Hardman 2005] (in which synchronization is defined outside the media object content, instead of media-based synchronization as provided by HTML), support to content and content presentation adaptation, support to multiple exhibition devices, and support to live editing are key features related with TV technologies. Hypermedia languages that support spatiotemporal-synchronization specification such as first class features and at a declarative level—like SMIL [Bulterman and Rutledge 2009], and NCL (Nested Context Language) [ABNT 2011]—have advantages over those languages that do not care about time and must be extended by imperative (usually scripting) languages to define temporal behaviors.

However, many technologies developed for the Web, including HTML, also fit well for interactive digital TV when they are integrated into structure-based hypermedia languages. This is particularly the case of the Ginga middleware that allows for HTML pages to be rendered embedded into NCL [Soares et al. 2009(b)] applications.

NCL is the declarative language of Ginga, the middleware standard of the aforementioned ISDB-Tb terrestrial DTV system, and also ITU-T Recommendation for IPTV services [ITU-T 2009]. NCL is a glue-language that relates media-object in time and space, independently of their types and without restricting any content type. The content of each media object is defined in its own language or media format. In using NCL, each specific iDTV system must define which media types it supports. Ginga-NCL is the NCL user agent, responsible for receiving NCL applications, and orchestrating and distributing their presentation on multiple exhibition devices.
Aiming at supporting 3D interactive content for iDTV (both terrestrial and IPTV), this paper discusses how to extend NCL (Nested Context Language) in order to incorporate 3D-media-objects, such as the unstructured mesh-based and scene graph-based ones. In special, we will discuss how X3D can be embedded into NCL documents, and how we can relate it temporally with the other media objects that composes an NCL multimedia presentation. The solution proposed here follows the same principle of how HTML is integrated into NCL through a well-defined interface at a declarative level.

The solution proposed in this paper will contribute to extend NCL-based DTV middleware systems to allow for 3D interactive scene presentations. Nowadays, only the ISDB-T system comprises a market of about 600 million of people. Thus, the improvements proposed by this work may have a great impact.

As a side-effect of our proposal, 3D-objects could also take advantage of the high-level temporal specification of NCL to specify temporal behavior of 3D scenes.

Moreover, by providing integration between X3D and NCL, our proposal also contributes to the convergence process between different multimedia (both 2D and 3D) technologies, which is a trend in both the Web and iDTV systems. This convergence of technologies is also a key feature for systems like IPTV and Broadband TV, which traditionally merge iDTV and Web features.

After this introductory section, the paper is organized as follows. Section 2 introduces a motivation example of applications we want to support with our proposal. Section 3 presents some main related work. Section 4 discusses how 3D-objects (both unstructured mesh and scene graph objects) can be embedded into NCL documents. Section 5 presents an implementation based on the ITU-T Ginga-NCL reference implementation [Ginga Community] to validate our approach. Finally, Section 6 presents the conclusions and some future work.

2. Motivation Example

To illustrate a practical use case that our proposal supports, Figure 1 shows an iDTV application composed of:

1) An educational video (e.g., a TV show transmitted by a TV Channel) describing the different bones of the human body.
2) While the video is being played, viewers are invited to interact with the show, through an image showed in the upper right corner of the screen.
3) Upon accepting the invitation, the educational video shrinks to the left side of the screen (see Figure 1 (b)), and an interactive 3D model of the human skeleton appears on the right side of the screen.
4) The user is then able to manipulate the 3D model, and if he/she selects a specific bone, the NCL host application is notified, which triggers the presentation of additional information about the selected bone (the 3D model could also be synchronized with the current explanation of the main video, but we do not do this here to keep our example as simple as possible).
5) The additional information together with a more descriptive 2D image are gotten from the Wikipedia.

![Figure 1](image1.png)

**Figure 1.** An educational NCL application for iDTV, with an embedded X3D model about the human skeleton. (a) The application starts with a video describing the different bones of the human body and shows an icon inviting users to interact. (b) If a user selects the icon, a 3D model of the human skeleton is presented. (c) Then, if the user selects a bone in the model, a detailed description of this bone is presented.

This application will be modeled as an NCL parent document, whose structural view, a graphic abstraction that represents media-objects and their relationships, is depicted in Figure 2.

![Figure 2](image2.png)

**Figure 2.** Structural view of our motivation example.

NCL embeds other media objects through a well-defined interface and follows a non-intrusive mechanism [Soares et al. 2009(b)]. To notify a viewer selection of a specific bone (that happens inside the 3D-model), we have to define interfaces in the media-object containing the X3D model and to map these interfaces to the X3D’s internal objects – Section 4.2 details our solution for X3D objects. In doing these mappings, the NCL application can receive internally generated X3D events and relate them to other media objects. The structural view of Figure 2 also shows the relationships between media objects (expressed using NCL <link> elements). The application is started presenting the main video. The beginning event of this video starts the presentation of the invitation icon. The selection event on this icon, resized the main video, starts the presentation of the X3D model and starts the execution of the Lua script. An internal selection event coming from an internal X3D’s object is exported through the X3D media-object interface. This event calls Lua’s event handlers that get information from Wikipedia and presents it.
3. Related Work

The proposal of this paper is closely related to the one presented in [Soares et al. 2009(b)]. That work describes abstractly how NCL can embed and relate imperative and declarative hypermedia objects. As a use case, that work describes how to integrate declarative HTML documents and imperative Lua objects into NCL applications. In this paper, we are extending and instantiating the abstract proposal of [Soares et al. 2009(b)] to 3D-objects.

There is also work on integrating 3D and time-based languages, but unlike the NCL proposal, almost all work mixes the two original languages into a new language. For XML-based languages this is commonly archived by embedding XML modules of a language into the other one. Some examples of integrating different XML Modules into one new target language are XHTML+SMIL, SVG+SMIL, etc.

It can be argued that merging different languages into one could be simpler from an author point of view; and for simple applications, and it can be. However, this is not completely true in a development process composed of professionals with different skills, with different expertise. It could be better if each professional could develop its work using the language he/she knows better, without worrying about details that are not important for his/her current task.

However, we have at least two good examples of technologies that follow this merging language approach.

**Extensible MPEG-4 Language**

MPEG is an ISO/IEC standard for communicating interactive audiovisual scenes. The standard defines a set of binary tools for coding individual audio-visual objects, text/graphics and synthetic objects. The interactive behavior of these objects and the way they are composed in space and time is coded in a binary format, known as BIFS (Binary Format for Scenes). In order to allow authors to create the scenes, MPEG-4 Part 11 defines the XMT-A, which closely mirrors the binary format in a XML-based version. Moreover, in order to allow a high-level abstraction, they also provide the XMT-O (Extensible MPEG-4 Language - Ômega) [Kim et al. 2000]. One of the main advantages of the BIFS is the fact that it is streaming oriented, and that it has scene update mechanisms, fundamentals features in DTV domain.

XMT-O is an example of integration between a 3D language and a time-based language. XMT-O inherits the scene graph specification from X3D – with some adaptations – and the temporal containers from SMIL. In addition to SMIL basic events, XMT-O also supports events related to 3D objects: 2D and 3D objects are able to generate events like click, mouseup, mousedown, mouseout and mouseover, while viewable, near, and collide are events related to 3D objects. In mixing the scene structure with the temporal behavior specified using SMIL timing module, XMT-O does not preserve rendering optimizations that could be extracted from the scene graphs [Azevedo 2010].

Unlike XMT-O, the proposal of this paper still keeps NCL language separated from the 3D language, but interoperating through a well-defined interface. The border-line between the languages will be very well-defined, allowing each different professional to do his/her job without worrying about the other language syntax and semantics. Moreover, all rendering optimizations allowed by scene graph structure are preserved. Like XMT-O, we also extend the original 2D events, to allow NCL be aware of 3D related events.

**X3D + HTML**

X3DOM integrates X3D and HTML also in a unique target language that mixes both languages and allows authors to treat HTML and X3D in a same merged DOM tree. This has some advantages, for example, as aforementioned, it can be easier for simple applications, and allows authors to use DOM events to control scenes through an imperitive code (for instance, using JavaScript).

Unlike the current X3DOM proposal, the integration of X3D and HTML historically (and in many cases until today, because of its better performance) has been using plug-ins to render X3D content, following a loosely-coupled model.

One of the major reported [Behr et al. 2010] drawbacks that contributes to the little adoption of plug-in based technologies in the Web (except of course Flash [Adobe Systems]), comes from the fact that they are not installed by default. In iDTV systems this could not be a problem since the technology standardized, every manufacturer must implement it. A second drawback pointed out by [Behr et al. 2010] comes from the plug-in model decoupled from the DOM content. This fact requires that authors have to deal with the plug-in-specific interface and its synchronization model.

NCL is a glue language. Every media object must follow a well-defined interface part of the NCL conceptual model. Thus, an NCL author does not have to worry about internal object constructions, but only in how to synchronize in time and space events generated by these object. On the other hand, authors of media object content do not have to worry about NCL structures. Therefore, our proposal unifies the advantages coming from X3DOM proposal with the plug-ins proposal. Moreover, we allow authors to control scene behaviors declaratively.

In this initial phase, unlike XMT-O and like X3DOM, this paper focus on embedding X3D content to be rendered into 2D regions, targeting the convergence between 2D and 3D multimedia technologies.

**Other works embedding 3D-content into NCL**

Other proposals also recognize the NCL glue facility of embedding 3D objects into NCL applications [Souza et al. 2010] [Tavares et al. 2010], although none of them had proposed a real integration between NCL and 3D objects defined using other specification languages. None of these related work recognized that NCL can also be used to define content anchors inside 3D objects and relate them with other media-object’s interfaces present in NCL applications. Moreover, they did not recognize that by defining anchors inside 3D objects, NCL is able to define temporal relationships inside these 3D objects as well, controlling their behavior.

4. 3D Objects inside NCL Documents

In order to systemize our approach, let’s first define two categories of 3D objects NCL is able to embed: *simple* and composite 3D-objects.

Simple 3D-objects represent discrete 3D models: sets of geometry which, for NCL application purposes, are self-contained. Primitive geometric objects, like cube, sphere, etc., and also
unstructured mesh-objects are examples of simple 3D-objects. Some examples of file formats for these objects are: Wavefront OBJ file format, PLY (Polygon File Format), etc.

On the other hand, composite 3D-objects are handled by NCL as a set of internal objects the 3D-language can recognize. In doing so, NCL applications will be able to identify individual objects (or a subset of them) inside the composite 3D objects, and relate them to others internal or external objects. Scene graphs are the most common way to model these 3D composite objects. Some examples of languages used to specify such objects are: X3D, VRML, etc.

4.1 Embedding Simple 3D Objects

To allow simple 3D-objects into NCL applications, media players able to render these object types must be integrated to the NCL engine. Like any other media renderer, 3D-object players must follow the API that NCL defines for media players [Soares et al. 2011]. As a result, NCL applications will be able to start, stop, pause, resume, or abort the presentation of the embedded simple 3D-objects, as well as to establish relationships among events coming from these 3D-objects and events coming from other NCL media objects.

If the NCL engine has the associated media player integrated, to embed a simple 3D object inside an NCL application, authors just need to create a <media> element with its src attribute referring to a 3D media file, as shown in the next example:

```xml
...<body>
  <port id="p1" component="body3ds"/>
  <media id="bodyOBJ" src="body.obj">
    <property name="width" value="50%"/>
    <property name="height" value="50%"/>
  </media>
</body>
...
```

Other specific properties could also be defined for these simple 3D-objects, e.g., camera position, scale, translation and rotation, by using <property> elements (child of <media>). The semantics of content anchors defined on these objects is identical to that defined on basic 2D-media object types (video, audio, images, etc.).

Simple 3D objects are considered atomics. If we need to create content anchors inside a 3D object referring to any of its embedded geometry, we need to treat this media object as a composite 3D-object.

4.2 Embedding Scene Graph 3D-Objects

NCL also allows for embedding composite 3D-objects as an NCL media object specified using some 3D language coding. This is also done through using NCL <media> elements. In these objects, it is possible to define interfaces referring to their internal components. There are two types of interfaces represented by <area> and <property> elements, children of <media> elements.

By defining these interfaces to internal objects inside the composite 3D object, NCL will be able to establish relationships (through <link> elements) among events coming from internal individual objects; for example, reporting property changes (an NCL attribution event) from a specific node inside a scene graph, or reporting a specific 3D object (a Cube or Sphere, for instance) selection by a user (an NCL selection event).

Content Anchors

An NCL <area> element defines a content anchor, that is, a subset of information units that compose the object’s content. The subset of information units is dependent of the media type. For example: for a video object, the information unit can be frames; for an audio object, information units can be audio samples; and so on.

In special, for composite 3D-objects based on scene graph descriptions, a simple and useful approach is to allow scene nodes (group or simple nodes) to be defined as content anchors. For composite 3D-objects, the label attribute of the <area> element can be used to identify the content inside scene graph. In our proposal, the label value must be directly mapped to a node identifier in the scene graph. Of course, the scene graph player coupled to NCL player must be able to perform that mapping and to notify all events coming from this interface to the NCL engine. Using this approach, all nodes in scene graphs that have unique identifiers can be used as content anchors.

Figure 3 shows how NCL can define content anchors on embedded X3D scene graphs using an <area> element that refers to a graph node using its DEF attribute value, which uniquely identifies it.

![Figure 3](image.png)

Figure 3. Example of NCL content anchor defined on scene graph based object.

Since any node in the scene graph that has a unique identifier can be a content anchor, X3D sensor nodes are just particular examples. In this case, when a sensor node is activated, the X3D player must notify its starts presentation transition (eventType=presentation, transition=start), and when it is deactivated, the player must notify its stop presentation (eventType=presentation, transition=stop) to the NCL engine. However, as discussed in Section 4.3, we propose to extend NCL events so that the use of sensor nodes is not required anymore.

Another way to define content anchors in declarative composite 3D-objects is through the NCL clip attribute. The clip attribute is defined by the tuple “(chainId, beginOffset, endOffset)”. ChainId identifies a temporal chain in the embedded object (indeed its starting object). A temporal chain corresponds to a sequence of presentation events (occurrences in time), initiated from the event that corresponds to the beginning of the declarative media-object presentation [Soares et al. 2009(b)]. The beginOffset and
endOffset parameters define the begin time and end time of the content anchor in relation to the beginning time of the temporal chain.

A simple way to integrate scene graphs with NCL applications is considering the exhibition of a sub-graph in a scene graph as a temporal chain. In doing so, we are now able to define temporal anchors inside nodes of the scene graph. For example, the following code shows a temporal anchor that starts after two seconds from the beginning of the mySphere presentation and finished ten seconds after it starts:

```xml
<media id="mySceneObj" src="myScene.x3d">
  <area id="mySphereClipAnchor" clip="(mySphere,2s,10s)"/>
</media>
```

Any clip attribute parameter may be omitted. If the chainId attribute is not set, the default value refers to the root node of scene graph (i.e. the whole scene is the temporal chain). If beginOffset or endOffset is omitted, they assume their default values: 0s and chain end time, respectively. The following code exemplifies an anchor starting two seconds after the scene graphs presentation begins and finishing eight seconds later.

```xml
<media id="mySceneObj" src="myScene.x3d">
  <area id="myTemporalAnchor" clip="(2s,10s)"/>
</media>
```

**Property Anchors**

A property or a set of properties of NCL media objects can also be used as interface when externalized by the NCL <property> element. Thus, in embedding media objects with imperative or declarative code, NCL allows for controlling their internal properties; and this is true also for scene graph embedded objects.

Commonly, scene graph properties do not have any scope concept. When authors define a node in a scene graph, its properties may be modified by any other internal node or external API (for instance SAI [Web3D 2009(b)], or EAI [Web3D 2009(b)]), if the first node could be uniquely identified. Following the same reasoning, when authors set a unique identifier to an object internal to a composite 3D-object they also permit NCL to access this node and its attributes, if <property> elements are defined referring to the desired attributes. The attribute to be controlled is referred by having the “NODEID#ATTR” value set to the name attribute of the <property> element of the embedded composite 3D object. NODEID is the unique identifier (e.g., DEF attribute in X3D) of the scene-graph node, and ATTR is its internal attribute to be controlled.

Figure 4 presents the syntax we propose on how NCL could have access to an internal attribute of an embed X3D object. Following this proposal, NCL <link> elements are also able to change specific attributes of an internal node of an X3D object.

**4.3 Extending NCL events to 3D-objects**

In NCL it is possible to define constraint and causal relations. In NCL DTV profiles, however, only causal relations are allowed [Soares et al. 2009(b)]. A causal relation is defined using <causalConnector> elements, and it defines a set of conditions on events that must be satisfied to trigger actions on other events. The relations are intended to be reused on relationships (<link> elements), and defines a set of roles that media objects can play when taking part in a relationship of that type. For instance, the following link:

```xml
<link id="myLink" src="onBeginStop">
  <bindrole="onBegin" component="video1"/>
  <bindrole="stop" component="image1"/>
  <bindParam name="delay" value="5s"/>
</link>
```

refers to a previously defined causal connector “onBeginStop” to create a relationship. In this relationship we are defining that “when video1 begins, and after 5s of delay, the image1 will be stopped”. As we can see in the example, the connector can also receive a set of parameters (the required parameters are defined in the <causalConnector> through <connectorParam> child elements).

Both content anchors and properties define events in NCL [Soares et al. 2009(b)] that can be used in <causalConnector>, and, therefore, in <links>-. An event denotes any occurrence in time with a finite or infinitesimal duration [Pérez-Luque and Little 1996]. Each event defines a state machine (see Figure 5) that must be maintained by the corresponding NCL player. Transitions between states (sleeping, paused, and occurring) are conditions of <link> elements, and actions on a state machine coming from <link> elements cause transitions between states.

![Image 325x617 to 553x738](Image 325x617 to 553x738)

**Figure 5.** Event state machine.

In the current version of NCL, only presentation, selection and attribution event types are supported. However, NCL allows extensions by adding new event types; since they follow the same
event state machine semantics defined for the original presentation, selection and attribution events.

3D objects usually define specific events for 3D environments, as for example, collision events. By extending NCL events to include collision, proximity, and visibility events we are allowing events generated by embed composite 3D-objects to be related through NCL links. Thus, in our proposal NCL language is extended in order to offer at least these three new types of events besides the original ones.

NCL defines reserved words to specify event types and transitions on their event state machines. The goal is only to simplify the specification. Following the same principle, Table 1 summarizes the new event types we propose to allow NCL to use 3D specific-events in its <link>, and the corresponding reserved words to be used in NCL <causalConnector> elements.

<table>
<thead>
<tr>
<th>role</th>
<th>eventType</th>
<th>transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>onBeginCollision</td>
<td>collision</td>
<td>starts</td>
</tr>
<tr>
<td>onEndCollision</td>
<td>collision</td>
<td>stops</td>
</tr>
<tr>
<td>onBeginCloseness</td>
<td>proximity</td>
<td>starts</td>
</tr>
<tr>
<td>onEndCloseness</td>
<td>proximity</td>
<td>stops</td>
</tr>
<tr>
<td>onBeginVisibility</td>
<td>visibility</td>
<td>starts</td>
</tr>
<tr>
<td>onEndVisibility</td>
<td>visibility</td>
<td>stops</td>
</tr>
</tbody>
</table>

Like X3D, and unlike XMT-O, the proposed collision event means a collision between the observer avatar and an interface (an object). When a collision is detected, the collision state machine is changed to occurring, keeping in this state until the observer avatar stops to collide with the interface, when the state machine changes back to the sleeping state.

Proximity event means that the observer’s avatar in the virtual world is at most at some distance from the related content anchor (for instance, an internal X3D object). The proximity state machine is in the sleeping state while the observer keeps at a distance greater than a specified distance. This distance can be defined, for instance, as a real number. Another way, is defining the dimensions of a cube that bounds the object mapped by the interface. Authors can use the <bindParam> elements in NCL links to specify this distance.

A natural consequence of defining content and property anchors for 3D composite objects and of extending NCL to support 3D events is to allow NCL links to control the behavior of these embedded objects. Thus, authors can use NCL links to specify the time-dependent behaviors of a scene, taking advantage of the declarative approach of NCL to design more complex 3D temporal behavior.

Furthermore, as NCL links are totally independent from the X3D scene graph, our approach allows for modeling geometric relationships independent of the temporal behavior. The clear separation between scene structure and its behavior improves the maintenance and readability, and also promotes easier adaptation and exchange. Moreover, all rendering optimizations allowed by scene graph structure are preserved.

A new and important feature that arises from this new approach is the possibility of making relationships between different 3D worlds, besides relationships between 3D and 2D objects. As NCL can embed more than one composite 3D-object and define content anchors in each one of them, it is possible to specify relationships among these different 3D scenes. This is especially useful in distributed environments, in which each 3D world can be presented in a different device, taking profit of the NCL facility of supporting exhibitions on multiple device [Soares et al. 2009].

Figure 6 exemplifies some relationships 3D objects are able to participate when embedded in NCL applications. In (a) there is an NCL application composed by a scene graph and a 2D-object (image, video, etc.), in which a relationship is defined between the composite 3D-object and the 2D-object stating: “when the scene graph starts, the 2D-object must also start”. In (b) a content anchor inside the scene graph is defined, and a collision relationship defines that “when the user avatar collides with the geometry (i.e., the content anchor), the 2D-object must stop its presentation. In (c) we have an example of an NCL application controlling the behavior of a scene graph, instead of using route graphs. In the example there are two content anchors referring to internal geometry nodes, and an NCL link specifying that: “when an internal geometry node are selected, the property of the other geometry node must change”. Finally, in (d) there is an example of NCL application relating more than one scene graph. In this specific case, there are two composite 3D media object, each one with a content anchor. There is also a relationship between this two content anchors defining: “when the observer is close to the first anchor, the presentation of the second anchor (part of the other scene graph) must start”.

Table 1. The new proposed NCL events coming from 3D media objects
Figure 6. Some examples of relationships defined on embedded X3D objects.

5. Implementation

To validate our proposal, we have integrated a simple OBJ player and an X3D player to the Ginga-NCL reference implementation (an open-source NCL player [Ginga Community]). We have developed a simple OBJ player as a testbed to unstructured mesh embedded objects, and have extended and used the FreeWRL player [Stewart] as the X3D rendering engine to test embedded scene graph-based 3D objects.

The OBJ player works as any other NCL media player following the standard Ginga-NCL player API (called IPlayer). The single difference is that some properties common to 3D objects are also managed by OBJPlayer, like camera position, scale, etc. As any other property, these properties can also be controlled by NCL links.

The integration with X3D player is implemented using the EAI (External Application Interface) provided by FreeWRL. We do not use SAI because FreeWRL does not have a SAI implementation for C/C++. In addition to allow scene graph manipulation, EAI allows callback registrations to manage internal X3D events and properties. In our implementation, we are particularly interested in presentation, selection, attribution, collision, visibility and proximity NCL events (the last three introduced by our proposal).

As with all media object player, our X3D player must execute an initialization procedure when instantiated. In particular, this procedure registers one (or more) event handler for communication with the NCL player. To each defined content and property anchor specified in NCL documents, the X3D player registers callbacks through EAI, and notifies the NCL player when changes occur.

If X3D authors want to allow the notification of the 3D-related event occurrences to the NCL player, they should define sensor nodes (for instance, ProximitySensor, CollisionSensor, etc.) inside the scene graph. As previously mentioned, NCL allows for defining anchors referring to these sensor nodes, and thus is able to synchronize internal 3D-related event occurrences to external NCL events.

However, usually NCL authors are not familiar with the concept of sensors. In NCL, events are generated without needing sensor nodes. We can easily harmonize the requirements of these different environments simply adding sensor nodes automatically, in the initialization procedure of the X3D player, and without author intervention. When NCL authors define anchors (interfaces) to geometric nodes, sensor nodes are automatically created, based on the event types defined by the NCL links that refer to these interfaces.

In the current implementation, anchors can be defined for any kind of X3D node that has a unique identifier (the DEF attribute in X3D). Note thus that, a sensor node explicitly defined by the embedded X3D document is able to be an anchor, as well as any kind of geometry or group nodes, besides the engendered sensor nodes, of course.

Figure 7 shows an example of how sensor nodes are automatically created. In the NCL document, we can see an X3D media object with an anchor (<area> element “aSphere”), referring to the “sphere” internal geometric node, defined in the embedded X3D document. There is also a link in the NCL document describing that some action (omitted in the figure) should be performed when user selects this anchor (“onSelectionaSphere interface of scene component”). When Ginga-NCL parses the NCL document, upon reaching the link, a TouchSensor is automatically created using EAI. Similar procedures are used to create CollisionSensor, VisibilitySensor, and ProximitySensor nodes when collision, visibility and proximity events need to be notified to the NCL player.
Secondary devices can be the parent device of other secondary devices, following a hierarchical model. This key feature of Ginga-NCL can be used to support distributed 3D multimedia presentation. For instance, we could have different 3D worlds, each one being presented in a different device, and being synchronized from the main NCL application, in a true virtual environment, which can move social applications one step ahead.

As a future work we intend to compare the route graph (provided by X3D), the XMT-O approach, and the NCL models concerning functional and non-functional requirements, in special efficiency and usability issues. This could verify some assumptions that we already can state in the following three paragraphs.

When XMT-O mixes content and behavior into a single hierarchy, authors are not anymore able to model a scene just considering spatial relationships. Therefore, rendering optimizations based on scene graphs are allowed only for applications in which there is no conflict between the hierarchical spatial structure of scenes and their temporal hierarchy [Azevedo 2010]. Both route graphs and NCL links have not this drawback of XMT-O, since they keep the scene structure and scene behavior separated.

NCL modules and profiles can be combined with modules of other languages, allowing for integrating NCL features into those language profiles and vice-versa. Thus, instead of having NCL as a glue language relating embedded 2D and 3D objects as proposed by this paper, we are also planning to investigate the use of NCL facilities integrating NCL modules into X3D. This approach could be used at least to allow behavior specification, using the expressiveness and high-level declarative syntax notation of NCL, instead of routing graphs. To reuse existing X3D players, documents written in that new language (X3D scene graph plus NCL links) could be compiled into the current X3D specification in which NCL links would be converted to route graph specifications. Of course, NCL features not supported by X3D route graphs, like distributed multimedia exhibition or constraint relationships, would be lost in this case.

As NCL <context> elements allows for logically structuring an application, we also intend to investigate the use of contexts similarly to the use of group nodes in scene graphs. Without losing the optimization procedures provided by scene graphs, contexts will be evaluated in their support to 3D authoring processes.

Finally, we have also started the specification of a new version of the NCL language (version 4.0) and the implementation of its new player. This new version includes the definition of several

---

**Figure 7.** Automatic creation of TouchSensor node based on NCL links.

**Figure 8.** Three samples of 3D objects rendered over the main video of a Ginga-NCL reference implementation: (a) an unstructured mesh-based OBJ media-object; (b) an X3D model embedded in a NCL application; and (c) a complete application that allows the user to navigate through a model of a condominium (named Mediterrâneo) that is for sale (bottom and right).

---

6. Conclusions and Future Work

Based on the proposal of Section 4, and in addition to our motivation example (Figure 1), we have developed some other examples of iDTV applications to validate the functional aspects of our approach. Figure 8 shows some screenshots of them, got from their running on our extended Ginga-NCL implementation. The examples we have developed until now are simple, but some represent interesting applications; for instance: a condominium advertisement (Figure 8.c) shows the apartment blocks in a 3D model that allows viewers to freely navigate or navigate through a guided tour on it.

By providing interfaces mapped to internal 3D objects of a scene graph and in extending NCL events to 3D support, our proposal also allows application authors to define all scene behavior externally, and mainly, declaratively, using NCL links. In special, we are working in stressing these possibilities, in order to identify the real necessity of high level temporal abstractions (as the ones provided by NCL) in 3D applications in general (and not only for DTV domain).

We have also already started the integration of FreeWRL player to Ginga-NCL [Daher et al. 2010] secondary devices. As aforementioned, Ginga-NCL supports distributed exhibitions on multiple secondary devices that are controlled by a parent device.
properties present in 3D world, including the exhibition of 2D objects on 3D surfaces; the definition of new 3D events, the definition of new causal and also constraint relationships between events; among other features. In the new NCL 4.0 player we are also working to embed additional new 3D players, like 3DS, Collada, etc., besides OBJ player and X3D player.

References


Ginga Community, at Software Público Brasileiro. Available at: http://www.softwarepublico.gov.br


