PUC

Nested Context Language 3.0

Part 11 – Declarative Hypermedia Objects in NCL: Nesting Objects with NCL Code in NCL Documents

Luiz Fernando Gomes Soares

Departamento de Informática

Nested Context Language 3.0

Part 11 – Declarative Hypermedia Objects in NCL: Nesting Objects with NCL code in NCL Documents

Luiz Fernando Gomes Soares

Laboratório TeleMídia DI – PUC-Rio
Rua Marquês de São Vicente, 225, Rio de Janeiro, RJ - 22451-900.

`lfgs@inf.puc-rio.br`

Abstract. *This technical report describes how declarative hypermedia objects, including objects with NCL code, may be related with other objects in an NCL application, and how declarative hypermedia object players shall behave. NCL (Nested Context Language) is an XML application language based on the NCM (Nested Context Model) conceptual model for hypermedia document specification, with temporal and spatial synchronization among its media objects.*

Keywords: *declarative hypermedia objects, digital TV; middleware; declarative environment; NCL, Lua.*

Resumo. *Este relatório técnico descreve como objetos com código declarativo, incluindo objetos com código NCL, podem se relacionar com outros objetos em uma aplicação NCL e como exibidores (players) para esses objetos devem se comportar. NCL é uma aplicação XML baseada no modelo conceitual NCM (Nested Context Model) para a especificação de documentos hipermídia com sincronismo espacial e temporal entre seus objetos.*

Palavras chave: *objetos declarativos; TV digital; middleware; linguagem declarativa; NCL, Lua.*

Nested Context Language 3.0
Part 11 – Declarative Hypermedia Objects in NCL: Nesting Objects with NCL code in NCL Documents

Table of Contents

Nested Context Language 3.0
Part 11 – Declarative Objects in NCL: Nesting Objects with NCL code in NCL Documents

Luiz Fernando Gomes Soares

Laboratório TeleMídia DI – PUC-Rio
Rua Marquês de São Vicente, 225, Rio de Janeiro, RJ - 22451-900.

`lfgs@inf.puc-rio.br`

***Abstract.** This technical report describes how declarative hypermedia objects, including objects with NCL code, may be related with other objects in an NCL application, and how declarative hypermedia object players shall behave. NCL (Nested Context Language) is an XML application language based on the NCM (Nested Context Model) conceptual model for hypermedia document specification, with temporal and spatial synchronization among its media objects.*

1. Introduction

Declarative hypermedia objects (with NCL code or coded with another declarative language) may be inserted into NCL documents. The way to add a declarative hypermedia object into an NCL document is to define a <media> element, whose content (located through the *src* attribute) is the declarative code to be executed. As an example, both EDTV and BDTV profiles of NCL 3.0 allows the <media> element of application/x-ncl-NCL type to be nested in a NCL document.

This technical report describes how to define declarative hypermedia objects and how to relate them with other objects of an NCL application. The report is organized as follows. Section 2 gives an historical evolution of the NCL versions. Section 3 presents a brief overview of the NCL 3.0 elements. Section 4 describes how declarative hypermedia objects may be defined together with their content anchors and properties. Section 5 discusses the expected behavior of declarative hypermedia object players, and how declarative hypermedia objects may be related with other objects in an NCL document. Section 6 presents the final remarks.

2. NCL Historical Evolution

The first version of NCL [Anto00, AMRS00] was specified through an XML DTD – Document Type Definition [XML98].

The second version of NCL, named NCL 2.0, was specified using XML Schema [SCHE01]. Following recent trends, from version 2.0 on, NCL has been specified in a modular way, allowing the combination of its modules in language profiles.

Besides the modular structure, NCL 2.0 introduced new facilities to the previous version 1.0, among others:
- definition of hypermedia connectors and connector bases;
- use of hypermedia connectors for link authoring;
- definition of ports and maps for composite nodes, satisfying the document compositionality property;
- definition of hypermedia composite-node templates, allowing the specification of constraints on documents;
- definition of composite-node template bases;
- use of composite-node templates for authoring composite nodes;
- refinement of document specifications with content alternatives, through the <switch> element, grouping a set of alternative nodes;
- refinement of document specifications with presentation alternatives, through the <descriptorSwitch> element, grouping a set of alternative descriptors;
- use of a new spatial layout model.

NCL 2.1 brought some refinements to the previous version: a module for defining cost functions associated with media object duration was introduced; a module aiming at describing the selection rules of <switch> and <descriptorSwitch> elements was defined; and refinements in some NCL modules were made, mainly in the XTemplate module.

NCL 2.2 made minor refinements in some NCL 2.1 modules, concerning their element definitions, and introduced a different approach in defining NCL modules and profiles.

NCL 2.3 introduced two new modules for supporting base and entity reuse, and refined the definition of some elements in order to support the new features.

NCL 2.4 reviewed and refined the reuse support introduced in version 2.3, and the specification of the switch and descriptor switch elements. This version also split the Timing module introduced by NCL 2.1, creating a new module to encapsulate issues related with time-scaling operations (elastic time computation using temporal cost functions) in hypermedia documents.

The NCL 3.0 edition revised some functionalities contained in NCL 2.4. NCL 3.0 is more specific regarding some attribute values. This new version introduced two new functionalities, as well: Key Navigation and Animation functionalities. In addition, NCL 3.0 made depth modifications on the Composite-Node Template functionality and introduces some SMIL based modules to NCL profiles for transition effects in media presentation and for metadata definition. NCL 3.0 also reviewed the hypermedia connector specification in order to have a more concise notation. Relationships among imperative and

declarative objects and other objects are also refined in NCL 3.0, as well as the behavior of imperative e declarative object players. Finally, NCL 3.0 also refined the support to multiple exhibition devices and introduced the support to NCL live editing commands.

NCM is the model underlying NCL. However, in its present version 3.0, NCL does not reflect all NCM 3.0 facilities yet. In order to understand NCL facilities in depth, it is necessary to understand the NCM concepts. With the aim of offering a scalable hypermedia model, with characteristics that may be progressively incorporated in hypermedia system implementations, the NCM and NCL family was divided in several parts.

The Nested Context Model is composed of Parts 1, 2, 3, and 4 of the collection:

- Part 1 – NCM Core
  concerned with the main model entities, which should be present in all NCM implementations[1].

- Part 2 – NCM Virtual Entities
  concerned mainly with the definition of virtual anchors, nodes and links.

- Part 3 – NCM Version Control
  concerned with model entities and attributes to support versioning.

- Part 4 – NCM Cooperative Work
  concerned with model entities and attributes to support cooperative document handling.

The NCL (Nested Context Language) specification is composed of Parts 5 to 12 of the collection:

- Part 5 – NCL (Nested Context Language) Full Profile
  concerned with the definition of an XML application language for authoring and exchanging NCM-based documents, using all NCL modules, including those for the definition and use of templates, and also the definition of constraint connectors, composite-connectors, temporal cost functions, transition effects and metainformation characterization.

- Part 6 – NCL (Nested Context Language) XConnector Profile Family
  concerned with the definition of an XML application language for authoring connector bases. One profile is defined for authoring causal connectors, another one for authoring causal and constraint connectors, and a third one for authoring both simple and composite connectors.

- Part 7 – Composite Node Templates
  concerned with the definition of the NCL Composite-Node Template functionality, and with the definition of an XML application language (XTemplate) for authoring template bases.

- Part 8 – NCL (Nested Context Language) Digital TV Profiles
  concerned with the definition of an XML application language for authoring documents

---

[1] It is also possible to have NCM implementations that ignore some of the basic entities, but this is not relevant so as to deserve a minimum-core definition.

aiming at the digital TV domain. Two profiles are defined: the Enhanced Digital TV (EDTV) profile and the Basic Digital TV (BDTV) profile.

- Part 9 – NCL Live Editing Commands
  concerned with editing commands used for live authoring applications based on NCL.

- Part 10 – Imperative Objects in NCL: The NCLua Scripting Language
  concerned with the definition of objects that contain imperative code and how these objects may be related with other objects in NCL applications.

- Part 11 – Declarative Hypermedia Objects in NCL: Nesting Objects with NCL Code in NCL Documents
  concerned with the definition of hypermedia objects that contain declarative code (including nested objects with NCL code) and how these objects may be related with other objects in an NCL application.

- Part 12 – Support to Multiple Exhibition Devices
  concerned with the use of multiple devices for simultaneously presenting an NCL document.

In order to understand NCL, the reading of Part 1: NCM Core is recommended.

3. Overview of NCL Elements

NCL is an XML application that follows the modularization approach. The modularization approach has been used in several W3C language recommendations. A *module* is a collection of semantically-related XML elements, attributes, and attribute's values that represents a unit of functionality. Modules are defined in coherent sets. A *language profile* is a combination of modules. Several NCL profiles have been defined, among them those defined by Parts 5, 6, 7, and 8 of the NCL collection presented in Section 2. Of special interest are the profiles defined for Digital TV, the EDTVProfile (*Enhanced Digital TV Profile*) and the BDTVProfile (*Basic Digital TV Profile*). This section briefly describes the elements that compose these profiles. The complete definition of the NCL 3.0 modules for these profiles, using XML Schemas, is presented in [SoRo06]. Any ambiguity found in this text can be clarified by consulting the XML Schemas.

The basic NCL structure module defines the root element, called <ncl>, and its children elements, the <head> element and the <body> element, following the terminology adopted by other W3C standards.

The <head> element may have <importedDocumentBase>, <ruleBase>, <transitionBase> <regionBase>, <descriptorBase>, <connectorBase>, <meta>, and <metadata> elements as its children.

The <body> element may have <port>, <property>, <media>, <context>, <switch>, and <link> elements as its children. The <body> element is treated as an NCM context node. In NCM [SoRo05], the conceptual data model of NCL, a node may be a context, a switch or a media object. Context nodes may contain other NCM nodes and links. Switch nodes contain other NCM nodes. NCM nodes are represented by corresponding NCL elements.

The <media> element defines a media object specifying its type and its content location. NCL only defines how media objects are structured and related, in time and space. As a glue language, it does not restrict or prescribe the media-object content types. However, some types are defined by the language. For example: the "application/x-ncl-settings" type, specifying an object whose properties are global variables defined by the document author or are reserved environment variables that may be manipulated by the NCL document processing; and the "application/x-ncl-time" type, specifying a special <media> element whose content is the Greenwich Mean Time (GMT).

The <context> element is responsible for the definition of context nodes. An NCM context node is a particular type of NCM composite node and is defined as containing a set of nodes and a set of links [SoRo05]. Like the <body> element, a <context> element may have <port>, <property>, <media>, <context>, <switch>, and <link> elements as its children.

The <switch> element allows the definition of alternative document nodes (represented by <media>, <context>, and <switch> elements) to be chosen during presentation time. Test rules used in choosing the switch component to be presented are defined by <rule> or <compositeRule> elements that are grouped by the <ruleBase> element, defined as a child element of the <head> element.

The NCL Interfaces functionality allows the definition of node interfaces that are used in relationships with other node interfaces. The <area> element allows the definition of content anchors representing spatial portions, temporal portions, or temporal and spatial portions of a media object (<media> element) content. The <port> element specifies a composite node (<context>, <body> or <switch> element) port with its respective mapping to an interface of one of its child components. The <property> element is used for defining a node property or a group of node properties as one of the node's interfaces. The <switchPort> element allows the creation of <switch> element interfaces that are mapped to a set of alternative interfaces of the switch's internal nodes.

The <descriptor> element specifies temporal and spatial information needed to present each document component. The element may refer a <region> element to define the initial position of the <media> element (that is associated with the <descriptor> element) presentation in some output device. The definition of <descriptor> elements shall be included in the document head, inside the <descriptorBase> element, which specifies the set of descriptors of a document. Also inside the document <head> element, the <regionBase> element defines a set of <region> elements in a class of exhibition devices, each of which may contain another set of nested <region> elements, and so on, recursively; regions define device areas (e.g. screen windows) and are referred by <descriptor> elements, as previously mentioned.

A <causalConnector> element represents a relation that may be used for creating <link> elements in documents. In a causal relation, a condition shall be satisfied in order to trigger an action. A <link> element binds (through its <bind> elements) a node interface with connector roles, defining a spatio-temporal relationship among objects (represented by <media>, <context>, <body> or <switch> elements).

The <descriptorSwitch> element contains a set of alternative descriptors to be associated with an object. Analogous to the <switch> element, the <descriptorSwitch> choice is done during the document presentation, using test rules defined by <rule> or <compositeRule> elements.

In order to allow an entity base to incorporate another already-defined base, the <importBase> element may be used. Additionally, an NCL document may be imported through the <importNCL> element. The <importedDocumentBase> element specifies a set of imported NCL documents, and shall also be defined as a child element of the <head> element.

Some important NCL element's attributes are defined in other NCL modules. The EntityReuse module allows an NCL element to be reused. This module defines the *refer* attribute, which refers to an element URI that will be reused. Only <media>, <context>, <body> and <switch> may be reused. The KeyNavigation module provides the extensions necessary to describe focus movement operations using a control device like a remote control. Basically, the module defines attributes that may be incorporated by <descriptor> elements. The Animation module provides the extensions necessary to describe what happens when a property value is changed. The change may be instantaneous, but it may also be carried out during an explicitly declared duration, either linearly or step by step. Basically, the Animation module defines attributes that may be incorporated by actions, defined as child elements of <causalConnector> elements.

Some SMIL functionalities are also incorporated by NCL. The <transition> element and some transition attributes have the same semantics of homonym element and attributes defined in the SMIL BasicTransitions module and the SMIL TransitionModifiers module. The NCL <transitionBase> element specifies a set of transition effects, defined by <transition> elements, and shall be defined as a child element of the <head> element.

Finally, the MetaInformation module is also incorporated, inheriting the same semantics of SMIL MetaInformation module. Meta-information does not contain content information that is used or display during a presentation. Instead, it contains information about content that is used or displayed. The Metainformation module contains two elements that allow describing NCL documents. The <meta> element specifies a single property/value pair. The <metadata> element contains information that is also related to meta-information of the document. It acts as the root element of an RDF tree: RDF element and its sub-elements (for more details, refer to W3C metadata recommendations [RDF99]).

4. Declarative Hypermedia Objects in NCL Applications

A <media> element of a declarative *type* (application/x-???) shall be used to specify a declarative hypermedia-object in an NCL application. In this case, the object's content (located through the *src* attribute) shall be a declarative code span to be executed. As an example, the DTV profiles of NCL 3.0 allow the application/x-ncl-NCL type, for defining NCL applications (file extension .ncl) nested in an NCL parent application.

Figure 1 illustrates an example of a declarative hypermedia-object specification with NCL codes. Note that, as usual for all <media> elements, the file extension used in the object locator makes optional the type attribute definition, which assumes a default value given by the extension.

```
<media id="nestedNCLObject" src="example.ncl">

    <property name="bounds"/>

    <property name="globalName"/>

    <area id="anchor1" clip="(chainId="entry", beginOffset="5s")"/>

</media>
```

Figure 1 – Media object with NCL code

Like any other media-object, a <media> element containing declarative code may define content anchors (through <area> elements) and properties (through <property> elements). As usual, its *descriptor* attribute may refer to a <descriptor> element that is responsible for initializing several of the corresponding properties of the media-object which are necessary for its presentation, as for example, its position on the screen.

The declarative hypermedia-object player is in charge of interpreting the semantics associated with the object's content anchors, properties and descriptors.

The declarative hypermedia-object descriptor defines, besides a player that should be used and a set of properties necessary for the presentation, a group of properties for the media-object initialization. For example, in the case of a media-object of the "application/x-ncl-NCL" type, the player, an NCL formatter, should be able to initiate values of the NCL settings object (<media type= "application/x-ncl-settings">) through values passed by the descriptor. In particular, the *region* attribute specified by the descriptor is used to initiate the system.screenSize variable of the NCL application being started.

A declarative hypermedia-object is handled by the NCL parent application as a set of temporal chains [CoMS 08]. 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 hypermedia-object presentation. Sections in these chains may be associated with declarative hypermedia-object's <area> child elements using the *clip* attribute. The *clip* value is a triple "(chainId, beginOffset, endOffset)". The *chainId* parameter identifies one of the chains defined by the declarative hypermedia-object. The *begingOffset* and *endOffset* parameters define the begin time and the end time of the content anchor, with regards the chain beginning time. When a declarative hypermedia-object defines just one

temporal chain, the *chainId* parameter may be omitted. The *begingOffset* and *endOffset* may also be omitted when they assume their default values: 0s and the chain end time, respectively.

As an example, for a declarative hypermedia-object with NCL code, a temporal chain is identified by one of the NCL document entry points, defined by <port> elements, children of the document's <body> element. In Figure 1, "anchor1" starts when the chain accessed through the <port id="entry"…> reaches 5 seconds. The anchor ends when the chain ends.

A declarative hypermedia-object's content anchor can also refer to any content anchor defined inside the declarative code itself identified by the *src* attribute of the declarative hypermedia-object. In this case, the *label* attribute of the <area> element that defines the content anchor has a value such that the declarative hypermedia-object player is able to identify one of its internally defined content anchors. As an example, for a declarative hypermedia-object with NCL code (<media type="application/x-ncl-NCL" …>) one of its <area> elements may refer to a <port> element, child of its <body> element, through its *label* attribute (that must have the <port>'s *id* as its value). In its turn, the <port> element may be mapped to a <area> element defined in any object nested in the declarative NCL hypermedia-object. Thus, note that a declarative hypermedia-object can externalize content anchors defined inside its content to be used in links defined by the NCL parent object in which the declarative hypermedia-object is included.

As usual in NCL, a declarative hypermedia-object shall have a content anchor called the *whole content anchor* declared by default in NCL documents. This content anchor, however, has a special meaning. It represents the presentation of any chain defined by the media-object. Every time a declarative hypermedia-object is started without specifying one of its content anchors, the *whole content anchor* is assumed, as usual, meaning that the presentation of every chain shall be started in parallel.

A <media type="application/x-???" …> element representing a declarative hypermedia-object may have <property> elements used both to define properties common to the whole media-object, and to externalize properties defined inside the media-object. Examples of the first group are the usual properties to parameterize the media-object player behavior, like *left, top, height, width, soundLevel, background, etc*. In the second group are properties whose *name* attribute has a value such that the declarative hypermedia-object player is able to identify one of its internally defined properties. As an example, for a declarative hypermedia-object with NCL code (<media type="application/x-ncl-NCL" …>) one of its <property> elements may refer to a <port> element, child of its <body> element, through its *name* attribute (that must have the <port>'s *id* as its value). In its turn, the <port> element may be mapped to a <property> element defined in any object nested in the declarative NCL hypermedia-object, includind its settings node.

As an example, Figure 1 defines two properties for the declarative hypermedia-object with NCL code. The first one refers to the *bounds* attribute of the object, specifying its position in the screen. The second one refers to a <port> element of the object's <body> element, whose identifier is "globalName". This <port> can be mapped to a <property> defined in a child <media> element, which will then be exposed for external use, as for example, to serve as a condition to trigger <link> elements defined external to the declarative NCL hypermedia-object, or to have its value set by an action of a <link> element defined external to the declarative NCL hypermedia-object. In this last case, the change on the

internal property value may also trigger internal procedures to the declarative hypermedia-object. For example, in the case of a declarative hypermedia-object with NCL code, the change on the internal property value may trigger <link> elements defined child element of the declarative NCL hypermedia-object.

As a consequence of all the previous discussion, <area> and <property> elements defined in a declarative hypermedia-object may be used as interface points of <link> elements, which establishes a two-way bridge between the NCL player and the declarative player that runs the declarative hypermedia-object.

5. Expected Behavior of Declarative Hypermedia Players in NCL Applications

Declarative hypermedia-objects have their life cycle controlled by their parent NCL application. This implies an execution model different from when the declarative code runs under the total control of its own engine.

Document authors may define NCL links to start, stop, pause, resume or abort the execution of a declarative code. On the other hand, a declarative code may also command the start, stop, pause or resume of its associated content anchors and properties. These transitions may be used as conditions of NCL links to trigger actions on other objects of the same NCL parent document. Thus, a two-way synchronization can be established between a declarative code and the remainder of the NCL document.

NCL links may be bound to declarative hypermedia-object interfaces (<area> and <property> elements, and the default content anchors). A declarative player (the language engine) shall interface its declarative execution environment with the NCL formatter. Analogous to conventional media content players, declarative-code players shall control event state machines associated with the declarative media-object, reporting changes to their parent NCL player. A declarative hypermedia-object shall be able to reflect in its content anchors and properties behavior changes of its temporal chains.

5.1. Instructions to Presentation Events

NCL formatters may control declarative hypermedia-object players issuing instructions that may cause changes on state machines of presentation events (sections of a temporal chain). On the other hand, any state changes on these presentation event state machines must be notified to the NCL formatter.

5.1.1. *start* instruction

The *start* instruction issued by a formatter shall inform the following parameters to the declarative hypermedia-object player: the declarative hypermedia-object to be controlled, its associated descriptor, a list of events (defined by the <media> element's <area> and <property> child elements, and by the default content anchor) that need to be monitored by the declarative hypermedia-object player, the content-anchor *label,* or by default the *whole content anchor*, identifying the associated temporal chain section to be started (called here main-event), an optional offset-time and an optional delay-time. From the *src* attribute, the declarative hypermedia-object player tries to locate the temporal chain section and start its execution. If the content cannot be located, the player shall finish the starting operation, without performing any action.

The descriptor shall be chosen by the formatter following the directives specified in the NCL document. If the *start* instruction results from a link action that has a descriptor explicitly declared in its <bind> element (*descriptor* attribute of the <link> element's children <bind> element), the resulting descriptor informed by the formatter shall merge the attributes of the bind descriptor with the attributes of the descriptor specified in the corresponding <media> element, if this attribute was specified. For the common attributes,

the <bind> descriptor information shall superpose the <media> descriptor data. If the <bind> element does not contain an explicit descriptor, the descriptor informed by the formatter shall be the <media> descriptor, if this attribute was specified. Otherwise, a default descriptor for that imperative-object *type* of <media> shall be chosen by the formatter.

The list of events to be monitored by a declarative hypermedia-object player should also be computed by the NCL formatter, taking into account the NCL document specification. The formatter shall check all links where the declarative hypermedia-object and the resulting descriptor participate. When computing the events to be monitored, the formatter shall take into account the media-object perspective, i.e., the path of <body> and <context> elements to reach the <media> element. Only links contained in these <body> and <context> elements should be considered to compute the monitored events.

As with any other <media> element, the delay-time is an optional parameter and its default value is "zero". If greater than zero, this parameter contains a time to be waited by the declarative hypermedia-object player before starting the code execution.

The offset-time parameter is optional, it has "zero" as its default value. This parameter defines a time offset from the beginning (beginning-time) of the main-event, from which the presentation of the main-event shall be immediately started (i.e., it commands the player to jump to the beginning-time + offset-time). Obviously, the offset-time value shall be lower than the main-event duration. If the offset-time is greater than zero, the media player shall put the main-event in the *occurring* state, but the event *starts* transition shall not be notified. If the offset-time is zero, the media player shall put the main-event in the *occurring* state and notify the *starts* transition occurrence. Events that would have their end-times previous to the beginning-time of the main-event and events that would have their beginning times after the end-time of the main-event do not need to be monitored by the media player (the formatter should do this verification when building the monitored event list). Monitored events that would have beginning-times before the beginning-time of the main-event and end-times after the beginning-time of the main-event shall be put in the *occurring* state, but their *starts* transitions shall not be notified (links that depend on this transition shall not be fired). Monitored events that would have their end times after the main-event beginning-time, but before the start time (beginning-time + offset-time) shall have their *occurrences* attribute incremented but the *starts* and *stops* transitions shall not be notified. Monitored events that have beginning-times before the start time (beginning-time + offset-time) and end time after the start time shall be put in the *occurring* state, but the corresponding *starts* transition shall not be notified.

The delay-time is also an optional parameter and its default value is "zero" too. If greater than zero, this parameter contains a time to be waited by the media player before starting the presentation. This parameter shall only be considered if the offset-time parameter is equal to "zero".

If a declarative hypermedia-object player receives a *start* instruction for a temporal chain already being presented (paused or not), it shall ignore the instruction and keep on controlling the ongoing presentation. However, different from what is performed on other <media> elements, if the start instruction is for a temporal chain that is not being presented, the instruction must be executed even if another temporal chain is being presented (paused or occurring). As a consequence, different from what happens for other <media> elements,

a <simpleAction> element with an *actionType* attribute equal to "stop", "pause", "resume" or "abort" shall be bound through a link to a declarative hypermedia-object's interface, which shall not be ignored when the action is applied.

Different from other <media> elements, if any content anchor is started and the event associated with the *whole content anchor* is in *sleeping* or *paused* state, it shall be put in the *occurring* state and the corresponding transition shall be notified.

5.1.2. *stop* instruction

The *stop* instruction needs to identify a temporal chain already being controlled (or by default, all of them). To identify the temporal chain means to identify the corresponding <media> element, the corresponding descriptor, a <media> element's interface and the declarative hypermedia-object perspective.

The *stop* instruction issued by an NCL formatter shall be ignored by a declarative hypermedia-object player if the temporal chain associated with the specified interface is not being presented (if none of the events in the object list of events is in the *occurring* or *paused* state) and the declarative hypermedia-object player is not waiting to exhibit that temporal chain due to a delayed *start* instruction. If the temporal chain associated with the specified interface is being presented, the main-event (the event passed as a parameter when the temporal chain was started) and all monitored events of this temporal chain in the *occurring* or in the *paused* state with end time equal or previous to the main-event end time shall transit to the *sleeping* state, and their *stops* transitions shall be notified. Monitored events in the *occurring* or in the *paused* state with end time posterior to the main-event end time shall be put in the *sleeping* state, but their *stops* transitions shall not be notified and their *occurrences* attribute shall not be incremented. The temporal chain presentation shall be stopped. If the *repetitions* event attribute is greater than zero, it shall be decremented by one and the main-event presentation shall restart after the repeat delay time (the repeat delay shall have been passed to the media player as the start delay parameter). If the temporal chain associated with the specified interface is waiting to be presented after a delayed *start* instruction and a *stop* instruction is issued, the previous *start* instruction shall be removed.

Different from other <media> elements, if any content anchor is stopped and all other presentation events are in the *sleeping* state the *whole content anchor* shall be put in the *sleeping* state. If a content anchor is stopped and at least one other presentation event is in the *occurring* state the *whole content anchor* shall remain in the *occurring* state. In all other cases, if a content anchor is stopped the *whole content anchor* shall be put in the *paused* state. If the *stop* instruction is applied to a declarative hypermedia-object without specifying the node's interface, the *whole content anchor* is assumed. In this case, *stop* instructions shall be issued for all temporal chains.

5.1.3. *abort* instruction

The *abort* instruction needs to identify a temporal chain already being controlled (or by default, all of them). To identify the temporal chain means to identify the corresponding <media> element, the corresponding descriptor, a <media> element's interface and the declarative hypermedia-object perspective.

The *abort* instruction issued by an NCL formatter shall be ignored by a declarative hypermedia-object player if the temporal chain associated with the specified interface is not being presented (if none of the events in the object list of events is in the *occurring* or *paused* state) and the declarative hypermedia-object player is not waiting to exhibit that temporal chain due to a delayed *start* instruction. If the temporal chain associated with the specified interface is being presented, the main-event (the event passed as a parameter when the temporal chain was started) and all monitored events of this temporal chain in the *occurring* or in the *paused* state, shall transit to the *sleeping* state, and their *aborts* transitions shall be notified. The temporal chain presentation shall be stopped. If the *repetitions* event attribute is greater than zero, it shall be set to zero and the temporal chain presentation shall not restart. If the temporal chain associated with the specified interface is waiting to be presented after a delayed *start* instruction and an *abort* instruction is issued, the previous *start* instruction shall be removed.

Different from other <media> elements, if any content anchor is aborted and all other presentation events are in the *sleeping* state the *whole content anchor* shall be put in the *sleeping* state. If a content anchor is aborted and at least one other presentation event is in the *occurring* state the *whole content anchor* shall remain in the *occurring* state. In all other cases, if a content anchor is stopped the *whole content anchor* shall be put in the *paused* state. If the *abort* instruction is applied to a declarative hypermedia-object without specifying the node's interface, the *whole content anchor* is assumed. In this case, *abort* instructions shall be issued for all temporal chains.

5.1.4. *pause* instruction

The *pause* instruction needs to identify a temporal chain already being controlled (or by default, all of them). To identify the temporal chain means to identify the corresponding <media> element, the corresponding descriptor, a <media> element's interface and the declarative hypermedia-object perspective.

The *pause* instruction issued by an NCL formatter shall be ignored by a declarative hypermedia-object player if the temporal chain associated with the specified interface is not being presented (if none of the events in the object list of events is in the *occurring* or *paused* state) and the declarative hypermedia-object player is not waiting to exhibit that temporal chain due to a delayed *start* instruction. If the temporal chain associated with the specified interface is being presented, the main-event (the event passed as a parameter when the temporal chain was started) and all monitored events of this temporal chain in the *occurring* shall transit to the *paused* state and their *pauses* transitions shall be notified. The temporal chain presentation shall be paused and the pause elapsed time shall not be considered as part its duration.

If the temporal chain associated with the specified interface is waiting to be presented after a delayed *start* instruction and a *pause* instruction is issued, the temporal chain shall wait for a resume instruction to continue waiting for the remaining start delay.

Different from other <media> elements, if any content anchor is paused and all other presentation events are in the *sleeping* state or *paused* state the *whole content anchor* shall be put in the *paused* state. If a content anchor is paused and at least one other presentation event is in the *occurring* state the *whole content anchor* shall remain in the *occurring* state. If the *pause* instruction is applied to a declarative hypermedia-object without specifying the

node's interface, the *whole content anchor* is assumed. In this case, *pause* instructions shall be issued for all other content anchors that are in the occurring state.

5.1.5. *resume* instruction

The *resume* instruction needs to identify a temporal chain already being controlled (or by default, all of them). To identify the temporal chain means to identify the corresponding <media> element, the corresponding descriptor, a <media> element's interface and the declarative hypermedia-object perspective.

The *resume* instruction issued by an NCL formatter shall be ignored by a declarative hypermedia-object player if the temporal chain associated with the specified interface is not paused or the declarative hypermedia-object player is not waiting to exhibit that temporal chain due to a delayed *start* instruction. If the declarative hypermedia-object player is paused waiting for the start delay, it shall resume the wait from the instant it was paused. If the temporal chain is in the *paused* state, the main-event and all monitored events in the *paused* state shall be put in the *occurring* state and their *resumes* transitions shall be notified.

Different from other <media> elements, if any content anchor is resumed, the *whole content anchor* shall be set to the *occurring* state. If the *resume* instruction is applied to a declarative hypermedia-object without specifying the node's interface, the *whole content anchor* is assumed. If the *whole content anchor* is not in the *paused* state due to a previous receive of a *pause* instruction, the *resume* instruction is ignored. Otherwise, *resume* instructions shall be issued for all other content anchors that are in the *paused* state, except those that were already paused before the *whole content anchor* received the *paused* instruction.

5.1.6. *Natural end* of a temporal chain section presentation

Events of a declarative hypermedia-object normally end their execution naturally, without needing external instructions. In this case, the declarative hypermedia-object player shall transit the event to the *sleeping* state and notify the *stops* transition. The same shall be done for monitored events of the same temporal chain in the *occurring* state with the same end time of the main-event or with unknown end time, when the main-event ends. Events chain of the same temporal in the *occurring* state with end time posterior to the main-event end time shall be put in the sleeping state but without generating the *stops* transition and without incrementing the *occurrences* attribute.

In the case of a natural end of a main-event, if the *repetitions* event attribute is greater than zero, it shall be decremented by one and the main-event presentation shall restart after the repeat delay time (the repeat delay shall have been passed to the media player as the start delay parameter).

Different from other <media> elements, if any content anchor execution ends and all other presentation events are in the *sleeping* state the *whole content anchor* shall be put in the *sleeping* state. If a content anchor execution ends and at least one other presentation event is in the *occurring* state the *whole content anchor* shall remain in the *occurring* state. In all other cases, if a content anchor execution ends, the *whole content anchor* shall be set to the *paused* state.

5.2. Instructions to Attribution Events

NCL formatters may also send instructions that may cause changes on state machines of attribution events. Similarly to presentation events, any state changes on attribution event state machines are notified to the NCL formatter.

5.2.1. *start* instruction

The *start* instruction issued by a formatter may be applied to a declarative hypermedia-object's property independent from the fact whether the object is being in execution (the *whole content anchor* is in the *occurring* state) or not (in this latter case, although the object is not being executed, its declarative hypermedia-object player shall have already been instantiated). In the first case, the *start* instruction needs to identify the declarative hypermedia-object, a monitored attribution event, and, if it is the case, a value to be to be assigned to the attribute wrapped by the event. In the second case, the instruction shall also identify the <descriptor> element that will be used when presenting the object (as it is done for the *start* instruction for presentation). When setting a value to an attribute, the declarative hypermedia-object player shall set the event state machine to the *occurring* state, and after finishing the attribution, again to the *sleeping* state, generating the *starts* transition and afterwards the *stops* transition.

For every monitored attribution event, if a declarative hypermedia-object player changes by itself the corresponding attribute value, it shall proceed as if it had received an external *start* instruction.

5.2.2. *stop, abort, pause and resume* instructions

With the exception of the *start* instruction, discussed in the previous section, all other instructions has the same effect on the corresponding property attribution as they have on any property attribution of any type of NCL object.

The *stop* instruction only stops the property attribution procedure, bringing the attribution event state machine to the *sleeping* state.

The *abort* instruction stops the property attribution procedure, bringing the attribution event state machine to the *sleeping* state and the property value to its original one.

The *pause* instruction only pauses the property attribution procedure, bringing the attribution event state machine to the *paused* state.

Finally, the *resume* instruction only resumes the property attribution procedure, bringing the attribution event state machine to the *occurring* state.

6. Final Remarks

In order to offer a scalable hypermedia model, with characteristics that may be progressively incorporated in hypermedia system implementations, NCM was divided in several parts, and also its declarative XML application language: NCL. This technical report deals with how declarative hypermedia-objects may be related with other objects in NCL applications and how declarative hypermedia-object players shall behave, which comprises Part 11 – Declarative Hypermedia Objects in NCL.

References

[Anto00]   Antonacci M.J. NCL: Uma Linguagem Declarativa para Especificação de Documentos Hipermídia com Sincronização Temporal e Espacial. **Master Dissertation, Departamento de Informática, PUC-Rio**, April 2000.

[AMRS00] Antonacci M.J., Muchaluat-Saade D.C., Rodrigues R.F., Soares L.F.G. NCL: Uma Linguagem Declarativa para Especificação de Documentos Hipermídia na Web, **VI Simpósio Brasileiro de Sistemas Multimídia e Hipermídia - SBMídia2000**, Natal, Rio Grande do Norte, June 2000.

[CoMS 08] Costa R. R; Moreno, M. F.; Soares, L. F. G. Intermedia Synchronization Management in DTV Systems. *Proceedings of the ACM Symposium on Document Engineering*. São Paulo, Brazil. September 2008; pp. 289-297. ISBN: 978-1-60558-081-4.

[RDF99]   Resource Description Framework (RDF) Model and Syntax Specification, Ora Lassila and Ralph R. Swick. W3C Recommendation, 22 February 1999. Available at http://www.w3.org/TR/REC-rdf-syntax/

[SCHE01] XML Schema Part 0: Primer, **W3C Recommendation**, in http://www.w3.org/TR/xmlschema-0/, May 2001.

[SoRo05] Soares L.F.G; Rodrigues R.F. Nested Context Model 3.0: Part 1 – NCM Core, **Technical Report, Departamento de Informática PUC-Rio**, May 2005, ISSN: 0103-9741.

[SoRo06] Soares L.F.G; Rodrigues R.F. Nested Context Language 3.0: Part 8 – NCL Live Editing Commands, **Technical Report, Departamento de Informática PUC-Rio**, December 2006, ISSN: 0103-9741.

[XML98]   Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. Extensible Markup Language (XML) 1.0 (Second Edition), **W3C Recommendation**, in http://www.w3.org/TR/REC-xml, February 1998.