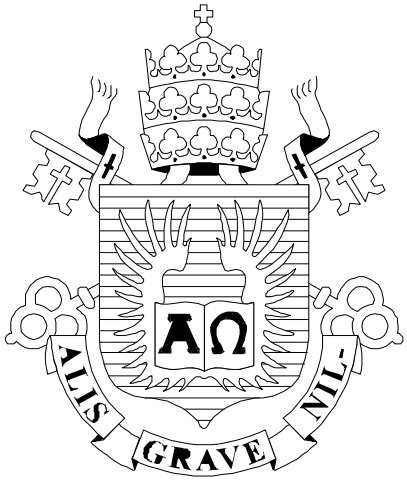

PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 35/06

Nested Context Language 3.0

Part 8 – NCL Digital TV Profiles

Luiz Fernando Gomes Soares

Rogério Ferreira Rodrigues

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 35/06 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Outubro/2006

Nested Context Language 3.0

Part 8 – NCL Digital TV Profiles

Luiz Fernando Gomes Soares
Rogério Ferreira Rodrigues

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

lfgs@inf.puc-rio.br, rogerio@telemidia.puc-rio.br

Abstract. *This technical report describes the elements and attributes of the Nested Context Language (NCL) Basic Digital TV (BDTV) and Enhanced Digital TV (EDTV) Profiles version 3.0. NCL 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: *digital TV; middleware; declarative language; NCL.*

Resumo. *Este relatório técnico descreve os elementos e atributos dos perfis "Enhanced Digital TV (EDTV)"e "Basic Digital TV (BDTV), da versão 3.0 da linguagem NCL (Nested Context Language). 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: *Tv digital; middleware; linguagem declarativa; NCL.*

Nested Context Language 3.0
Part 8 – NCL Digital TV Profiles

Laboratório TeleMídia

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rua Marquês de São Vicente, 225, Prédio ITS - Gávea

22451-900 – Rio de Janeiro – RJ – Brasil

http://www.telemidia.puc-rio.br

Table of Contents

Nested Context Language 3.0
Part 8 – NCL Digital TV Profiles

Luiz Fernando Gomes Soares
Rogério Ferreira Rodrigues

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

lfgs@inf.puc-rio.br, rogerio@telemidia.puc-rio.br

Abstract. . *This technical report describes the elements and attributes of the Nested Context Language (NCL) Basic Digital TV (BDTV) and Enhanced Digital TV (EDTV) Profiles version 3.0. NCL 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

This report describes the entities of the NCL (*Nested Context Language*) Basic Digital TV (BDTV) and Enhanced Digital TV (EDTV) Profiles. NCL is an XML application language for authoring hypermedia documents, including non-linear TV programs. NCL is based on NCM (Nested Context Model) [SoRo05].

The version number of an NCL document consists of a major number and a minor number, separated by a dot. The numbers are represented as a decimal number character string with leading zeros suppressed. This report deals with the NCL version 3.0 (NCL 3.0).

From version 3.0 on, if formatters[1] that conform to older versions can still receive a document based on the revised specification, in relation to error corrections or operational reasons, the minor version number must be updated. If formatters that conform to older versions cannot receive a document based on the revised specifications, the major number must be updated.

A specific module of version x.x is specified in the URI path http://www.ncl.org.br/NCLx.x/modules/moduleName, where the version number is written immediately after the NCL. Similarly, a specific profile of version x.x is specified in the URI path http://www.ncl.org.br/NCLx.x/profiles/profileName. In the case of this report, the version number (x.x) is 3.0, and the profileName, in the URI path, must be NCL30EDTV.xsd or NCL30BDTV.xsd.

This technical report is organized as follows. Section 2 gives an historical evolution of the NCL versions. Section 3 presents an overview of the NCL 3.0 modules that compose the Basic DTV and Enhanced DTV profiles. Section 4 introduces the XML Schemas of these NCL modules. Section 5 introduces the EDTV and BDTV language profiles and presents their XML Schemas. Section 6 gives an example of authoring an NCL document, presenting some of the main NCL elements and their attributes. Section 7 presents the final remarks.

[1] Document renderer, user agent, and player are other names used with the same meaning of document formatter.

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 and introduces some SMIL modules to NCL profiles, specially aiming at the Digital TV domain. NCL 3.0 also reviewed the hypermedia connector specification in order to have a more concise notation.

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, it is necessary to

understand the NCM concepts. With the aim of offering a scalable hypermedia model, with characteristics that can be progressively incorporated in hypermedia system implementations, the NCM and NCL family was divided in several parts. The Nested Context Model is composed by 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[2].
- 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 Nested Context Language specification is composed by Parts 5, 6, 7, 8 and 9 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) Main Profile – concerned with the definition of an XML application language for authoring and exchanging NCM-based documents, without the modules for template use and definition; for constraint connector and composite-connector specification; for temporal cost function definition; for transition effects definition and for metainformation characterization.
- Part 7: NCL (Nested Context Language) XConnector Profile Family– concerned with the definition of an XML application languages 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 8: NCL (Nested Context Language) Digital TV Profiles – concerned with the definition of an XML application languages for authoring documents 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.

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

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

3. The NCL 3.0 Modularization

This section describes the entities of the NCL (*Nested Context Language*) version 3.0 that compose the Basic DTV and Enhanced DTV profiles.

This section is organized as follows. Section 3.1 introduces the concepts of modular languages and language profiles. Section 3.2 presents an overview of NCL 3.0 modules and the extensions used in the definition of the Enhanced DTV profiles. Section 3.3 presents an overview of extensions used in the definition of the Basic DTV profiles. The XML Schemas of these modules are presented in Section 4.

3.1. Modular Languages and Language Profiles

Modules are collections of semantically-related XML elements, attributes, and attribute values that represent a unit of functionality. Modules are defined in coherent sets. This coherency is expressed in that the elements of these modules are associated with the same namespace [W3C99].

A *language profile* is a combination of modules. Modules are *atomic*, i.e. they cannot be subset when included in a language profile. Furthermore, a module specification may include a set of integration requirements, to which language profiles that include the module must comply.

NCL has been specified in a modular way since version 2.0, allowing the combination of its modules in language profiles. Each profile can group a subset of NCL modules, allowing the creation of languages according to the users' needs. Moreover, NCL modules and profiles can be combined with other language modules, allowing the incorporation of NCL features into those languages and vice-versa.

Commonly, there is a language profile that incorporates nearly all the modules associated with a single namespace. This is the case of the NCL language profile (or NCL Full profile).

Other language profiles can be specified as subsets of the larger one, or to incorporate a combination of modules associated with different namespaces. An example of the latter is the use of SMIL modules (for example the SMIL Transition Module) in NCL profiles. Examples of the first case are the Basic DTV and the Enhanced DTV profiles.

Subsets of the language profile modules used in the definition of the Basic DTV and Enhanced DTV profiles are defined to fit the language to the data TV broadcasting environment with its multiple possible presentation devices: TV set, mobile devices, etc. A similar approach is also found in other languages [SMIL05] [W3C02].

The main purpose of language profile conformance is to enhance interoperability. The mandatory modules are defined in such a way that any document interchanged in a conforming language profile will yield a reasonable presentation. The document formatter, while supporting the associated mandatory module set, would ignore all other (unknown) elements and attributes.

NCL 3.0 is partitioned into twelve functional areas, which are partitioned again into modules. From the twelve functional areas, eleven are used to define the Enhanced DTV and the Basic DTV profiles. Besides the eleven NCL 3.0 functional areas, two functional areas have been imported from SMIL 2.0 [SMIL05] to compose the profiles. The thirteen used functional areas and their corresponding modules are:

1. Structure
   a. Structure Module

2. Layout
   a. Layout Module

3. Components
   a. Media Module
   b. Context Module

4. Interfaces
   a. MediaContentAnchor Module
   b. CompositeNodeInterface Module
   c. AttributeAnchor Module
   d. SwitchInterface Module

5. Presentation Specification
   a. Descriptor Module

6. Linking
   a. Linking Module

7. Connectors
   a. ConnectorCommonPart Module
   b. ConnectorAssessmentExpression Module
   c. ConnectorCausalExpression Module
   d. ConnectorTransitionAssessment Module
   e. CausalConnector Module
   f. CausalConnectorFunctionality Module
   g. ConnectorBase Module

8. Presentation Control
   a. TestRule Module
   b. TestRuleUse Module
   c. ContentControl Module
   d. DescriptorControl Module

9. Timing
   a. Timing Module

10. Reuse
   a. Import Module
   b. EntityReuse Module
   c. ExtendedEntityReuse Module

11. Navigational Key
    a. KeyNavigation Module

12. SMIL Transition Effects
    a. TransitionBase Module[3]
    b. BasicTransition Module
    c. TransitionModifiers Module

13. SMIL Meta-Information
    a. Metainformation Module

3.1.1. Identifiers for NCL 3.0 Module and Language Profiles

Each NCL profile should explicitly state the namespace URI that is to be used to identify it.

Documents authored in language profiles that include the NCL Structure module can be associated with the "application/x-ncl+xml" mime type. Documents using the "application/x-ncl+xml" mime type are required to be host language conformant.

The XML namespace identifier for the complete set of NCL 3.0 modules, elements and attributes, are contained within the following namespace:

- http://www.ncl.org.br/NCL3.0/

Each NCL module has a unique identifier associated with it. Table 3.1 summarizes the identifiers for NCL 3.0 modules.

Each SMIL module used in the Enhanced DTV and the Basic DTV profiles has a unique identifier associated with it. Table 3.2 summarizes the identifiers for SMIL 2.0 modules.

Modules can also be identified collectively. The following module collections are defined:

- http://www.ncl.org.br/NCL3.0/LanguageProfile

The modules used by the NCL 3.0 Language profile.

- http://www.ncl.org.br/NCL3.0/CausalConnectorProfile

The modules used by the NCL 3.0 Causal Connector profile.

- http://www.ncl.org.br/NCL3.0/EDTVProfile

The modules used by the NCL 3.0 Enhanced DTV profile.

- http://www.ncl.org.br/NCL3.0//BDTVProfile

The modules used by the NCL 3.0 Basic DTV profile.

[3] Actually, the TransitionBase Module is a module defined by NCL 2.4; it does not exist in SMIL 2.0.

Table 3.1 – The NCL 3.0 Module Identifiers

AttributeAnchor	http://www.ncl.org.br/NCL3.0/AttributeAnchor
CompositeNodeInterface	http://www.ncl.org.br/NCL3.0/CompositeNodeInterface
CausalConnector	http://www.ncl.org.br/NCL3.0/CausalConnector
CausalConnectorFunctionality	http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality
ConnectorCausalExpression	http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression
ConnectorAssessmentExpression	http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression
ConnectorBase	http://www.ncl.org.br/NCL3.0/ConnectorBase
ConnectorCommonPart	http://www.ncl.org.br/NCL3.0/ConnectorCommonPart
ConnectorTransitionAssessment	http://www.ncl.org.br/NCL3.0/ConnectorTransitionAssessment
ContentControl	http://www.ncl.org.br/NCL3.0/ContentControl
Context	http://www.ncl.org.br/NCL3.0/Context
Descriptor	http://www.ncl.org.br/NCL3.0/Descriptor
DescriptorControl	http://www.ncl.org.br/NCL3.0/DescriptorControl
EntityReuse	http://www.ncl.org.br/NCL3.0/EntityReuse
ExtendedEntityReuse	http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse
Import	http://www.ncl.org.br/NCL3.0/Import
Layout	http://www.ncl.org.br/NCL3.0/Layout
Linking	http://www.ncl.org.br/NCL3.0/Linking
Media	http://www.ncl.org.br/NCL3.0/Media
MediaContentAnchor	http://www.ncl.org.br/NCL3.0/MediaContentAnchor
KeyNavigation	http://www.ncl.org.br/NCL3.0/KeyNavigation
Structure	http://www.ncl.org.br/NCL3.0/Structure
SwitchInterface	http://www.ncl.org.br/NCL3.0/SwitchInterface
TestRule	http://www.ncl.org.br/NCL3.0/TestRule
TestRuleUse	http://www.ncl.org.br/NCL3.0/TestRuleUse
Timing	http://www.ncl.org.br/NCL3.0/Timing
TransitionBase	http://www.ncl.org.br/NCL3.0/TransitionBase

Table 3.2 – The SMIL 2.0 Module Identifiers used in NCL profiles

BasicTransitions	http://www.w3.org/2001/SMIL20/BasicTransitions
Metainformation	http://www.w3.org/2001/SMIL20/Metainformation

3.1.2. NCL Version Information

The following processing instructions must be written in an NCL document. They identify NCL documents, and the NCL version to which the document conforms.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="any string"
xmlns="http://www.ncl.org.br/NCL3.0/profileName">
```

The <ncl> element *id* attribute may receive any string as a value.

The version number of an NCL document consists of a major number and a minor number, separated by a dot. The numbers are represented as a decimal number character string with leading zeros suppressed. This reported is related with version number 3.0.

3.2. NCL Modules

This section briefly describes the main definitions made by each NCL 3.0 modules that are present in the NCL 3.0 Basic DTV and the Enhanced DTV profiles. The complete definition of these NCL 3.0 modules, using XML Schemas, is presented in Section 4. Any ambiguity found in this text may be clarified by consulting the XML Schemas.

This section also briefly describes the main definitions made by each SMIL 2.0 modules that are present in the Enhanced DTV profile. The complete definition of the modules imported from the SMIL 2.0 specification can be found in [SMIL05].

After discussing each module, a table is presented indicating the module elements and their attributes. For a given profile, the attributes and contents (child elements) of elements can be defined in the module itself or in the language profile that groups the modules. Therefore, the tables in this section show the attributes and contents that come from the NCL Enhanced DTV profile, besides those defined in the modules themselves. Tables in Section 3.3 show the attributes and contents that come from the NCL Basic DTV profile, besides those defined in the modules themselves. Element attributes that are required are underlined. In the tables, the following symbols are used: (?) optional (zero or one occurrence), (|) or, (*) zero or more occurrences, (+) one or more occurrences. The child element order is not specified in the tables.

3.2.1. Structure Functionality

The Structure functionality has just one module, called Structure, which defines the basic structure of an NCL document. It defines the root element, called <ncl>, the <head> element and the <body> element, following the terminology adopted by other W3C standards. The <body> element of an NCL document is treated as a NCM context node.

In NCM [SoRo05], the conceptual data model of NCL, a node can be a context, a switch or a media object. All NCM nodes are represented by corresponding NCL elements. Context nodes, as defined in Section 3.2.3, contain other NCM nodes and links.

The <ncl> and <body> elements may have the *id* attribute defined. The *id* attribute uniquely identifies an element within a document. Its value is an XML identifier.

The *title* attribute of <ncl> offers advisory information about the element. Values of the title attribute may be rendered by user agents in a variety of ways.

The *xmlns* attribute declares an XML namespace — that is, it declares the primary collection of XML-defined constructs the document uses. The attribute's value is the URL (Uniform Resource Locator) identifying where the namespace is officially defined. Two values allowed for the *xmlns* attribute are: http://www.ncl.org.br/NCL3.0/EDTVProfile,

and http://www.ncl.org.br/NCL3.0/BDTVProfile, for the Enhanced and Basic DTV profiles, respectively. An NCL formatter must know that the schemaLocation for these namespaces are, by default, respectively:
http://www.ncl.org.br/NCL3.0/profiles/NCL30EDTV.xsd,
http://www.ncl.org.br/NCL3.0/profiles/NCL30BDTV.xsd.

Child elements of <head> and <body> are defined in other NCL modules. The order in which the <head> elements must be declared is: importedDocumentBase?, ruleBase?, transitionBase?, regionBase*, descriptorBase?, connectorBase?, meta*, metadata*.

Table 3.3 presents the elements of this module, their child elements, and their attributes.

Table 3.3 – Extended Structure Module used in the Enhanced DTV profile

Elements	Attributes	Content
ncl	*id, title, xmlns*	*(head?, body?)*
head		*(importedDocumentBase?, ruleBase?, transitionBase?, regionBase*, descriptorBase?, connectorBase?, meta*, metadata*)*
body	*id*	*(port| attribute| media| context| switch| link)**

3.2.2. Layout Functionality

The Layout functionality has a single module, named Layout, which specifies elements and attributes that define how objects will be initially presented inside regions of output devices. Indeed, this module defines initial values for homonym NCL attributes defined in <media>, <body>, and <context> elements, discussed in Section 3.2.3.

In short, a <regionBase> element, which must be declared in the NCL document <head>, defines a set of <region> elements, each of which may contain another set of nested <region> elements, and so on, recursively.

The <regionBase> element may have the *id* attribute, and <region> elements must have the *id* attribute. As usual, the *id* attribute uniquely identifies the element within a document.

Each <regionBase> element is associated with a device where presentation will take place. In order to identify the association, the <regionBase> element defines the *device* attribute, which can have the values: “systemScreen (i)” or “systemAudio(i)”. The chosen device defines global environment variables: systemScreenSize(i), systemScreenGraphicSize(i), and systemAudioType(i), as defined in Table 3.6 and discussed in Section 3.2.3).

The interpretation of the region nesting inside a <regionBase> should be made by the software in charge of the document presentation orchestration (called *formatter*). For example, in the ISDTV-T recommendation, a first nesting level must be interpreted as defining the device area where the presentation would take place; the second nesting level

as windows (i.e. presentation areas in the screen) of the parent area; and the other levels as regions inside these windows.

A <region> can also define the following attributes: *title*, *left*, *right, top, bottom*, *height*, *width*, and *zIndex*. All those attributes have the usual meaning.

The position of a region, as specified by its *top*, *bottom*, *left*, and *right* attributes, is always relative to the parent geometry, which is defined by the parent <region> element or the total device area in the case of first nesting level regions. Attribute values can be non-negative "percentage" values, or pixel units. For pixel values, the author may omit the "px" unit qualifier (e.g. "100"). For percentage values, on the other hand, the "%" symbol must be indicated (e.g. "50%"). The percentage is always relative to the parent's width, in the case of *right*, *left* and *width* definitions, and parent's height, in the case of *bottom*, *top* and *height* definitions.

The *top* and *left* attributes are the primary region positioning attributes. They place the left-top corner of the region in the specified distance away from the left-top edge of the parent region (or the device left-top edge in the case of the outermost region). Sometimes, explicitly setting the *bottom* and *right* attributes is helpful. Their values state the distance between the region's right-bottom corner and the right-bottom corner of the parent region (or the device right-bottom edge in the case of the outermost region). Figure 3.1 illustrates the relative values of these attributes.

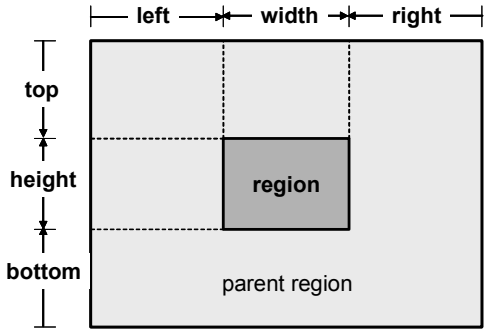

Figure 3.1 – Region positioning attributes

Regarding region sizes, when they are specified by declaring *width* and *height* attributes using the "%" notation, the size of the region is relative to the size of its parent geometry as aforementioned. Sizes declared as absolute pixel values maintain those absolute values. The intrinsic size of a region is equal to the size of the logical parent's geometry. This means that, if a nested region doesn't specify any positioning or size values, it will be assumed to have the same position and size values of its parent region. In particular, when a first level region doesn't specify any positioning or size values, it will be assumed to be the whole device presentation area.

When the user specifies *top*, *bottom* and *height* information for the same <region>, spatial inconsistencies may occur. In this case, the *top* and *height* values must have precedence over the *bottom* value. Analogously, when the user specifies inconsistent values for *left*, *right* and *width* <region> attributes, the *left* and *width* values must be used to compute a new *right* value. When any of these attributes is not specified and cannot have its value computed from the other attributes, it must be inherited from the corresponding parent

value. Another restriction is that child regions cannot stay outside the area established by their parent regions.

The *zIndex* attribute specifies the region superposition precedence, where regions with greater *zIndex* values are stacked on top of regions with smaller *zIndex* values. If two presentations generated by elements A and B have the same stack level then, if the display of an element B starts later than the display of an element A, the presentation of B is stacked on top of the presentation of A (temporal order); otherwise, if the display of the elements starts at the same time, the stacked order is chosen arbitrarily by the formatter.

The Layout module also defines the *region* attribute to be used by a <descriptor> element (as discussed in Section 3.2.5) to refer a Layout <region> element.

Table 3.4 presents the elements of this module, their child elements, and their attributes.

Table 3.4 – Extended Layout Module in the Enhanced DTV profile

Elements	Attributes	Content
regionBase	*id, device*	*(importBase|region)+*
region	*id, title, left, right, top, bottom, height, width, zIndex*	*(region)**

3.2.3. Components Functionality

The Components functionality is partitioned into two modules, called Media and Context.

The Media module defines basic media object types. For defining media objects, this module defines the <media> element. Each media object has two main attributes, besides its *id* attribute: *src*, which defines a URI [IETF98] of the object content, and *type*, which defines the object type.

The URI (Uniform Resource Identifier) allowed are shown in Table 3.5

Table 3.5 – Allowed URIs

Scheme	Scheme-Specific-Part	Use
file:	///file_path/#fragment_identifier	for local files
http:	//server_identifier/file_path/#fragment_identifier	for remote files downloaded from the return channel using the http protocol
rstp:	//server_identifier/file_path/#fragment_identifier	for streams downloaded from the return channel using the rstp protocol
rtp:	//server_identifier/file_path/#fragment_identifier	for streams downloaded from the return channel using the rtp protocol

Scheme	Scheme-Specific-Part	Use
isdtv-ts:	//program_id	for streams received from the transport stream as defined by the ISDTV-T Standard

An absolute URI by itself contains all information needed to locate its resource. Relative URIs are also allowed. Relative URIs are incomplete addresses that are applied to a base URI to complete the location. The portions omitted are the URI scheme and server, and potentially part of URI path, as well.

The primary benefit of using relative URIs is that documents and directories containing them can be moved or copied to other locations without requiring changing the URI attribute values within the documents. This is especially interesting when transporting documents from the server part (usually broadcasters) to the receptors. Relative URI paths are typically used as a short means of locating media files stored in the same directory as the current NCL document, or in a directory close to it. They often consist of just the filename (optionally with a fragment identifier into the file). They can also have a relative directory path before the filename.

It should be emphasized that references to streaming video or audio resources shall not cause tuning to occur. References that imply tuning to access a resource shall behave as if the resource were unavailable.

The *type* attribute's allowed values depend on the NCL profile and must follow MIME Media Types format (or, more simply, mimetypes). A mimetype is a character string that defines the class of media (audio, video, image, text, application) and a media encoding type (such as jpeg, mpeg, etc.). Mimetypes may be registered or informal. Registered mimetypes are controlled by the Internet Assigned Numbers Authority (IANA). Informal mimetypes are not registered with IANA, but are defined by common agreement; they usually have an "x-"before the media type name.

Four special types are defined: application/x-NCLua, application/x-NCLet, application/x-NCL-settings, and application/x-NCL-time. The application/x-NCLua type must be applied to <media> elements with Lua procedural code content [ISDTV-T N06 V02]. The application/x-NCLet type must be applied to <media> elements with Xlet procedural code content [ISDTV-T N06 V04]. The application/x-NCL-settings must be applied to a special <media> element (it can be only one in an NCL document) whose attributes are global variables defined by the document author or reserved environment variables that can be manipulated by the NCL document processing. Table 3.6 states the already defined environment variables and their semantics. The application/x-NCL-time type must be applied to a special <media> element (it can be only one in an NCL document) whose content (*src* attribute) is the Greenwich Mean Time (GMT). Note that any continuous <media> element with no source can be used to define a clock relative to the <media> element start time.

Table 3.6 – Environment Variables

Variable	Semantics	Possible Values
systemLanguage	Audio language	ISO 639-1 code
systemCaption	Caption language	ISO 639
systemSubtitle	Subtitle Language	ISO 639
systemReturnBitRate(i)	Bit rate of the return channel (i) in Kbps	real
systemScreenSize (i)	Screen size of the device (i) in (lines, pixels/line	(integer, integer)
systemScreenGraphicSize (i)	Resolution set for the screen graphics plane of the device (i) in (lines, pixels/line)	(integer, integer)
systemAudioType(i)	Type of the audio of the device (i)	“mono” | “stereo” | “5.1”
systemCPU	CPU performance in MIPS	real
systemMemory	Memory space in Mbytes	integer
systemOperatingSystem	Type of the Operating System	string to be defined
systemXXX	Any variable initiating by “system” must be reserved for future use	
userAge	User age	integer
userLocation	User location (postal code number)	integer-integer
userGenre	User genre	“m”| “f”
userXXX	Any variable initiating by “user” must be reserved for future use	
defaultFocusBorderColor	Default color applied to the border of an element in focus	“white” | “black” | “silver” | “gray” | “red” | “maroon” | “fuchsia” | “purple” | “lime” | “green” | “yellow” | “olive” | “blue” | “navy” | “aqua” | “teal”
defaultSelBorderColor	Default color applied to the border of an element in focus when activated	“white” | “black” | “silver” | “gray” | “red” | “maroon” | “fuchsia” | “purple” | “lime” | “green” | “yellow” | “olive” | “blue” | “navy” | “aqua” | “teal”
defaultFocusBorderWidth	Default width (in pixels) applied to the border of an element in focus	integer

Variable	Semantics	Possible Values
defaultFocusTransparency	Default transparency applied to the border of an element in focus	a real value between 0 and 1, with "0" meaning full transparency and "1" meaning no transparency

The *type* attribute is optional (except for <media> elements with no *src* attribute defined) and should be used to guide the player (presentation tool) choice by the formatter. When the *type* attribute is not specified, the formatter should use the content extension specification in the *src* attribute to make the player choice.[4]

The Context module is responsible for the definition of context nodes through <context> elements. 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. As usual, the *id* attribute uniquely identifies the <context> and <media> elements within a document.

The *newInstance*, *refer* and *descriptor* attributes are extensions defined in other modules and are discussed in the definition of those modules.

Tables 3.7 and 3.8 present the elements of these two modules, their child elements, and their attributes.

Table 3.7 – Extended Media Module in the Enhanced DTV profile

Elements	Attributes	Content
media	*id, src, refer, newInstance, type, descriptor*	*(area|attribute)**

Table 3.8 – Extended Context Module in the Enhanced DTV profile

Elements	Attributes	Content
context	*id, refer, descriptor*	*(port|attribute|media|context|link|switch)**

[4] When there is more than one player for the type supported by the formatter, the <descriptor> element may specify which one will be used for presentation. Otherwise the formatter must use a default player for that type of media.

3.2.3.1. XHTML Objects Embedded in NCL

As NCL has a stricter separation between content and structure, NCL does not define any media itself. Instead, it defines the glue that holds media together in multimedia presentations.

Thus, an NCL document 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. In this sense, we can have image objects (GIF, JPEG, etc.), video objects (MPEG, MOV, etc.), audio objects (MP3, WMA, etc.), text objects (TXT, PDF, etc.), execution objects (Xlet, LUA, etc.), etc., as NCL media objects. Which are the media objects supported depends on the media players that are embedded in the NCL formatter.

An NCL media object that should certainly be supported is the HTML-based media object. Therefore, NCL does not substitute but embed HTML-based documents (or objects). As with other media objects, what HTML-based language will have support in an NCL formatter is an implementation choice, and therefore it will depend on which HTML browser will act as a media player embedded in the NCL formatter.

As a consequence, it is possible to have BML browsers, DVB-HTML browsers and ACAP-X browsers embedded in an NCL document player, for example. It is even possible to have all of them.

Although an XHTML-based browser must be supported, the use of XHTML elements to define relationships (including XHTML links) should be dissuaded when authoring NCL documents. Structure-based authoring should be emphasized for the well-known reasons largely reported in the literature.

3.2.4. Interfaces Functionality

The Interfaces functionality allows the definition of node (media object or composite node) interfaces that will be used in relationships with other node interfaces. This functionality is partitioned into four modules:

1. MediaContentAnchor, which allows content anchor (or area) definitions for media nodes (<media> elements);
2. CompositeNodeInterface, which allows port definitions for composite nodes (<context> and <switch> elements);
3. AttributeAnchor, which allows the definition of node attributes or node attribute groups as node interfaces; and
4. SwitchInterface, which allows the definition of special interfaces for <switch> elements.

The MediaContentAnchor module defines the <area> element, which extends the syntax and semantics of the homonym element defined by SMIL [SMIL05] and XHTML [W3C02]. As such, it allows the definition of content anchors representing spatial portions, through the *coords* attribute (as in XHTML); the definition of content anchors representing temporal portions, through *begin* and *end* attributes; and the definition of content anchors representing temporal and spatial portions through *coords*, *begin* and *end* attributes (as in SMIL). In addition, the <area> element allows the definition of textual anchors, through the *text* and *position* attributes that define the string and the string's initial position in the text,

respectively. Besides, the <area> element can also define a content anchor based on the number of audio samples or video frames, through attributes *first* and *last*, which should indicate the initial and final sample/frame. Moreover, the <area> element can also define a content anchor based on an *anchorLabel*, which specifies a string that should be used by the media player to identify a content region.

As usual, <area> elements must have the *id* attribute, which uniquely identifies the element within a document.

Table 3.9 summarizes the <area> element and its attributes.

Table 3.9 – Extended MediaContentAnchor Module in the Enhanced DTV profile

Elements	Attributes	Content
area	*id, coords, begin, end, text, position, first, last, anchorLabel*	*empty*

The CompositeNodeInterface module defines the <port> element, which specifies a composite node port with its respective mapping to an interface (*interface* attribute) of one of its components (specified by the *component* attribute).

In NCM, every node (media or context node) must have an anchor with a region representing the whole content of the node. This anchor is called the *whole content anchor* and is declared by default in NCL. Every time an NCL component is referenced without specifying one of its anchors, the *whole content anchor* is assumed.

Table 3.10 summarizes the <port> element and its attributes.

Table 3.10– Extended CompositeNodeInterface Module in the Enhanced DTV profile

Elements	Attributes	Content
port	*id, component, interface*	*empty*

The AttributeAnchor module defines an element named <attribute>, which can be used for defining a node attribute or a group of node attributes as one of its interfaces (anchors)[5]. As usual, <attribute> elements must have the *id* attribute, which uniquely identifies the element within a document. The <attribute> element has also an attribute called *name*, which indicates the name of the attribute or attribute group, and *value*, an optional attribute that defines a value for the *name*.

[5] It is possible to have NCL document players (formatters) that define some node attributes as node interfaces, implicitly. However, in general, it is a good practice to explicitly define the interfaces.

The <body>, <context>, and <media> elements may have several embedded attributes. Examples of these attributes can be found among those that define the object placement during a presentation, the presentation duration, and others that define additional presentation characteristics: top, left, bottom, right, width, height, explicitDur, background, transparency, visible, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, bassLevel, etc. Whenever not specified in the <media>, <context> or <body> elements, these attributes assume as their initial values those defined in homonym attributes of their node-associated descriptor and region, as discussed in Sections 3.2.2 and 3.2.5. However, when any of these attributes is used in a relationship, it must be explicitly declared as an <attribute> (interface) element[6]. Moreover, a group of node attributes can also be explicitly declared as an <attribute> (interface) element. The following groups must be recognized by an NCL formatter: *location*, grouping (left, top) in this order; *size*, grouping (width, height) in this order; and *bounds*, grouping (left, top, width, height) in this order. When a formatter treats a change in an attribute group it must only test the process consistency at its end. Therefore top, left, bottom, right, width, height, explicitDur, background, transparency, visible, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, bassLevel, location, size and bounds are reserved words for values of the *name* attribute of the <attribute> element.

Table 3.11 summarizes the <attribute> element and its attributes.

Table 3.11 – Extended AttributeAnchor Module in the Enhanced DTV profile

Elements	Attributes	Content
attribute	*id, name, value*	*empty*

The SwitchInterface module allows the creation of <switch> element interfaces that will be mapped to a set of alternative interfaces of internal nodes, allowing a link to anchor on the component chosen when the <switch> is processed (see [SoRo05]). This module introduces the <switchPort> element, which contains a set of *mapping* elements. A *mapping* element defines a path from the <switchPort> to an interface (*interface* attribute) of one of the switch components (specified by its *component* attribute).

It is important to remark that every element representing an object interface (<area>, <port>, <attribute>, and <switchPort>) must have an identifier (*id* attribute).

Table 3.12 summarizes the <switchPort> element, its child elements, and its attributes.

[6] It must be stressed here the difference between attributes of an element (<context>, <switch>, and <media> elements) and the NCL <attribute> element. The NCL <attribute> element is used to declare only those element's attributes used in relationships. It must also be stressed that, although homonym, the NCL <attribute> is an XML element and not an XML attribute.

Table 3.12 – Extended SwitchInterface Module in the Enhanced DTV profile

Elements	Attributes	Content
switchPort	*id*	*mapping+*
mapping	*component, interface*	*empty*

3.2.5. Presentation Specification Functionality

The Presentation Specification functionality has a single module named Descriptor. The purpose of this module is to specify temporal and spatial information needed to present each document component. This information is modeled by *descriptor* objects, according to NCM [SoRo05].

The Descriptor module allows the definition of <descriptor> elements, which contain a set of optional attributes, grouping all temporal and spatial definitions, which should be used according to the type of object to be presented. The definition of <descriptor> elements must be included in the document head, inside the <descriptorBase> element, which specifies the set of descriptors of a document. The <descriptor> element must have the *id* attribute and the <descriptorBase> element may have the *id* attribute, which, as usual, uniquely identifies the elements within a document.

A <descriptor> element may have temporal attributes: *explicitDur* and *freeze,* defined by the Timing module (see Section 3.2.9); an attribute named *player*, which identifies the presentation tool to be used; an attribute named *region*, which refers to a region defined by elements of the Layout module (see Section 3.2.2), and key-navigation attributes: *moveLeft*, *moveRight*, *moveUp*; *moveDown*, *focusIndex*, *focusBorderColor*; *focusBorderWidth*; *focusBorderTransparency*, *focusSrc*, *selBorderColor*, and *focusSelSrc*, defined by the KeyNavigation module (see Section 3.2.11).

A <descriptor> element may also have <descriptorParam> child elements, which are used to parameterize the presentation control of the object associated with the descriptor element. These parameters can, for example, redefine some attribute values defined by the region attributes. They can also define new attributes such as *background*, specifying the background color used to fill the area of a region displaying media that is not filled by the media itself; *visible,* allowing the object presentation to be seen or hidden; *fit*, indicating how an object will be presented; *scroll*, which allows the specification of how an author would like to configure the scroll in a region; *transparency*, indicating the degree of transparency of an object presentation (it must have a value between 0 and 1); *style*, which refers to a style sheet [IETF98] with information for text presentation, for example; and also specific attributes for audio objects, such as *soundLevel*, *balanceLevel*, *trebleLevel* and *bassLevel*. Besides, <descriptorParam> child elements can determine if a new player must be instantiated or if a player already instantiated must be used (*reusePlayer*), and specify what will happen to the player instance at the end of the presentation (*playerLife*). Therefore top, left, bottom, right, width, height, explicitDur, location, size, bounds, background, visible, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, and bassLevel

are reserved words for values of the *name* attribute of the <descriptorParam> element. Table 3.13 summarizes the possible values for the reserved parameter/attribute names.

Table 3.13 – Reserved parameter/attribute and possible values

Parameter/attribute Name	Value
top, left, bottom, right, width, height	A real number in the range [0,100] ending with the character "%" (e.g. 30%), or a non-negative integer value specifying the attribute in pixels.
location	Two numbers separated by comma, each one one following the value rule specified for left and top parameters, respectively
size	Two values separated by comma. Each value must follow the same rule specified for width and height parameters, respectively
bounds	Four values separated by comma. Each value must follow the same rule specified for left, top, width and height parameters, respectively.
background	Reserved color names: "white", "black", "silver", "gray", red", "maroon", fuchsia", "purple", "lime", "green", "yellow", "olive", "blue", "navy", "aqua", or "teal". Other options to specify the color value may be adopted. The background value can also be the reserved value "transparent". This can be helpful to present transparent images, like transparent GIFs, superposed on other images or videos. When not specified, the background attribute will take the default value "transparent".
visible	"true" or "false".
transparency	A real number in the range [0,1] specifying the degree of transparency of an object presentation ("0" means full transparency and "1" means opaque.
fit	"fill", "hidden", "meet", "meetBest", "slice". "fill": scale the object's media content so that it touches all edges of the box defined by the object's width and height attributes. "hidden": if the intrinsic height (width) of the media content is smaller than the height (width) attribute, the object must be rendered starting from the top (left) edge and have the remaining height (width) filled up with the background color; if the intrinsic height (width) of the media content is greater than the height (width) attribute, the object must be rendered starting from the top (left) edge until the height (width) defined in the attribute is reached, and have the part of the media content below (right of) the height (width) clipped. "meet": scale the visual media object while preserving its aspect ratio until its height or width is equal to the value specified by the height or width attributes. The media content left-top corner is positioned at the top-left coordinates of the box; the empty space at the right or the bottom must be filled up with the background color. "meetBest": the semantic is identical to "meet" except that the image is not scaled greater than 100% in either dimension. "slice": scale the visual media content while preserving its aspect ratio until its height or width are equal to the value specified in the height and width attributes and the defined presentation box is completely filled. Some parts of the content may get clipped. Overflow width is clipped from the right of the media object. Overflow height is clipped from the bottom of the media object.
scroll	"none", "horizontal", "vertical", "both", or "automatic"
style	The locator of a stylesheet file.
soundLevel, balanceLevel, trebleLevel, bassLevel	A real number in the range [0, 1]

Besides all aforementioned attributes, the <descriptor> element can also have attributes defined in the SMIL Transition Effects Functionality, as specified in Section 3.2.12.

Besides the <descriptor> element, the Descriptor module defines a homonym attribute, which refers to an element of the document descriptor set. When a language profile uses the Descriptor module, it has to determine how *descriptors* will be associated with document components. Following NCM directives, NCL establishes that the *descriptor* attribute is associated with any media or context node through <media> and <context> elements or through link endpoint (<bind> element).

It should be remarked that the set of descriptors of a document may contain <descriptor> elements or <descriptorSwitch> elements, which allow specifying alternative descriptors, as it will be presented in Section 3.2.8.

Table 3.14 presents the elements of the Descriptor module, their child elements, and their attributes.

Table 3.14 – Extended Descriptor Module in the Enhanced DTV profile

Elements	Attributes	Content
descriptor	*id, player, explicitDur, region, freeze, moveLeft, moveRight, moveUp; moveDown, focusIndex, focusBorderColor; focusBorderWidth; focusBorderTransparency , focusSrc,focusSelSrc, selBorderColor, transIn, transOut*	*(descriptorParam)**
descriptorParam	*name, value*	
descriptorBase	*id*	*(importBase|descriptor|descriptorSwitch)+*

3.2.6. Linking Functionality

The Linking functionality defines the Linking module, responsible for defining links using connectors. A <link> element may have an *id* attribute, which uniquely identifies the element within a document, and must have an *xconnector* attribute, which refers to a hypermedia connector URI. The reference must have the format: *alias#connector_id*, or *documentURI_value#connector_id*, for connectors defined in an external document (as discussed in Section 3.2.10); or simply *connector_id*, for connectors defined in the document itself.

The <link> element also contains child elements called <bind> elements, which allow to associate nodes with connector roles, as defined in Section 3.2.7. In order to make this association, a <bind> element has four basic attributes. The first one is called *role*, which is used for referring to a connector role. The second one is called *component*, which is used for identifying the node. The third is an optional attribute called *interface*[7], used for making reference to the node interface. The fourth is an optional attribute called *descriptor* used to refer to a descriptor to be associated with the node, as defined by the Descriptor module (see Section 3.2.5).

If the connector element defines parameters (as discussed in Section 3.2.7), the <bind> and <link> elements should define parameter values, through child elements called <bindParam> and <linkParam>, respectively, both with *name* and *value* attributes. In this case the name attribute must refer to the name of a connector parameter while the value attribute must define a value to be assigned to the respective parameter.

Table 3.15 presents the elements of the Linking module, their attributes, and their child elements.

Table 3.15 - Extended Linking Module in the Enhanced DTV profile

Elements	Attributes	Content
bind	*role, component, interface, descriptor*	*(bindParam)**
bindParam	*name, value*	*empty*
linkParam	*name, value*	*empty*
link	*id, xconnector*	*(linkParam*, bind+)*

3.2.7. Connectors Functionality

The NCL 3.0 Connectors Functionality is partitioned into eight basic modules: ConnectorCommonPart, ConnectorAssessmentExpression, ConnectorCausalExpression, ConnectorTransitionAssessment, CausalConnector, ConstraintConnector (not considered in the profiles presented in this report), ConnectorBase, and CompositeConnector (not considered in this recomendation).

The Connectors-Functionality's modules are totally independent from the other NCL modules. These modules are the core by themselves of an XML application language (indeed other NCL 3.0 profiles) for the definition of connectors, which can be used to specify spatio-temporal synchronization relations, treating reference (user interaction) relations as a particular case of temporal synchronization relations.

[7] Note that the *interface* attribute can refer to any node interface, that is, an anchor, an attribute or a port, if it is a composite node. The interface attribute is optional. When it is not specified the association will be done with the whole node content, as explained in Section 3.2.4.

Besides the basic modules, the Connector Functionality also defines modules that group sets of basic modules, in order to make easy a language profile definition. This is the case of the CausalConnectorFunctionality module, used in the definition of the EDTV, BDTV and CausalConnector profiles. The CausalConnectorFunctionality module groups the following modules: ConnectorCommonPart, ConnectorAssessmentExpression, ConnectorCausalExpression, ConnectorTransitionAssessment, and CausalConnector.

A <causalConnector> element represents a relation that can be used for creating <link> elements in documents. In a causal relation, a condition must be satisfied in order to trigger an action.

A <causalConnector> specifies a relation independently of relationships, that is, it does not specify which nodes (represented by <media>, <context>, <body>, and <switch> elements) will interact through the relation. A <link> element, in its turn, represents a relationship, of the type defined by its connector, interconnecting different nodes. Links representing the same type of relation, but interconnecting different nodes, can reuse the same connector, reusing all previous specifications. A <causalConnector> specifies, through its child elements, a set of interface points, called *roles*. A <link> element refers to a <causalConnector> and defines a set of binds (<bind> child elements of the <link> element), which associate each link endpoint (node interface) to a role of the used connector.

Relations in NCL are based on events. An event is an occurrence in time that can be instantaneous or have a measurable duration. NCL 3.0 defines the following types of events:

- *presentation event*, which is defined by the presentation of a subset of the information units of a media object, specified in NCL by the <area> element, or by the media node itself (whole content presentation). Presentation events may also be defined on composite nodes (represented by a <body>, <context>, or <switch> element), representing the presentation of the information units of any node inside a composite node);
- *selection event*, which is defined by the selection of a subset of the information units of a media object, specified in NCL by the <area> element, or by the media node itself (whole content presentation);
- *attribution event*, which is defined by the attribution of a value to an attribute of a node (represented by the <media>, <body>, <context>, or <switch> element), which must be declared in a <attribute> child element of the node; and
- *composition event*, which is defined by the presentation of the structure of a composite node (represented by a <body>, <context>, or <switch> element). Composition events are used to present the composite map (composite organization).

Each event defines a state machine that must be maintained by the NCL formatter, as shown in Figure 3.2. Moreover, every event has an associate attribute, named *occurrences*, which counts how many times the event transits from occurring to sleeping state during a document presentation. Events like presentation and attribution have also an attribute named *repetitions*, which counts how many times the event must be automatically restarted (transited from sleeping to occurring states) by the formatter. This attribute can contain the "indefinite" value, leading to an endless loop of the event occurrences until some external interruption.

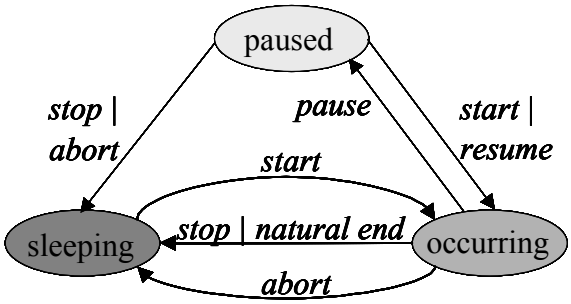

Figure 3.2 - Event state machine

Table 3.16 defines transition names for the event state machine.

Table 3.16 - Transition names for an event state machine

Transition (caused by action)	Transition Name
sleeping→occurring (start)	*starts*
occurring→sleeping (stop or natural end)	*stops*
occurring→sleeping (abort)	*aborts*
occurring→paused (pause)	*pauses*
paused→occurring (resume or start)	*resumes*
paused→sleeping (stop)	*stops*
paused→sleeping (abort)	*aborts*

A presentation event associated with a media node, represented by a <media> element, initializes in the sleeping state. At the beginning of the exhibition of its information units, the event goes to the occurring state. If the exhibition is temporarily suspended, the event stays in the paused state, while this situation lasts. A presentation event can change from occurring to sleeping as a consequence of the natural end of the presentation duration, or due to an action that stops the event. In both cases, the *occurrences* attribute is incremented, and the *repetitions* attribute is decremented by one. If after being decremented, the *repetitions* attribute value is greater than zero, the event is automatically restarted (set again to the occurring state). When the presentation of an event is abruptly interrupted, through an abort presentation command, the event also goes to the sleeping state, but without incrementing the *occurrences* attribute and setting the *repetitions* attribute value to zero. The duration of an event is the time it remains in the occurring state. This duration can be intrinsic to the media object, explicitly specified by an author (*explicitDur* attribute of a <descriptor> element), or derived from a relationship, as will be discussed.

A presentation event associated with a composite node represented by a <body> or a <context> element stays in the occurring state while at least one presentation event associated with one of the composite child nodes is in the occurring state, or at least one context node child link is being evaluated. It is in the paused state if at least one presentation event associated with one of the composite child nodes is in the paused state and all other presentation events associated with the composite nodes are in the sleeping or paused state. Otherwise, the presentation event is in the sleeping state. [8]

[8] More details about the behavior of presentation event state machines for media and composite nodes are given in Appendix B.

A presentation event associated with a switch node, represented by a <switch> element, stays in the occurring state while the switch child element chosen from the bind rules (selected node) is in the occurring state. It is in the paused state if the selected node is in the paused state. Otherwise, the presentation event is in the sleeping state.

A selection event initializes in the sleeping state. It stays in the occurring state while the corresponding anchor (subset of the information units of a media object) is being selected.

Attribute events stay in the occurring state while the corresponding attribute values are being modified. Obviously, instantaneous events, like attribute events for simple value assignments, stay in the occurring state only during an infinitesimal time.

A composition event (associated to a composite node represented by a <body>, <context> or <switch> element) stays in the occurring state while the composition map is being presented.

Relations are defined based on event states, changes on the event state machines, on event attribute values, and on node (<media>, <body>, <context> or <switch> element) attribute values. The CausalConnectorFunctionality module allows only the definition of causal relations, defined by the <causalConnector> element of the CausalConnector module.

A <causalConnector> element has a glue expression, which defines a condition expression and an action expression. When the condition expression is satisfied, the action expression must be executed. As usual, the <causalConnector> element must have the id attribute, which uniquely identifies the element within a document.

A condition expression may be simple (<simpleCondition> element) or composite (<compoundCondition> element), both elements defined by the ConnectorCausalExpression module.

The <simpleCondition> element has a *role* attribute, whose value must be unique in the connector's role set. As aforementioned, a role is a connector interface point, which is associated to node interfaces by a link that refers to the connector. A <simpleCondition> also defines an event type (*eventType* attribute) and to which transition it refers (*transition* attribute). The *eventType* and *transition* attributes are optional. They can be inferred by the *role* value if reserved values are used. Otherwise, the *eventType* and *transition* attributes are required. Reserved values used for defining <simpleCondition> roles are stated in Table 3.17. If an *eventType* value is "selection", the role may also define to which selection apparatus (for example, keyboard or remote control keys) it refers, through its *key* attribute. The role cardinality specifies the minimal (*min* attribute) and maximal (*max* attribute) number of participants that may play the role (number of binds) when the <causalConnector> is used for creating a <link>. When the maximal cardinality value is greater than one, several participants may play the same role, i.e., there may be several binds connecting diverse nodes to the same role. The "unbounded" value can be set to the *max* attribute, if the role can have unlimited binds associated with. In these two latter cases, a *qualifier* attribute should be specified informing the logical relationship among the simple condition binds. As described in Table 3.18, the possible values for the *qualifier* attribute are: "or" or "and". If the qualifier establishes an "or" logical operator, the link action will be fired whenever any condition occurs. If the qualifier establishes an "and" logical operator, the link action will be fired after all the simple conditions occur. If not specified, the default value "or" must be assumed.

Table 3.17 – Reserved condition *role* values associated to event state machines

Role Value	Transition Value	Event Type
onBegin	*starts*	*presentation*
onEnd	*stops*	*presentation*
onAbort	*aborts*	*presentation*
onPause	*pauses*	*presentation*
onResume	*resumes*	*presentation*
onSelection	*stops*	*selection*
onAttribution	*stops*	*attribution*

Table 3.18 – Simple condition *qualifier* values

Role Element	Qualifier	Semantics
simpleCondition	*or*	True whenever any associated simple condition occurs.
simpleCondition	*and*	True immediately after all associated simple conditions had occurred.

A *delay* attribute can also be defined for a <simpleCondition> specifying that the condition is true after a time delay from the time the transition occurs.

The <compoundCondition> element has a Boolean *operator* attribute ("and" [9] or "or") relating its child elements: <simpleCondition>, <compoundCondition>, <assessmentStatement> and <compoundStatement>. A *delay* attribute can also be defined specifying that the compound condition is true after a time delay the expression relating its child elements is true. The <assessmentStatement> and <compoundStatement> elements are defined by the ConnectorAssessmentExpression module, as discussed further on.

An action expression captures actions that can be executed in causal relations and may be composed by a <simpleAction> or a <compoundAction> element, also defined by the ConnectorCausalExpression module.

The <simpleAction> element has a *role* attribute, which has to be unique in the connector role set. As usual, the role is a connector interface point, which is associated to node interfaces by a <link> that refers to the connector. A <simpleAction> also defines an event type (*eventType* attribute) and which event state transition it triggers (*actionType*). The *eventType* and *actionType* attributes are optional. They can be inferred by the *role* value if reserved values are used. Otherwise, the *eventType* and *actionType* are required. Reserved values used for defining <simpleAction> *roles* are stated in Table 3.19. If an *eventType* value is "attribution", the <simpleAction> must also define the value that must be assigned, through its *value* attribute. As with <simpleCondition> elements the role cardinality specifies the minimal (*min* attribute) and maximal (*max* attribute) number of participants that may play the role (number of binds) when the <causalConnector> is used for creating a link. When the maximal cardinality value is greater than one, several participants may play the same role. When it has the "unbounded" value, the number of binds is unlimited. In these two later cases, a qualifier must be specified. Table 3.20 presents possible qualifier values.

[9] When an "and" compound condition relates more than one trigger condition (i.e. a condition that is satisfied only in an infinitesimal time instant – as for example, the end of an object presentation), the compound condition must be considered true in the instant immediately after all the trigger conditions are satisfied.

Table 3.19 – Reserved action *role* values associated to event state machines

Role Value	Action Type	Event Type
start	*start*	*presentation*
stop	*stop*	*presentation*
abort	*abort*	*presentation*
pause	*pause*	*presentation*
resume	*resume*	*presentation*
set	*set*	*attribution*

Table 3.20 – Action *qualifier* values

Role Element	Qualifier	Semantics
simpleAction	*par*	All actions must be executed in parallel
simpleAction	*seq*	All actions must be executed in the bind sequence

A *delay* attribute can also be defined for a <simpleAction> specifying that the action must be fired only after waiting for the specified time. Besides, the <simpleAction> can also define a *repeat* attribute to be assigned to the *repetitions* attribute of the event, and a *repeatDelay* to be waited before repeating the action.

The <compoundAction> element has an *operator* attribute ("par" or "seq") relating its child elements: <simpleAction> and <compoundAction>. Parallel ("par") and sequential ("seq") compound actions specify that the execution of actions must be performed in any order or in a specific order[10], respectively. A *delay* attribute can also be defined specifying that the compound action must be applied after the specified delay.

The ConnectorAssessmentExpression module defines four elements: <assessmentStatement>, <attributeAssessment>, <valueAssessment> and <compoundStatement>.

The <attributeAssessment> has a *role* attribute, which has to be unique in the connector role set. As usual, the *role* is a connector interface point, which is associated to node interfaces by a <link> that refers to the connector. An < attributeAssessment > also defines an event type (*eventType* attribute). If the *eventType* value is "selection", the <attributeAssessment> should also define to which selection apparatus (for example, keyboard or remote control keys) it refers, through its *key* attribute. If the *eventType* value is "presentation", the *attributeType* attribute specifies the event attribute ("occurrences" or "repetition") or the event state ("state"); if the *eventType* value is "selection", the *attributeType* attribute is optional and, if present, may have the value "occurrences" (default) or "state"; if the *eventType* is "attribution" the *attributeType* is optional and may have the value "nodeAttribute" (default), "occurrences", "repetition" or "state". In the first case, the event represents a node attribute to be evaluated, in the other ones the event represents the evaluation of the corresponding attribution event attribute or the attribution event state. An *offset* value can be added to an <attributeAssessment> before the comparison. For example, an offset may be added to an attribute assessment to specify: "the screen vertical position plus 50 pixels".

[10] It is important to mention that when the sequential operator is used, actions should be fired in the specified order. However, an action does not need to wait the previous one to be finished in order to be fired.

The <valueAssessment> element has a *value* attribute that should assume an event state value, or any value to be compared with a node or event attribute.

The <assessmentStatement> element has a *comparator* attribute that compares the values inferred from the <assessmentStatement> child elements.
a) In the case of <attributeAssessment>: a node attribute value [*eventType* = "attribution" and the *attributeType* = "nodeAttribute"]; or an event attribute value [*eventType* = ("presentation", "attribution" or "selection") and the *attributeType* = ("occurrences", or "repetition")]; or an event state [*eventType* = ("presentation", "attribution" or "selection") and the *attributeType* = "state"].
b) In the case of <valueAssessment>: a value of its *value* attribute.

The <compoundStatement> element has a Boolean *operator* attribute ("and" or "or") relating its child elements: <assessmentStatement> and <compoundStatement>. An *isNegated* attribute can also be defined to specify that the <compoundStatement> child element must be negated before the Boolean operation is evaluated.

The <causalConnector> element may have <connectorParam> child elements, which are used to parameterize connector attribute values. The ConnectorCommonPart module defines the type of the <connectorParam> element, which has *name* and *type* attributes. In order to specify which attributes receive parameter values defined by the connector, their values are specified as the parameter name preceded by the *$* symbol. For instance, in order to parameterize the *delay* attribute, a parameter called *actionDelay* is defined (<*connectorParam name="actionDelay" type="unsignedLong"/*>) and the value *"$actionDelay"* is used in the attribute (*delay*="$*actionDelay*").

Table 3.21 presents the elements of the CausalConnectorFunctionality module, their attributes, and their child elements.

Table 3.21 – Extended CausalConnectorFunctionality module in the Enhanced DTV profile

Elements	Attributes	Content
causalConnector	*id*	*(connectorParam*, (simpleCondition | compoundCondition), (simpleAction | compoundAction))*
connectorParam	*name, type*	*empty*
simpleCondition	*role, delay, eventType, key, transition, min, max, qualifier*	*empty*
compoundCondition	*operator, delay*	*((simpleCondition | compoundCondition)+, (assessmentStatement | compoundStatement)*)*

simpleAction	*role, delay, eventType, actionType, value, min, max, qualifier, repeat, repeatDelay*	*empty*
compoundAction	*operator, delay*	*(simpleAction | compoundAction)+*
assessmentStatement	*comparator*	*(attributeAssessment, (attributeAssessment | valueAssessment))*
attributeAssessment	*role, eventType, key, attributeType, offset*	*empty*
valueAssessment	*value*	*empty*
compoundStatement	*operator, isNegated*	*(assessmentStatement | compoundStatement)+*

The ConnectorBase module defines an element named <connectorBase>, which allows grouping connectors. As usual, the <connectorBase> element should have the *id* attribute, which uniquely identifies the element within a document. The exact content of a connector base is specified by the language profile that uses the Connectors Facility. However, since the definition of connectors is not easily done by naïve users, the idea is to have expert users defining connectors, storing them in libraries (*connectorBases*) that can be imported, and making them available to others for creating links. Appendix E gives an extensive example of connector definitions that can be imported.

Table 3.22 presents the element of the ConnectorBase module, its attributes, and its child elements.

Table 3.22 – Extended ConnectorBase module in the Enhanced DTV profile

Elements	Attributes	Content
connectorBase	*id*	*(importBase|causalConnector)**

3.2.8. Presentation Control Functionality

The purpose of the Presentation Control functionality is to specify content and presentation alternatives for a document. This functional area is partitioned into four modules, named TestRule, TestRuleUse, ContentControl and DescriptorControl.

The TestRule module allows the definition of rules that, when satisfied, select alternatives for document presentation. The specification of rules in NCL 3.0 was done in a separate module, because they will be useful for defining either alternative components or alternative descriptors.

The <ruleBase> element specifies a set of rules, and must be defined as a child element of the <head> element. These rules can be simple, defined by the <rule> element, or composite, defined by the <compositeRule> element. Simple rules define an identifier (*id* attribute), a variable (*var* attribute), a value (*value* attribute), and a comparator (*comparator* attribute) relating the variable to the value. Composite rules have an identifier (*id* attribute) and a Boolean operator ("and" or "or" – *operator* attribute) relating their child rules. As usual, the *id* attribute uniquely identifies the <rule> elements within a document.

Table 3.23 summarizes the elements of the TestRule module, their attributes, and their child elements.

Table 3.23 – Extended TestRule Module in the Enhanced DTV profile

Elements	Attributes	Content
ruleBase	*id*	*(importBase|rule|compositeRule)+*
rule	*id, var, comparator, value*	*empty*
compositeRule	*id, operator*	*(rule | compositeRule)+*

The TestRuleUse defines the <bindRule> element, which is used to associate rules with components of a <switch> or <descriptorSwitch> element, through its *rule* and *constituent* attributes, respectively.

Table 3.24 summarizes the element of the TestRuleUse module and its attributes.

Table 3.24 – Extended TestRule Module in the Enhanced DTV profile

Elements	Attributes	Content
bindRule	*constituent, rule*	*empty*

The ContentControl module specifies the <switch> element, allowing the definition of alternative document nodes to be chosen during presentation time. It should be noted that the NCL <switch> element does not have the same content as the other language homonyms (as for example the switch element defined by SMIL 2.0 BasicContentControl

module [SMIL05]), because it can contain, besides <media> and <switch> elements, <context> elements defined by the Context module of NCL 3.0. Test rules used to choose the switch component to be presented are defined by the TestRule module, as previously mentioned, or are test rules specifically defined and embedded in a NCL formatter implementation. The ContentControl module also defines the <defaultComponent> element, whose *component* attribute (also of IDREF type) identifies the default element that should be selected if none of the bindRule rules is evaluated as true.

In order to allow links to anchor on the component chosen after evaluating the rules of a *switch*, a language profile should also include the SwitchInterface module, which allows the definition of special interfaces, named <switchPort>.

As usual, <switch> elements must have the *id* attribute, which uniquely identifies the element within a document. The *refer* attribute is an extension defined in the Reuse module and is discussed in Section 3.2.10.

When a <context> is defined as a child of a <switch> element, the <link> elements recursively contained in the <context> must be considered by an NCL player only if the <context> is selected after the switch evaluation. Otherwise, the <link> elements should be considered disabled and must not interfere in the document presentation.

Table 3.25 summarizes the ContentControl module elements, their attributes and their child elements.

Table 3.25 – Extended ContentControl Module in the Enhanced DTV profile

Elements	Attributes	Content
switch	*id, refer*	*defaultComponent?, (switchPort | bindRule | media | context | switch)*)*
defaultComponent	*component*	*empty*

The DescriptorControl module specifies the <descriptorSwitch> element, which contains a set of alternative descriptors to be associated with an object. The <descriptorSwitch> elements must have the *id* attribute, which uniquely identifies the element within a document. Analogous to the <switch> element, the <descriptorSwitch> choice is done during presentation time, using test rules defined by the TestRule module, or test rules specifically defined and embedded in a NCL formatter implementation. The DescriptorControl module also defines the <defaultDescriptor> element, whose *descriptor* attribute (also of IDREF type) identifies the default element that should be selected if none of the bindRule rules is evaluated as true.

Table 3.26 summarizes the DescriptorControl module elements, their attributes, and their child elements.

Table 3.26 – Extended DescriptorControl Module

Elements	Attributes	Content
descriptorSwitch	*id*	*(defaultDescriptor?, (bindRule | descriptor)*)*
defaultDescriptor	*descriptor*	*empty*

During a document presentation, from the moment on a <switch> is evaluated, it is considered resolved until the end of the current switch presentation, that is, while its corresponding presentation event is in the "occurring" or "paused" state. During a document presentation, from the moment on a <descriptorSwitch> is evaluated, it is considered resolved until the end of the presentation of the <media> element that was associated to it, that is, while any presentation event associated with the <media> element is in the "occurring" or "paused" state.[11]

3.2.9. Timing Functionality

The Timing functionality defines the Timing module. The Timing module allows the definition of temporal properties for document components. Basically, this module defines attributes for specifying what will happen with an object at the end of its presentation (*freeze*), and the ideal duration of an object (*explicitDur*). These attributes can be incorporated by <descriptor> elements.

3.2.10. Reuse Functionality

NCL allows intensive reuse of its elements. The NCL Reuse functionality is partitioned into three modules: Import, EntityReuse and ExtendedEntityReuse.

In order to allow an entity base to incorporate another already-defined base, the Import module defines the <importBase> element, which has two attributes: *documentURI* and *alias*. The *documentURI* refers to a URI corresponding to the NCL document containing the base to be imported. The *alias* attribute specifies a name to be used as prefix when referring elements of this imported base. The alias name must be unique in a document and its scope is constrained to the document that has defined the alias attribute. The reference would have the format: *alias#element_id*, or *documentURI_value#element_id*. The import operation has the transitive property, that is, if *baseA* imports *baseB* that imports *baseC*, then *baseA* imports *baseC*. However, the *alias* defined for *baseC* inside *baseB* must not be considered by *baseA*.

When a language profile uses the Import module, the following specifications are allowed:

[11] NCL formatters should delay the switch evaluation to the moment that a link anchoring in the switch needs to be evaluated. The descriptorSwitch evaluation should be delayed until the object referring the descriptorSwitch needs to be prepared to be presented.

- The <descriptorBase> element may have a child <importBase> element referring to a URI corresponding to another NCL document containing the descriptor base (in fact its child elements) to be imported and nested. When a descriptor base is imported, the region base and the rule base, when present in the imported document, are also automatically imported to the corresponding region and rule bases of the importing document.

- The <connectorBase> element may have a child <importBase> element referring to a URI corresponding to another connector base (in fact its child elements) to be imported and nested.

- The <transitionBase> element may have a child <importBase> element referring to a URI corresponding to another transition base (in fact its child elements) to be imported and nested.

- The <ruleBase> element may have a child <importBase> element referring to a URI corresponding to another NCL document containing the rule base (in fact its child elements) to be imported and nested.

- The <regionBase> element may have a child <importBase> element referring to a URI corresponding to another NCL document containing the region base (in fact its child elements) to be imported and nested. Although NCL defines its layout model, nothing prevents an NCL document from using other layout models, since they define regions where objects can be presented, as for example SMIL 2.0 [SMIL05] layout models. On importing a <regionBase> an optional attribute, named *region,* may be specified within the <importBase> element. When present, the attribute must identify the *id* of a <region> element declared in the <regionBase> element of the host document (the document that did the importing operation). As a consequence, all child <region> elements of the imported <regionBase> must be considered as child <region> elements of the region referred by the <importBase>'s *region* attribute. If not specified, the child <region> elements of the imported <regionBase> must be considered children of the host document <regionBase> element.

The <importedDocumentBase> element specifies a set of imported NCL documents, and must be defined as a child element of the <head> element. As usual, <importedDocumentBase> elements must have the *id* attribute, which uniquely identifies the element within a document.

An NCL document can be imported through the <importNCL> element. All bases defined inside an NCL document, as well as the document body element, are imported all at once through the <importNCL> element. The bases will be treated as if each one is imported by an <importBase> element. The imported <body> element will be treated as a <context> element. The imported <body>, as well as any of its contained nodes, can be reused inside the <body> element of the importing NCL document. It should be remarked that the imported elements (the <body> and its nested child node elements) can only be reused: new relationships can be defined among new nodes created by reuse (see next paragraph), but no new relationships can be defined inside imported context nodes[12].

[12] In other words, when reused, a node cannot have its content modified, but can be related to other nodes.

The <importNCL> element has two attributes: *documentURI*, and *alias*. The *documentURI* refers to a URI corresponding to the document to be imported. The *alias* attribute specifies a name to be used when referring an element of this imported document. As in the <importBase> element, the name must be unique (type=ID) and its scope is constrained to the document that has defined the alias attribute. The reference would have the format: *alias#element_id*, or *documentURI_value#element_id*. It is important to note that the same alias should be used when referring to elements defined in the imported document bases (<regionBase>, <connectorBase>, <descriptorBase>, etc.). The <importNCL> element operation has also the transitive property, that is, if *documentA* imports *documentB* that imports *documentC*, then *documentA* imports *documentC*. However, the *alias* defined for *documentC* inside *documentB* must not be considered by *documentA*.

Table 3.27 presents the elements of the Import module, their child elements, and their attributes.

Table 3.27 – Extended Import Module in the Enhanced DTV profile

Elements	Attributes	Content
importBase	*alias, documentURI, region*	*empty*
importedDocumentBase	*id*	*(importNCL)+*
importNCL	*alias, documentURI*	*empty*

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> can be reused. In this case, the referred element and the element that refers to it must be considered the same. In other words it means that an NCM node can be represented by more than one NCL element. As nodes contained in an NCM composite node defines a set, an NCM node can be represented by no more than one NCL element inside a composition[13]. This means that the *id* attribute of an NCL element representing an NCM node is not only a unique identifier for the element, but also the unique identifier for the NCM node in the composition.

When a language profile uses this module, it may add the *refer* attribute to:

- A <media> or <switch> element. In this case, the referred element must be, respectively, a <media> or <switch> element, which will represent the same node previously defined in

[13] As an example, aiming at clarifying the concept, assume the NCL element (*node1*) that defines an NCM node. The NCL elements that refer to it (*node1ReuseA, node1ReuseB*) represent the same NCM node. In other words, the single NCM node is represented by more than one NCL element (*node1, node1ReuseA,* and *node1ReuseB*). Moreover, since nodes contained in an NCM composite node define a set, the NCL elements *node1, node1ReuseA,* and *node1ReuseB* must each be declared inside a different composition.

the document <body> itself or in an external imported <body>. This referred element must directly contain the definition of all its attributes and child elements.

- A <context> element. In this case, the referred element must be a <context> or a <body> element that will represent the same context, which is previously defined in the document <body> itself or in an external imported <body>. This referred element must directly contain the definition of all its attributes and child elements.

When an element declares a *refer* attribute, all of its other attributes (except the *id*) and child elements must be ignored by the formatter. The only exception is for <media> elements, in which new child <area> elements can be added, and a new Boolean attribute, *newInstance*, can be defined. The *newInstance* attribute is defined in the ExtendedEntityReuse module and has "true" as its default value. If *newInstance* receives a "false" value, the following semantics must be respected. Assume the set of <media> elements composed by the referred <media> element and all the referring <media> elements. If one of these <media> elements is being presented using a certain <descriptor> and if another of these elements is scheduled to be presented with the same <descriptor> and has the *newInstance* attribute as "false", no new presentation instance will be created. The instance in presentation must, from that moment on, notify all events associated with the <area> elements of both <media> elements. In all other cases, a new presentation instance, completely independent is created.

3.2.11.Navigational Key Functionality

The Navigational Key Functionality defines the KeyNavigation module that provides the extensions necessary to describe focus movement operations using a control device like a remote control. Basically, the module defines attributes that can be incorporated by <descriptor> elements.

The *focusIndex* attribute specifies an index for the <media> element to which the focus can be applied, when this element is in exhibition using the <descriptor> element that defined the attribute. When a <descriptor> element does not define this attribute, it is considered as if no focus could be set. In a certain presentation moment, if the focus has not been already defined, or is lost, a focus will be initially applied to the element that is being presented whose descriptor has the smallest index value. Values of *focusIndex* attribute must be unique in a NCL document. Otherwise, the attributes will be ignored.

The *moveUp* attribute specifies a value equal to the *focusIndex* value associated to an element to which the focus must be applied when the "up arrow key" is pressed. The *moveDown* attribute specifies a value equal to the *focusIndex* value associated to an element to which the focus must be applied when the "down arrow key" is pressed. The *moveRight* attribute specifies a value equal to the *focusIndex* value associated to an element to which the focus must be applied when the "right arrow key" is pressed. The *moveLeft* attribute specifies a value equal to the *focusIndex* value associated to an element to which the focus must be applied when the "left arrow key" is pressed. In order to allow import operations, the reference to destination focus must be of type *alias#focusIndex, documentURI_value#focusIndex*, or *#focusIndex*, the latter in the case of referring an element of the same document. The *alias* must be defined in the Import module, as specified in Section 3.2.10.

When the focus is applied to an element with the visibility property set to hidden, the current focus does not move.

The *focusSrc* attribute may specify an alternative media source to be presented, instead of the current presentation, if an element receives the focus. This attribute follows the same rules of the *src* attribute of the <media> element.

When an element receives a focus, the square box defined by the element positioning attributes must be decorated. The *focusBorderColor* attribute defines the decorative color and can receive the reserved color names: “white”, “black”, “silver”, “gray”, “red”, “maroon”, “fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, or “teal”. The *focusBorderWidth* attribute defines the width in pixels of the decorative border (0 means that no border will appear, positive values means that the border is outside the object content, and negative values means that the border is drawn over the object content), and the *focusBorderTransparency* attribute defines the decorative color transparency. The *focusBorderTransparency* must be a real value between 0 and 1 with “0” meaning full transparency and “1” meaning no transparency. When the *focusBorderColor,* the *focusBorderWidth,* or the *focusBorderTransparency* are not defined, default values must be assumed. These values are specified in attributes of the <media> element of type application/x-NCL-settings: *defaultFocusBorderColor, defaultFocusBorderWidth, defaultFocusTransparency*, respectively

When an element on focus is selected by pressing the activate (select or enter) key, the *focusSelSrc* attribute may specify an alternative media source to be presented, instead of the current presentation. This attribute follows the same rules of the *src* attribute of the <media> elements. When selected, the square box defined by the element positioning attributes must be decorated with the color defined by the *selBorderColor* attribute (default specified by the *defaultFocusBorderColor* of the <media> element of type application/x-NCL-settings), the width of the decorative border defined by the *focusBorderWidth* attribute, and the decorative color transparency defined by the *focusBorderTransparency* attribute.

When an element on focus is selected by pressing the “activate (select or enter) key”, the focus control must be passed to the <media> element renderer (player). The player can then follow its own rules for navigation. The focus control must be passed back to the NCL formatter when the “back key” is pressed. In this case, the activated element must be decorated as when it was on focus, before being activated.

3.2.12. The SMIL Transition Effects functionality is divided into three modules: TransitionBase SMIL Transition Effects Functionality

The SMIL Transition Effects functionality is divided into three modules: TransitionBase, coming from NCL specifications, BasicTransitions and TransitionModifiers.

The TransitionBase module is defined by NCL 3.0 and consists on the <transitionBase> element that specifies a set of transition effects, and must be defined as a child element of the <head> element.

Table 3.28 presents the <transitionBase> element, its child elements, and its attributes.

Table 3.28 – Extended TransitionBase Module in the Enhanced DTV profile

Elements	Attributes	Content
transitionBase	*id*	*(importBase, transition)+*

The BasicTransitions module is imported from SMIL 2.0 specification [SMIL05]. It has just one element called <transition> element.

In NCL 3.0 Enhanced DTV profile, the <transition> element is specified in the <transitionBase> element and allows a transition template to be defined. Each <transition> element defines a single transition template and must have an *id* attribute so that it can be referenced inside a <descriptor> element.

The <transition> element attributes are: *type*; *subtype*; *dur*; *startProgress*; *endProgress*; *direction*; and *fadeColor*.

Transitions are classified according to a two-level taxonomy of types and subtypes. Each of the transition types describe a group of transitions which are closely related. Within that type, each of the individual transitions is assigned a subtype which emphasizes the distinguishing characteristic of that transition.

The *type* attribute is required and is used to specify the general transition. If the named type is not supported by the NCL formatter, the transition is ignored. Note that this is not an error condition, since implementations are free to ignore transitions.

The *subtype* attribute provides transition-specific control. This attribute is optional and, if specified, must be one of the transition subtypes appropriate for the specified type. If this attribute is not specified then the transition reverts to the default subtype for the specified transition type. Only the default subtype for the five required transition types listed in Table 3.29 must be supported.

Table 3.29 – Required transition types and subtypes.

Transition type	**Default Transition subtype**
barWipe	leftToRight
irisWipe	rectangle
clockWipe	clockwiseTwelve
snakeWipe	topLeftHorizontal
fade	crossfade

The *dur* attribute specifies the duration of the transition. The default duration is 1 second.

The *startProgress* attribute specifies the amount of progress through the transition at which to begin execution. Legal values are real numbers in the range [0.0,1.0]. For instance, we

may want to begin a crossfade with the destination image being already 50% faded in. For this case, *startProgress* would be 0.5. The default value is 0.0.

The *endProgress* attribute specifies the amount of progress through the transition at which to end execution. Legal values are real numbers in the range [0.0,1.0], and the value of this attribute must be greater than or equal to the value of the *startProgress* attribute. If *endProgress* is equal to *startProgress*, then the transition remains at a fixed progress for the duration of the transition. The default value is 1.0.

The *direction* attribute specifies the direction the transition will run. The legal values are "forward" and "reverse". The default value is "forward". Note that not all transitions will have meaningful reverse interpretations. For instance, a crossfade is not a geometric transition, and therefore has no interpretation of reverse direction. Transitions that do not have a reverse interpretation should have the direction attribute ignored and the default value of "forward" assumed.

If the value of the *type* attribute is "fade" and the value of the *subtype* attribute is "fadeToColor" or "fadeFromColor", then the *fadeColor* attribute specifies the starting or ending color of the fade. If the value of the *type* attribute is not "fade", or the value of the *subtype* attribute is not "fadeToColor" or "fadeFromColor", then the *fadeColor* attribute must be ignored. The default value is "black".

The BasicTransition module also defines attributes to be used in <descriptor> elements to use the transition templates defined by <transition> elements: *transIn* and *transOut* attributes. Transitions specified with a transIn attribute will begin at the beginning of the media element's active duration (when the object presentation begins to occur). Transitions specified with a transOut attribute will end at the end of the media element's active duration (when the object presentation transits from occurring to sleeping state).

The *transIn* and *transOut* attributes are added to <descriptor> elements. The default value of both attributes is an empty string, which indicates that no transition must be performed.

The value of these attributes is a semicolon-separated list of transition identifiers. Each of the identifiers must correspond to the value of the XML identifier of one of the transition elements previously defined in the <transitionBase> element. The purpose of the semicolon-separated list is to allow authors to specify a set of fallback transitions if the preferred transition is not available. The first transition in the list should be performed if the user-agent has implemented this transition. If this transition is not available, then the second transition in the list should be performed, and so on. If the value of the *transIn* or *transOut* attribute does not correspond to the value of the XML identifier of any one of the transition elements previously defined, then this is an error. In the case of this error, the value of the attribute should be considered to be the empty string and therefore no transition should be performed.

All the transitions defined in the SMIL BasicTransitions module accept four additional attributes that can be used to control the visual appearance of the transitions, as specified by the TransitionModifiers module. The *horRepeat* attribute specifies how many times to perform the transition pattern along the horizontal axis. The default value is 1 (the pattern occurs once horizontally). The *vertRepeat* attribute specifies how many times to perform the transition pattern along the vertical axis. The default value is 1 (the pattern occurs once vertically). The *borderWidth* attribute specifies the width of a generated border along a

wipe edge. Legal values are integers greater than or equal to 0. If *borderWidth* value is equal to 0, then no border should be generated along the wipe edge. The default value is 0. If the value of the *type* attribute is not “fade”, then the *borderColor* attribute specifies the content of the generated border along a wipe edge. If the value of this attribute is a color, then the generated border along the wipe or warp edge is filled with this color. If the value of this attribute is "blend", then the generated border along the wipe blend is an additive blend (or blur) of the media sources. The default value for this attribute is "black".

Table 3.30 presents the element of the Extended BasicTransition Module, its child elements, and its attributes.

Table 3.30 – Extended BasicTransition module in the Enhanced DTV profile

Elements	Attributes	Content
transition	*id, type, subtype, dur, startProgress, endProgress, direction, fadeColor, horRepeat, vertRepeat, borderWidth, borderColor*	*empty*

3.2.13.SMIL Meta-Information Functionality

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 SMIL Meta-Information Functionality is composed by the Metainformation module.

The Metainformation module contains two elements that allow description of NCL documents. The <meta> element specifies a single property/value pair in the *name* and *content* attributes, respectively. The <metadata> element contains information that is also related to meta-information of the document. It acts as the root element of the RDF tree. The <metadata> element can have as child elements: RDF element and its sub-elements (refer to W3C metadata Recommendations [RDF99]).

Table 3.31 presents the elements of the Metainformation module, their child elements, and their attributes.

Table 3.31 – Extended Metainformation module in the Enhanced DTV profile

Elements	Attributes	Content
meta	*name, content*	*empty*
metadata	*empty*	*RDF tree*

3.3. Attributes and Elements of the NCL 3.0 BasicDTV Profile

This section briefly describes the main definitions made by each NCL 3.0 module that are present in the NCL 3.0 Basic DTV profile. The elements, and their attributes, used in this profile are shown in the tables. Note that attributes and contents (child elements) of elements can be defined in the module itself or in the NCL Basic DTV profile that groups the modules. Element attributes that are required are underlined. As usual, in the tables, the following symbols are used: (?) optional (zero or one occurrence), (|) or, (*) zero or more occurrences, (+) one or more occurrences.

Table 3.32 – Extended Structure Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
ncl	*id, title, xmlns*	*(head?, body?)*
head		*(importedDocumentBase? ruleBase?, regionBase*, descriptorBase?, connectorBase?),*
body	*id*	*(port| attribute| media|context|switch|link)**

Table 3.33 - Extended Layout Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
regionBase	*id, device*	*(importBase|region)+*
region	*id, title, left, right, top, bottom, height, width, zIndex*	*(region)**

Table 3.34 – Extended Media Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
media	*id, src, refer, newInstance, type, descriptor*	*(area|attribute)**

Table 3.35 – Extended Context Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
context	*id, refer, descriptor*	*(port|attribute|media|context|link|switch)**

Table 3.36 – Extended MediaContentAnchor Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
area	*id, coords, begin, end, text, position, first, last, anchorLabel*	*empty*

Table 3.37 – Extended CompositeNodeInterface Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
port	*id, component, interface*	*empty*

Table 3.38 – Extended AttributeAnchor Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
attribute	*id, name, value*	*empty*

Table 3.39 – Extended SwitchInterface Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
switchPort	*id*	*mapping+*
mapping	*component, interface*	*empty*

Table 3.40 – Extended Descriptor Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
descriptor	*id, player, explicitDur, region, freeze, moveLeft, moveRight, moveUp; moveDown, focusIndex, focusBorderColor; focusBorderWidth; focusBorderTransparency, focusSrc,focusSelSrc, selBorderColor*	*(descriptorParam)**
descriptorParam	*name, value*	
descriptorBase	*id*	*(importBase | descriptor | descriptorSwitch)+*

Table 3.41 - Extended Linking Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
bind	*role, component, interface, descriptor*	*(bindParam)**
bindParam	*name, value*	*empty*
linkParam	*name, value*	*empty*
link	*id, xconnector*	*(linkParam*, bind+)*

Table 3.42 – Extended CausalConnectorFunctionality module elements and attributes in the Basic DTV profile

Elements	Attributes	Content
causalConnector	*id*	*(connectorParam*, (simpleCondition | compoundCondition), (simpleAction | compoundAction))*
connectorParam	*name, type*	*empty*
simpleCondition	*role, delay, eventType, key, transition, min,*	*empty*

	max, qualifier	
compoundCondition	*operator, delay*	*((simpleCondition | compoundCondition)+, (assessmentStatement | compoundStatement)*)*
simpleAction	*role, delay, eventType, actionType, value, min, max, qualifier, repeat, repeatDelay*	*empty*
compoundAction	*operator, delay*	*(simpleAction | compoundAction)+*
assessmentStatement	*comparator*	*(attributeAssessment, (attributeAssessment | valueAssessment))*
attributeAssessment	*role, eventType, key, attributeType, offset*	*empty*
valueAssessment	*value*	*empty*
compoundStatement	*operator, isNegated*	*(assessmentStatement | compoundStatement)+*

Table 3.43 – Extended ConnectorBase module element and attributes used in the Basic DTV profile

Elements	Attributes	Content
connectorBase	*id*	*(importBase|causalConnector)**

Table 3.44 – Extended TestRule Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
ruleBase	*id*	*(importBase|rule|compositeRule)+*
rule	*id, var, comparator, value*	*empty*
compositeRule	*id, operator*	*(rule | compositeRule)+*

Table 3.45 – Extended TestRuleUse Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
bindRule	*constituent, rule*	*empty*

Table 3.46 – Extended ContentControl Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
switch	*id, refer*	*(defaultComponent?,(switchPort| bindRule|media| context | switch)*)*
defaultComponent	*component*	*empty*

Table 3.47 – Extended DescriptorControl Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
descriptorSwitch	*id*	*(defaultDescriptor?, (bindRule | descriptor)*)*
defaultDescriptor	*descriptor*	*empty*

Table 3.48 – Extended Import Module elements and attributes used in the Basic DTV profile

Elements	Attributes	Content
importBase	*alias, documentURI, region*	*empty*
importedDocumentBase	*id*	*(importNCL)+*
importNCL	*alias, documentURI,*	*empty*

4. NCL 3.0 Module Schemas

This section presents all NCL 3.0 module schemas used in the Basic DTV and the Enhanced DTV profiles.

Structure Module: NCL30Structure.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Structure.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the Structure module namespace,
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Structure"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <!-- ================================================================= -->
  <!-- define the top-down structure of an NCL language document.        -->
  <!-- ================================================================= -->

  <complexType name="nclPrototype">
    <sequence>
      <element ref="structure:head" minOccurs="0" maxOccurs="1"/>
      <element ref="structure:body" minOccurs="0" maxOccurs="1"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="title" type="string" use="optional"/>
  </complexType>

  <complexType name="headPrototype">
  </complexType>

  <complexType name="bodyPrototype">
    <attribute name="id" type="ID" use="optional"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="ncl" type="structure:nclPrototype"/>
  <element name="head" type="structure:headPrototype"/>
  <element name="body" type="structure:bodyPrototype"/>

</schema>
```

Layout Module: NCL30Layout.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Layout.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Layout module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:layout="http://www.ncl.org.br/NCL3.0/Layout"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Layout"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="regionBasePrototype">
    <attribute name="id" type="ID" use="optional"/>
    <attribute name="type" type="string" use="optional"/>
    <attribute name="device" type="string" use="optional"/>
  </complexType>

  <complexType name="regionPrototype">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element ref="layout:region" />
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="title" type="string" use="optional"/>
    <attribute name="height" type="string" use="optional"/>
    <attribute name="left" type="string" use="optional"/>
    <attribute name="right" type="string" use="optional"/>
    <attribute name="top" type="string" use="optional"/>
    <attribute name="bottom" type="string" use="optional"/>
    <attribute name="width" type="string" use="optional"/>
    <attribute name="zIndex" type="integer" use="optional"/>
  </complexType>


  <!-- declare global attributes in this module -->

  <!-- define the region attributeGroup -->
  <attributeGroup name="regionAttrs">
    <attribute name="region" type="string" use="optional"/>
  </attributeGroup>

  <!-- declare global elements in this module -->
  <element name="regionBase" type="layout:regionBasePrototype"/>
  <element name="region" type="layout:regionPrototype"/>

</schema>
```

Media Module: NCL30Media.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Media.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Media module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:media="http://www.ncl.org.br/NCL3.0/Media"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Media"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="mediaPrototype">
    <attribute name="id" type="ID" use="required"/>
    <attribute name="type" type="string" use="optional"/>
    <attribute name="src" type="anyURI" use="optional"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="media" type="media:mediaPrototype"/>

</schema>
```

Context Module: NCL30Context.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Context.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Context module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:context="http://www.ncl.org.br/NCL3.0/Context"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Context"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <!-- define the compositeNode element prototype -->
  <complexType name="contextPrototype">
    <attribute name="id" type="ID" use="required"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="context" type="context:contextPrototype"/>

</schema>
```

MediaContentAnchor Module: NCL30MediaContentAnchor.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30MediaContentAnchor.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Media Content Anchor module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:mediaAnchor="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  targetNamespace="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <!-- define the temporalAnchorAttrs attribute group -->
  <attributeGroup name="temporalAnchorAttrs">
    <attribute name="begin" type="string" use="optional"/>
    <attribute name="end" type="string" use="optional"/>
  </attributeGroup>

  <!-- define the textAnchorAttrs attribute group -->
  <attributeGroup name="textAnchorAttrs">
    <attribute name="text" type="string" use="optional"/>
    <attribute name="position" type="unsignedLong" use="optional"/>
  </attributeGroup>

  <!-- define the sampleAnchorAttrs attribute group -->
  <attributeGroup name="sampleAnchorAttrs">
    <attribute name="first" type="unsignedLong" use="optional"/>
    <attribute name="last" type="unsignedLong" use="optional"/>
  </attributeGroup>

  <!-- define the coordsAnchorAttrs attribute group -->
  <attributeGroup name="coordsAnchorAttrs">
    <attribute name="coords" type="string" use="optional"/>
  </attributeGroup>

  <!-- define the labelAnchorAttrs attribute group -->
  <attributeGroup name="labelAnchorAttrs">
    <attribute name="anchorLabel" type="string" use="optional"/>
  </attributeGroup>

  <complexType name="componentAnchorPrototype">
    <attribute name="id" type="ID" use="required"/>
    <attributeGroup ref="mediaAnchor:coordsAnchorAttrs" />
    <attributeGroup ref="mediaAnchor:temporalAnchorAttrs" />
    <attributeGroup ref="mediaAnchor:textAnchorAttrs" />
    <attributeGroup ref="mediaAnchor:sampleAnchorAttrs" />
    <attributeGroup ref="mediaAnchor:labelAnchorAttrs" />
  </complexType>

  <!-- declare global elements in this module -->
  <element name="area" type="mediaAnchor:componentAnchorPrototype"/>

</schema>
```

CompositeNodeInterface Module: NC30CompositeNodeInterface.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30CompositeNodeInterface.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Composite Node Interface module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:compositeInterface="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
  targetNamespace="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="compositeNodePortPrototype">
    <attribute name="id" type="ID" use="required" />
    <attribute name="component" type="IDREF" use="required"/>
    <attribute name="interface" type="string" use="optional" />
  </complexType>


  <!-- declare global elements in this module -->
  <element name="port" type="compositeInterface:compositeNodePortPrototype" />

</schema>
```

AttributeAnchor Module: NCL30AttributeAnchor.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30AttributeAnchor.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Attribute Anchor module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:attributeAnchor="http://www.ncl.org.br/NCL3.0/AttributeAnchor"
  targetNamespace="http://www.ncl.org.br/NCL3.0/AttributeAnchor"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="attributeAnchorPrototype">
    <attribute name="id" type="ID" use="required" />
    <attribute name="name" type="string" use="required" />
    <attribute name="value" type="string" use="optional" />
  </complexType>


  <!-- declare global elements in this module -->
  <element name="attribute" type="attributeAnchor:attributeAnchorPrototype"/>

</schema>
```

SwitchInterface Module: NCL30SwitchInterface.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30SwitchInterface.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Switch Interface module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:switchInterface="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  targetNamespace="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="mappingPrototype">
    <attribute name="component" type="IDREF" use="required"/>
    <attribute name="interface" type="string" use="optional"/>
  </complexType>

  <complexType name="switchPortPrototype">
    <sequence>
      <element ref="switchInterface:mapping" minOccurs="1"
maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="mapping" type="switchInterface:mappingPrototype"/>
  <element name="switchPort" type="switchInterface:switchPortPrototype" />

</schema>
```

Descriptor Module: NCL30Descriptor.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Descriptor.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Descriptor module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:descriptor="http://www.ncl.org.br/NCL3.0/Descriptor"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Descriptor"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <complexType name="descriptorParamPrototype">
    <attribute name="name" type="string" use="required" />
    <attribute name="value" type="string" use="required"/>
  </complexType>

  <complexType name="descriptorPrototype">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element ref="descriptor:descriptorParam"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="player" type="string" use="optional"/>
  </complexType>

<!--
Formatters should support the following descriptorParam names.
* For audio players: soundLevel; balanceLevel; trebleLevel; bassLevel.
* For text players: style, which refers to a style sheet with information for
text presentation.
* For visual media players: background, specifying the background color used to
fill the area of a region displaying media; scroll, which allows the
specification of how an author would like to configure the scroll in a region;
fit, indicating how an object will be presented (hidden, fill, meet, meetBest,
slice); transparency, indicating the degree of transparency of an object
presentation (the value must be between 0 and 1); visible, indicating if the
presentation is to be seen or hidden; and the object positioning parameters: top,
left, bottom, right, width, height, sie and bounds.
* For players in general: reusePlayer, which determines if a new player must be
instantiated or if a player already instantiated must be used; and playerLife,
which specifies what will happen to the player instance at the end of the
presentation.
-->

  <complexType name="descriptorBasePrototype">
    <attribute name="id" type="ID" use="optional"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="descriptorParam" type="descriptor:descriptorParamPrototype"/>
  <element name="descriptor" type="descriptor:descriptorPrototype"/>
  <element name="descriptorBase" type="descriptor:descriptorBasePrototype"/>

  <!-- declare global attributes in this module -->
  <attributeGroup name="descriptorAttrs">
    <attribute name="descriptor" type="string" use="optional"/>
  </attributeGroup>

</schema>
```

Linking Module: NCL30Linking.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Linking.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Linking module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:linking="http://www.ncl.org.br/NCL3.0/Linking"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Linking"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="paramPrototype">
    <attribute name="name" type="string" use="required"/>
    <attribute name="value" type="anySimpleType" use="required"/>
  </complexType>

  <complexType name="bindPrototype">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element ref="linking:bindParam"/>
    </sequence>
    <attribute name="role" type="string" use="required"/>
    <attribute name="component" type="IDREF" use="required"/>
    <attribute name="interface" type="string" use="optional"/>
  </complexType>

  <complexType name="linkPrototype">
    <sequence>
      <element ref="linking:linkParam" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="linking:bind" minOccurs="2" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="ID" use="optional"/>
    <attribute name="xconnector" type="string" use="required"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="linkParam" type="linking:paramPrototype"/>
  <element name="bindParam" type="linking:paramPrototype"/>
  <element name="bind" type="linking:bindPrototype" />
  <element name="link" type="linking:linkPrototype" />

</schema>
```

ConnectorCommonPart Module: NCL30ConnectorCommonPart.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCommonPart.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Common Part module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


<complexType name="parameterPrototype">
  <attribute name="name" type="string" use="required"/>
  <attribute name="type" type="string" use="optional"/>
</complexType>

<simpleType name="eventPrototype">
  <restriction base="string">
    <enumeration value="presentation" />
    <enumeration value="selection" />
    <enumeration value="attribution" />
    <enumeration value="composition" />
  </restriction>
</simpleType>

<simpleType name="logicalOperatorPrototype">
  <restriction base="string">
    <enumeration value="and" />
    <enumeration value="or" />
  </restriction>
</simpleType>

<simpleType name="transitionPrototype">
  <restriction base="string">
    <enumeration value="starts" />
    <enumeration value="stops" />
    <enumeration value="pauses" />
    <enumeration value="resumes" />
    <enumeration value="aborts" />
    <enumeration value="abortsFromPaused" />
  </restriction>
</simpleType>

</schema>
```

ConnectorAssessmentExpression Module: NCL30ConnectorAssessmentExpression.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI:
http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorAssessmentExpression.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Assessment Expression module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"

xmlns:connectorAssessmentExpression="http://www.ncl.org.br/NCL3.0/ConnectorAssess
mentExpression"
  xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<!-- import the definitions in the modules namespaces -->
<import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"

schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCommonPart.xsd
"/>

<simpleType name="comparatorPrototype">
  <restriction base="string">
    <enumeration value="eq" />
    <enumeration value="ne" />
    <enumeration value="gt" />
    <enumeration value="lt" />
    <enumeration value="gte" />
    <enumeration value="lte" />
  </restriction>
</simpleType>

<simpleType name="attributePrototype">
  <restriction base="string">
    <enumeration value="repeat" />
    <enumeration value="occurrences" />
    <enumeration value="state" />
    <enumeration value="nodeAttribute" />
  </restriction>
</simpleType>

<simpleType name="statePrototype">
  <restriction base="string">
    <enumeration value="sleeping" />
    <enumeration value="occurring" />
    <enumeration value="paused" />
  </restriction>
</simpleType>

<simpleType name="valueUnion">
  <union memberTypes="string" connectorAssessmentExpression:statePrototype"/>
</simpleType>
```

```
<complexType name="assessmentStatementPrototype" >
  <sequence>
    <element ref="connectorAssessmentExpression:attributeAssessment"/>
    <choice>
      <element ref="connectorAssessmentExpression:attributeAssessment"/>
      <element ref="connectorAssessmentExpression:valueAssessment"/>
    </choice>
  </sequence>
  <attribute name="comparator"
type="connectorAssessmentExpression:comparatorPrototype" use="required"/>
</complexType>

<complexType name="attributeAssessmentPrototype">
  <attribute name="role" type="string" use="required"/>
  <attribute name="eventType" type="connectorCommonPart:eventPrototype"
use="required"/>
  <attribute name="key" type="string" use="optional"/>
  <attribute name="attributeType"
type="connectorAssessmentExpression:attributePrototype" use="optional"/>
  <attribute name="offset" type="string" use="optional"/>
</complexType>

<complexType name="valueAssessmentPrototype">
  <attribute name="value" type="connectorAssessmentExpression:valueUnion"
use="required"/>
</complexType>

<complexType name="compoundStatementPrototype">
  <choice minOccurs="1" maxOccurs="unbounded">
    <element ref="connectorAssessmentExpression:assessmentStatement" />
    <element ref="connectorAssessmentExpression:compoundStatement" />
  </choice>
  <attribute name="operator" type="connectorCommonPart:logicalOperatorPrototype"
use="required"/>
  <attribute name="isNegated" type="boolean" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="assessmentStatement"
type="connectorAssessmentExpression:assessmentStatementPrototype" />
<element name="attributeAssessment"
type="connectorAssessmentExpression:attributeAssessmentPrototype" />
<element name="valueAssessment"
type="connectorAssessmentExpression:valueAssessmentPrototype" />
<element name="compoundStatement"
type="connectorAssessmentExpression:compoundStatementPrototype" />

</schema>
```

ConnectorCausalExpression Module: NCL30ConnectorCausalExpression.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI:
http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCausalExpression.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Causal Expression module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"

xmlns:connectorCausalExpression="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpr
ession"
    xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<!-- import the definitions in the modules namespaces -->
<import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"

schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCommonPart.xsd
"/>

<simpleType name="conditionRoleUnion">
  <union memberTypes="string connectorCausalExpression:conditionRolePrototype"/>
</simpleType>

<simpleType name="conditionRolePrototype">
  <restriction base="string">
    <enumeration value="onBegin" />
    <enumeration value="onEnd" />
    <enumeration value="onPause" />
    <enumeration value="onResume" />
    <enumeration value="onAbort" />
    <enumeration value="onAbortFromPaused" />
  </restriction>
</simpleType>

<simpleType name="maxUnion">
  <union memberTypes="positiveInteger
connectorCausalExpression:unboundedString"/>
</simpleType>

<simpleType name="unboundedString">
  <restriction base="string">
    <pattern value="unbounded"/>
  </restriction>
</simpleType>

<complexType name="simpleConditionPrototype">
  <attribute name="role" type="connectorCausalExpression:conditionRoleUnion"
use="required"/>
  <attribute name="eventType" type="connectorCommonPart:eventPrototype"
use="optional"/>
  <attribute name="key" type="string" use="optional"/>
  <attribute name="transition" type="connectorCommonPart:transitionPrototype"
use="optional"/>
```

```
  <attribute name="delay" type="string" use="optional"/>
  <attribute name="min" type="positiveInteger" use="optional"/>
  <attribute name="max" type="connectorCausalExpression:maxUnion"
use="optional"/>
  <attribute name="qualifier" type="connectorCommonPart:logicalOperatorPrototype"
use="optional"/>
</complexType>

<complexType name="compoundConditionPrototype">
  <attribute name="operator" type="connectorCommonPart:logicalOperatorPrototype"
use="required"/>
  <attribute name="delay" type="string" use="optional"/>
</complexType>

<simpleType name="actionRoleUnion">
  <union memberTypes="string connectorCausalExpression:actionNamePrototype"/>
</simpleType>

<simpleType name="actionNamePrototype">
  <restriction base="string">
    <enumeration value="start" />
    <enumeration value="stop" />
    <enumeration value="pause" />
    <enumeration value="resume" />
    <enumeration value="abort" />
    <enumeration value="set" />
  </restriction>
</simpleType>

<simpleType name="actionOperatorPrototype">
  <restriction base="string">
    <enumeration value="par" />
    <enumeration value="seq" />
  </restriction>
</simpleType>

<complexType name="simpleActionPrototype">
  <attribute name="role" type="connectorCausalExpression:actionRoleUnion"
use="required"/>
  <attribute name="eventType" type="connectorCommonPart:eventPrototype"
use="optional"/>
  <attribute name="actionType"
type="connectorCausalExpression:actionNamePrototype" use="optional"/>
  <attribute name="delay" type="string" use="optional"/>
  <attribute name="value" type="string" use="optional"/>
  <attribute name="repeat" type="positiveInteger" use="optional"/>
  <attribute name="repeatDelay" type="string" use="optional"/>
  <attribute name="min" type="positiveInteger" use="optional"/>
  <attribute name="max" type="connectorCausalExpression:maxUnion"
use="optional"/>
  <attribute name="qualifier"
type="connectorCausalExpression:actionOperatorPrototype" use="optional"/>
</complexType>

<complexType name="compoundActionPrototype">
  <choice minOccurs="2" maxOccurs="unbounded">
    <element ref="connectorCausalExpression:simpleAction" />
    <element ref="connectorCausalExpression:compoundAction" />
  </choice>
  <attribute name="operator"
type="connectorCausalExpression:actionOperatorPrototype" use="required"/>
  <attribute name="delay" type="string" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="simpleCondition"
type="connectorCausalExpression:simpleConditionPrototype" />
```

```
<element name="compoundCondition"
type="connectorCausalExpression:compoundConditionPrototype" />
<element name="simpleAction"
type="connectorCausalExpression:simpleActionPrototype" />
<element name="compoundAction"
type="connectorCausalExpression:compoundActionPrototype" />

</schema>
```

CausalConnector Module: NCL30CausalConnector.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30CausalConnector.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Causal Connector module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:causalConnector="http://www.ncl.org.br/NCL3.0/CausalConnector"
  targetNamespace="http://www.ncl.org.br/NCL3.0/CausalConnector"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<complexType name="causalConnectorPrototype">
  <attribute name="id" type="ID" use="required"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="causalConnector" type="causalConnector:causalConnectorPrototype"/>
</schema>
```

ConnectorBase Module: NCL30ConnectorBase.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorBase.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Base module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorBase="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<complexType name="connectorBasePrototype">
  <attribute name="id" type="ID" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="connectorBase" type="connectorBase:connectorBasePrototype"/>
</schema>
```

NCL30CausalConnectorFunctionality.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/
NCL30CausalConnectorFunctionality.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL CausalConnectorFunctionality module namespace.
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/
ConnectorCommonPart"
        xmlns:connectorAssessmentExpression="http://www.ncl.org.br/NCL3.0/
ConnectorAssessmentExpression"
        xmlns:connectorCausalExpression="http://www.ncl.org.br/NCL3.0/
ConnectorCausalExpression"
        xmlns:causalConnector="http://www.ncl.org.br/NCL3.0/
CausalConnector"
        xmlns:causalConnectorFunctionality="http://www.ncl.org.br/NCL3.0/
CausalConnectorFunctionality"
        targetNamespace="http://www.ncl.org.br/NCL3.0/
CausalConnectorFunctionality"
        elementFormDefault="qualified" attributeFormDefault="unqualified">

    <!-- import the definitions in the modules namespaces -->

    <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorCommonPart.xsd"/>
    <import
namespace="http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorAssessmentExpression.xsd"/>
    <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorCausalExpression.xsd"/>
    <import namespace="http://www.ncl.org.br/NCL3.0/CausalConnector"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30CausalConnector.xsd"/>

  <!-- ========================================================= -->
  <!-- CausalConnectorFunctionality                              -->
  <!-- ========================================================= -->
   <element name="connectorParam" type="connectorCommonPart:parameterPrototype"/>

    <!-- extends causalConnector element -->

    <complexType name="causalConnectorType">
      <complexContent>
        <extension base="causalConnector:causalConnectorPrototype">
          <sequence>
            <element ref="causalConnectorFunctionality:connectorParam"
minOccurs="0" maxOccurs="unbounded"/>
            <choice>
              <element ref="causalConnectorFunctionality:simpleCondition" />
```

```
              <element ref="causalConnectorFunctionality:compoundCondition" />
            </choice>
            <choice>
              <element ref="causalConnectorFunctionality:simpleAction" />
              <element ref="causalConnectorFunctionality:compoundAction" />
            </choice>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

    <!-- extends compoundCondition element -->

    <complexType name="compoundConditionType">
      <complexContent>
        <extension base="connectorCausalExpression:compoundConditionPrototype">
          <sequence>
            <choice>
              <element ref="causalConnectorFunctionality:simpleCondition" />
              <element ref="causalConnectorFunctionality:compoundCondition" />
            </choice>
            <choice minOccurs="1" maxOccurs="unbounded">
              <element ref="causalConnectorFunctionality:simpleCondition" />
              <element ref="causalConnectorFunctionality:compoundCondition" />
              <element ref="causalConnectorFunctionality:assessmentStatement" />
              <element ref="causalConnectorFunctionality:compoundStatement" />
            </choice>
          </sequence>
        </extension>
      </complexContent>
    </complexType>

  <element name="causalConnector"
type="causalConnectorFunctionality:causalConnectorType"
substitutionGroup="causalConnector:causalConnector"/>

  <element name="simpleCondition"
substitutionGroup="connectorCausalExpression:simpleCondition"/>

  <element name="compoundCondition"
type="causalConnectorFunctionality:compoundConditionType"
substitutionGroup="connectorCausalExpression:compoundCondition"/>

  <element name="simpleAction"
substitutionGroup="connectorCausalExpression:simpleAction"/>

  <element name="compoundAction"
substitutionGroup="connectorCausalExpression:compoundAction"/>

  <element name="assessmentStatement"
substitutionGroup="connectorAssessmentExpression:assessmentStatement"/>

  <element name="attributeAssessment"
substitutionGroup="connectorAssessmentExpression:attributeAssessment"/>

  <element name="valueAssessment"
substitutionGroup="connectorAssessmentExpression:valueAssessment"/>

  <element name="compoundStatement"
substitutionGroup="connectorAssessmentExpression:compoundStatement"/>

</schema>
```

TestRule Module: NCL30TestRule.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30TestRule.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL TestRule module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRule"
  targetNamespace="http://www.ncl.org.br/NCL3.0/TestRule"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="rulePrototype">
    <attribute name="id" type="ID" use="required"/>
    <attribute name="var" type="string" use="required"/>
    <attribute name="value" type="string" use="required"/>
    <attribute name="comparator" use="required">
      <simpleType>
        <restriction base="string">
          <enumeration value="eq"/>
          <enumeration value="ne"/>
          <enumeration value="gt"/>
          <enumeration value="gte"/>
          <enumeration value="lt"/>
          <enumeration value="lte"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>

  <complexType name="compositeRulePrototype">
    <choice minOccurs="2" maxOccurs="unbounded">
      <element ref="testRule:rule"/>
      <element ref="testRule:compositeRule"/>
    </choice>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="operator" use="required">
      <simpleType>
        <restriction base="string">
          <enumeration value="and"/>
          <enumeration value="or"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>

  <complexType name="ruleBasePrototype">
    <attribute name="id" type="ID" use="optional"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="rule" type="testRule:rulePrototype"/>
  <element name="compositeRule" type="testRule:compositeRulePrototype"/>
  <element name="ruleBase" type="testRule:ruleBasePrototype"/>

</schema>
```

TestRuleUse Module: NCL30TestRuleUse.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30TestRuleUse.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL TestRuleUse module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  targetNamespace="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="bindRulePrototype">
    <attribute name="constituent" type="IDREF" use="required" />
    <attribute name="rule" type="string" use="required" />
  </complexType>


  <!-- declare global elements in this module -->
  <element name="bindRule" type="testRule:bindRulePrototype"/>

</schema>
```

ContentControl Module: NCL30ContentControl.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ContentControl.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL ContentControl module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:contentControl="http://www.ncl.org.br/NCL3.0/ContentControl"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ContentControl"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="defaultComponentPrototype">
    <attribute name="component" type="IDREF" use="required" />
  </complexType>

  <!-- define the switch element prototype -->

  <complexType name="switchPrototype">
    <choice>
    <element ref="contentControl:defaultComponent" minOccurs="0" maxOccurs="1"/>
    </choice>
    <attribute name="id" type="ID" use="required"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="defaultComponent"
type="contentControl:defaultComponentPrototype"/>
  <element name="switch" type="contentControl:switchPrototype"/>

</schema>
```

DescriptorControl Module: NCL30DescriptorControl.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30DescriptorControl.xsd
Author: TeleMidia Laboratory
Revision: 19/06/2006

Schema for the NCL DescriptorControl module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:descriptorControl="http://www.ncl.org.br/NCL3.0/DescriptorControl"
  targetNamespace="http://www.ncl.org.br/NCL3.0/DescriptorControl"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="defaultDescriptorPrototype">
    <attribute name="descriptor" type="IDREF" use="required" />
  </complexType>

  <!-- define the descriptor switch element prototype -->
  <complexType name="descriptorSwitchPrototype">
    <choice>
    <element ref="descriptorControl:defaultDescriptor" minOccurs="0"
maxOccurs="1"/>
    </choice>
    <attribute name="id" type="ID" use="required”/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="defaultDescriptor"
type="descriptorControl:defaultDescriptorPrototype"/>
  <element name="descriptorSwitch"
type="descriptorControl:descriptorSwitchPrototype"/>

</schema>
```

Timing Module: NCL30Timing.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Timing.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Timing module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:timing="http://www.ncl.org.br/NCL3.0/Timing"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Timing"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <!-- declare global attributes in this module -->

  <!-- define the explicitDur attribute group -->
  <attributeGroup name="explicitDurAttrs">
    <attribute name="explicitDur" type="string" use="optional"/>
  </attributeGroup>

  <!-- define the freeze attribute group -->
  <attributeGroup name="freezeAttrs">
    <attribute name="freeze" type="boolean" use="optional"/>
  </attributeGroup>

</schema>
```

Import Module: NCL30Import.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30Import.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Import module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:import="http://www.ncl.org.br/NCL3.0/Import"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Import"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="importBasePrototype">
      <attribute name="alias" type="ID" use="optional"/>
      <attribute name="region" type="IDREF" use="optional"/>
      <attribute name="documentURI" type="anyURI" use="required"/>
  </complexType>

  <complexType name="importNCLPrototype">
      <attribute name="alias" type="ID" use="optional"/>
      <attribute name="documentURI" type="anyURI" use="required"/>
  </complexType>

  <complexType name="importedDocumentBasePrototype">
    <sequence minOccurs="1" maxOccurs="unbounded">
      <element ref="import:importNCL" />
    </sequence>
    <attribute name="id" type="ID" use="optional" />
  </complexType>


  <!-- declare global elements in this module -->
  <element name="importBase" type="import:importBasePrototype"/>
  <element name="importNCL" type="import:importNCLPrototype"/>
  <element name="importedDocumentBase"
type="import:importedDocumentBasePrototype"/>

</schema>
```

EntityReuse Module: NCL30EntityReuse.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30EntityReuse.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL EntityReuse module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:entityReuse="http://www.ncl.org.br/NCL3.0/EntityReuse"
  targetNamespace="http://www.ncl.org.br/NCL3.0/EntityReuse"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <attributeGroup name="entityReuseAttrs">
    <attribute name="refer" type="string" use="optional"/>
  </attributeGroup>

</schema>
```

ExtendedEntityReuse Module: NCL30ExtendedEntityReuse.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ExtendedEntityReuse.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL ExtendedEntityReuse module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:extendedEntityReuse="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <attributeGroup name="extendedEntityReuseAttrs">
    <attribute name="newInstance" type="boolean" use="optional"/>
  </attributeGroup>

</schema>
```

KeyNavigation Module: NCL30KeyNavigation.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30KeyNavigation.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL KeyNavigation module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:keyNavigation="http://www.ncl.org.br/NCL3.0/KeyNavigation"
  targetNamespace="http://www.ncl.org.br/NCL3.0/KeyNavigation"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<simpleType name="colorPrototype">
  <restriction base="string">
    <enumeration value="white" />
    <enumeration value="black" />
    <enumeration value="silver" />
    <enumeration value="gray" />
    <enumeration value="red" />
    <enumeration value="maroon" />
    <enumeration value="fuchsia" />
    <enumeration value="purple" />
    <enumeration value="lime" />
    <enumeration value="green" />
    <enumeration value="yellow" />
    <enumeration value="olive" />
    <enumeration value="blue" />
    <enumeration value="navy" />
    <enumeration value="aqua" />
    <enumeration value="teal" />
  </restriction>
</simpleType>

  <!-- declare global attributes in this module -->

  <!-- define the keyNavigation attribute group -->
  <attributeGroup name="keyNavigationAttrs">
    <attribute name="moveLeft" type="string" use="optional"/>
    <attribute name="moveRight" type="string" use="optional"/>
    <attribute name="moveUp" type="string" use="optional"/>
    <attribute name="moveDown" type="string" use="optional"/>
    <attribute name="focusIndex" type="IDREF" use="optional"/>
    <attribute name="focusBorderColor" type="keyNavigation:colorPrototype"
use="optional"/>
    <attribute name="focusBorderWidth" type="string" use="optional"/>
    <attribute name="focusBorderTransparency" type="string" use="optional"/>
    <attribute name="focusScr" type="string" use="optional"/>
    <attribute name="focusSelScr" type="string" use="optional"/>
    <attribute name="selBorderColor" type="keyNavigation:colorPrototype"
use="optional"/>
  </attributeGroup>


</schema>
```

TransitionBase Module: NCL30TransitionBase.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30TransitionBase.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Transition Base module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:transitionBase="http://www.ncl.org.br/NCL3.0/TransitionBase"
  targetNamespace="http://www.ncl.org.br/NCL3.0/TransitionBase"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<complexType name="transitionBasePrototype">
  <attribute name="id" type="ID" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="transitionBase" type="transitionBase:transitionBasePrototype"/>
</schema>
```

5. NCL 3.0 Language Profiles for Digital TV

Each NCL profile can group a subset of NCL modules, allowing the creation of languages according to user needs.

Any document in conformance with NCL profiles must have the <ncl> element as its root element.

The profiles defined for the digital TV are:

NCL 3.0 Enhanced DTV Profile

NCL 3.0 Enhanced DTV profile includes the Structure, Layout, Media, Context, MediaContentAnchor, CompositeNodeInterface, AttributeAnchor, SwitchInterface, Descriptor, Linking, CausalConnectorFunctionality, ConnectorBase, TestRule, TestRuleUse, ContentControl, DescriptorControl, Timing, Import, EntityReuse, ExtendedEntityReuse KeyNavigation and TransitionBase modules of NCL 3.0, and also the BasicTransition, and Metainformation modules of SMIL2.0.

The tables of Section 3.2 show each module element, already extended by the attributes and child elements inherited from other modules, for this profile.

Section 5.1 presents the definition of the NCL 3.0 Enhanced DTV profile using XML Schema.

NCL 3.0 Basic DTV Profile

NCL 3.0 Basic DTV profile includes the Structure, Layout, Media, Context, MediaContentAnchor, CompositeNodeInterface, AttributeAnchor, SwitchInterface, Descriptor, Linking, CausalConnectorFunctionality, ConnectorBase, TestRule, TestRuleUse, ContentControl, DescriptorControl, Timing, Import, EntityReuse, ExtendedEntityReuse and KeyNavigation modules.

The tables of Section 3.3 show each module element for this profile, already extended by the attributes and child elements inherited from other modules.

Section 5.2 presents the definition of the NCL 3.0 Basic DTV profile using XML Schema.

5.1. The Schema of the NCL 3.0 Enhanced DTV Profile

This section presents the definition of the NCL 3.0 Enhanced DTV profile using XML Schema.

NCL30EDTV.xsd

```
<!--
XML Schema for the NCL Language

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/profiles/NCL30EDTV.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:attributeAnchor="http://www.ncl.org.br/NCL3.0/AttributeAnchor"
  xmlns:compositeInterface="http://www.ncl.org.br/NCL3.0/
CompositeNodeInterface"
  xmlns:causalConnectorFunctionality="http://www.ncl.org.br/NCL3.0/
CausalConnectorFunctionality"
  xmlns:connectorBase="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  xmlns:contentControl="http://www.ncl.org.br/NCL3.0/ContentControl"
  xmlns:context="http://www.ncl.org.br/NCL3.0/Context"
  xmlns:descriptor="http://www.ncl.org.br/NCL3.0/Descriptor"
  xmlns:entityReuse="http://www.ncl.org.br/NCL3.0/EntityReuse"
  xmlns:extendedEntityReuse="http://www.ncl.org.br/NCL3.0/
ExtendedEntityReuse"
  xmlns:descriptorControl="http://www.ncl.org.br/NCL3.0/
DescriptorControl"
  xmlns:import="http://www.ncl.org.br/NCL3.0/Import"
  xmlns:keyNavigation="http://www.ncl.org.br/NCL3.0/KeyNavigation"
  xmlns:layout="http://www.ncl.org.br/NCL3.0/Layout"
  xmlns:linking="http://www.ncl.org.br/NCL3.0/Linking"
  xmlns:media="http://www.ncl.org.br/NCL3.0/Media"
  xmlns:mediaAnchor="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
  xmlns:switchInterface="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRule"
  xmlns:testRuleUse="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  xmlns:timing="http://www.ncl.org.br/NCL3.0/Timing"
  xmlns:transitionBase="http://www.ncl.org.br/NCL3.0/TransitionBase"
  xmlns:metainformation="http://www.w3.org/2001/SMIL20/Metainformation"
  xmlns:basicTransition="http://www.w3.org/2001/SMIL20/BasicTransitions"
  xmlns:profile="http://www.ncl.org.br/NCL3.0/EDTVProfile"
  targetNamespace="http://www.ncl.org.br/NCL3.0/EDTVProfile"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <!-- import the definitions in the modules namespaces -->
  <import namespace="http://www.ncl.org.br/NCL3.0/AttributeAnchor"
```

```
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30AttributeAnchor.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30CompositeNodeInterface.xsd"/>
  <import
namespace="http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30CausalConnectorFunctionality.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorBase"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorBase.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/ContentControl"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ContentControl.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Context"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Context.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Descriptor"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Descriptor.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/DescriptorControl"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30DescriptorControl.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/EntityReuse"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30EntityReuse.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ExtendedEntityReuse.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Import"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Import.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/KeyNavigation"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30KeyNavigation.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Layout"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Layout.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Linking"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Linking.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Media"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Media.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30MediaContentAnchor.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Structure"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Structure.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/SwitchInterface"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30SwitchInterface.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/TestRule"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30TestRule.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/TestRuleUse"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30TestRuleUse.xsd"/>
```

```
  <import namespace="http://www.ncl.org.br/NCL3.0/Timing"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Timing.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/TransitionBase"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30TransitionBase.xsd"/>
  <import namespace="http://www.w3.org/2001/SMIL20/Metainformation"
    schemaLocation="http://www.w3.org/2001/SMIL20/
smil20-Metainformation.xsd"/>
  <import namespace="http://www.w3.org/2001/SMIL20/BasicTransitions"
    schemaLocation="http://www.w3.org/2001/SMIL20/
smil20-BasicTransitions.xsd"/>


  <!-- ============================================================ -->
  <!-- Structure                                                    -->
  <!-- ============================================================ -->
  <!-- extends ncl element -->

  <element name="ncl" substitutionGroup="structure:ncl"/>

  <!-- extends head element -->

  <complexType name="headType">
    <complexContent>
      <extension base="structure:headPrototype">
        <sequence>
          <element ref="profile:importedDocumentBase" minOccurs="0"
maxOccurs="1"/>
          <element ref="profile:ruleBase" minOccurs="0" maxOccurs="1"/>
          <element ref="profile:transitionBase" minOccurs="0"
maxOccurs="1"/>
          <element ref="profile:regionBase" minOccurs="0"
maxOccurs="unbounded"/>
          <element ref="profile:descriptorBase" minOccurs="0"
maxOccurs="1"/>
          <element ref="profile:connectorBase" minOccurs="0"
maxOccurs="1"/>
          <element ref="profile:meta" minOccurs="0"
maxOccurs="unbounded"/>
          <element ref="profile:metadata" minOccurs="0"
maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <element name="head" type="profile:headType"
substitutionGroup="structure:head"/>

  <!-- extends body element -->

  <complexType name="bodyType">
    <complexContent>
      <extension base="structure:bodyPrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <group ref="profile:contextInterfaceElementGroup"/>
          <element ref="profile:media"/>
          <element ref="profile:context"/>
          <element ref="profile:switch"/>
```

```
          <element ref="profile:link"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="body" type="profile:bodyType"
substitutionGroup="structure:body"/>

  <!-- ============================================================ -->
  <!-- Layout                                                       -->
  <!-- ============================================================ -->
  <!-- extends regionBase element -->

  <complexType name="regionBaseType">
    <complexContent>
      <extension base="layout:regionBasePrototype">
        <choice minOccurs="1" maxOccurs="unbounded">
          <element ref="profile:importBase"/>
          <element ref="profile:region"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="regionType">
    <complexContent>
      <extension base="layout:regionPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="regionBase" type="profile:regionBaseType"
substitutionGroup="layout:regionBase"/>
  <element name="region" type="profile:regionType"
substitutionGroup="layout:region"/>

  <!-- ============================================================ -->
  <!-- Media                                                        -->
  <!-- ============================================================ -->
  <!-- extends Media elements -->

  <!-- media interface element groups -->
  <group name="mediaInterfaceElementGroup">
    <choice>
      <element ref="profile:area"/>
      <element ref="profile:attribute"/>
    </choice>
  </group>

  <complexType name="mediaType">
    <complexContent>
      <extension base="media:mediaPrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <group ref="profile:mediaInterfaceElementGroup"/>
        </choice>
        <attributeGroup ref="descriptor:descriptorAttrs"/>
        <attributeGroup ref="entityReuse:entityReuseAttrs"/>
        <attributeGroup
ref="extendedEntityReuse:extendedEntityReuseAttrs"/>
```

```
      </extension>
    </complexContent>
  </complexType>

  <element name="media" type="profile:mediaType"
substitutionGroup="media:media"/>

  <!-- ========================================================== -->
  <!-- Context                                                    -->
  <!-- ========================================================== -->
  <!-- extends context element -->

  <!-- composite node interface element groups -->
  <group name="contextInterfaceElementGroup">
    <choice>
      <element ref="profile:port"/>
      <element ref="profile:attribute"/>
    </choice>
  </group>

  <complexType name="contextType">
    <complexContent>
      <extension base="context:contextPrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <group ref="profile:contextInterfaceElementGroup"/>
          <element ref="profile:media"/>
          <element ref="profile:context"/>
          <element ref="profile:link"/>
          <element ref="profile:switch"/>
        </choice>
        <attributeGroup ref="descriptor:descriptorAttrs"/>
        <attributeGroup ref="entityReuse:entityReuseAttrs"/>
      </extension>
    </complexContent>
  </complexType>

  <element name="context" type="profile:contextType"
substitutionGroup="context:context"/>

  <!-- ========================================================== -->
  <!-- MediaContentAnchor                                         -->
  <!-- ========================================================== -->
  <!-- extends area element -->

  <complexType name="componentAnchorType">
    <complexContent>
      <extension base="mediaAnchor:componentAnchorPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="area" type="profile:componentAnchorType"
substitutionGroup="mediaAnchor:area"/>

  <!-- ========================================================== -->
  <!-- CompositeNodeInterface                                     -->
  <!-- ========================================================== -->
  <!-- extends port element -->

  <complexType name="compositeNodePortType">
```

```
    <complexContent>
      <extension base="compositeInterface:compositeNodePortPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="port" type="profile:compositeNodePortType"
substitutionGroup="compositeInterface:port"/>

  <!-- ========================================================= -->
  <!-- AtributeAnchor                                            -->
  <!-- ========================================================= -->
  <!-- extends attribute element -->

  <complexType name="attributeAnchorType">
    <complexContent>
      <extension base="attributeAnchor:attributeAnchorPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="attribute" type="profile:attributeAnchorType"
substitutionGroup="attributeAnchor:attribute"/>

  <!-- ========================================================= -->
  <!-- SwitchInterface                                           -->
  <!-- ========================================================= -->
  <!-- extends switchPort element -->

  <complexType name="switchPortType">
    <complexContent>
      <extension base="switchInterface:switchPortPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="mapping" substitutionGroup="switchInterface:mapping"/>
  <element name="switchPort" type="profile:switchPortType"
substitutionGroup="switchInterface:switchPort"/>

  <!-- ========================================================= -->
  <!-- Descriptor                                                -->
  <!-- ========================================================= -->

  <!-- substitutes descriptorParam element -->

  <element name="descriptorParam"
substitutionGroup="descriptor:descriptorParam"/>


  <!-- extends descriptor element -->

  <complexType name="descriptorType">
    <complexContent>
      <extension base="descriptor:descriptorPrototype">
        <attributeGroup ref="layout:regionAttrs"/>
        <attributeGroup ref="timing:explicitDurAttrs"/>
        <attributeGroup ref="timing:freezeAttrs"/>
        <attributeGroup ref="keyNavigation:keyNavigationAttrs"/>
        <attributeGroup ref="profile:transitionAttrs"/>
```

```
      </extension>
    </complexContent>
  </complexType>

  <element name="descriptor" type="profile:descriptorType"
substitutionGroup="descriptor:descriptor"/>

  <!-- extends descriptorBase element -->
  <complexType name="descriptorBaseType">
    <complexContent>
      <extension base="descriptor:descriptorBasePrototype">
        <choice minOccurs="1" maxOccurs="unbounded">
          <element ref="profile:importBase"/>
          <element ref="profile:descriptor"/>
          <element ref="profile:descriptorSwitch"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="descriptorBase" type="profile:descriptorBaseType"
substitutionGroup="descriptor:descriptorBase"/>

  <!-- ========================================================= -->
  <!-- Linking                                                   -->
  <!-- ========================================================= -->

  <!-- substitutes linkParam and bindParam elements -->
  <element name="linkParam" substitutionGroup="linking:linkParam"/>
  <element name="bindParam" substitutionGroup="linking:bindParam"/>

  <!-- extends bind element and link element, as a consequence-->

  <complexType name="bindType">
    <complexContent>
      <extension base="linking:bindPrototype">
        <attributeGroup ref="descriptor:descriptorAttrs"/>
      </extension>
    </complexContent>
  </complexType>

  <element name="bind" type="profile:bindType"
substitutionGroup="linking:bind"/>

  <!-- extends link element -->
  <complexType name="linkType">
    <complexContent>
      <extension base="linking:linkPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="link" type="profile:linkType"
substitutionGroup="linking:link"/>

  <!-- ========================================================= -->
  <!-- Connector                                                 -->
  <!-- ========================================================= -->
  <!-- extends connectorBase element -->
```

```
    <complexType name="connectorBaseType">
        <complexContent>
            <extension base="connectorBase:connectorBasePrototype">
              <choice minOccurs="0" maxOccurs="unbounded">
                 <element ref="profile:importBase"/>
                 <element ref="profile:causalConnector" />
              </choice>
            </extension>
        </complexContent>
    </complexType>

  <element name="connectorBase" type="profile:connectorBaseType"
substitutionGroup="connectorBase:connectorBase"/>

  <element name="causalConnector"
substitutionGroup="causalConnectorFunctionality:causalConnector"/>

  <element name="connectorParam"
substitutionGroup="causalConnectorFunctionality:connectorParam"/>

  <element name="simpleCondition"
substitutionGroup="causalConnectorFunctionality:simpleCondition"/>

  <element name="compoundCondition"
substitutionGroup="causalConnectorFunctionality:compoundCondition"/>

  <element name="simpleAction"
substitutionGroup="causalConnectorFunctionality:simpleAction"/>

  <element name="compoundAction"
substitutionGroup="causalConnectorFunctionality:compoundAction"/>

  <element name="assessmentStatement"
substitutionGroup="causalConnectorFunctionality:assessmentStatement"/>

  <element name="attributeAssessment"
substitutionGroup="causalConnectorFunctionality:attributeAssessment"/>

  <element name="valueAssessment"
substitutionGroup="causalConnectorFunctionality:valueAssessment"/>

  <element name="compoundStatement"
substitutionGroup="causalConnectorFunctionality:compoundStatement"/>

  <!-- ============================================================ -->
  <!-- TestRule                                                     -->
  <!-- ============================================================ -->
  <!-- extends rule element -->
  <complexType name="ruleType">
    <complexContent>
      <extension base="testRule:rulePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="rule" type="profile:ruleType"
substitutionGroup="testRule:rule"/>

  <!-- extends compositeRule element -->
  <complexType name="compositeRuleType">
```

```
    <complexContent>
      <extension base="testRule:compositeRulePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="compositeRule" type="profile:compositeRuleType"
substitutionGroup="testRule:compositeRule"/>

  <!-- extends ruleBase element -->
  <complexType name="ruleBaseType">
    <complexContent>
      <extension base="testRule:ruleBasePrototype">
        <choice minOccurs="1" maxOccurs="unbounded">
          <element ref="profile:importBase"/>
          <element ref="profile:rule"/>
          <element ref="profile:compositeRule"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="ruleBase" type="profile:ruleBaseType"
substitutionGroup="testRule:ruleBase"/>

  <!-- ========================================================== -->
  <!-- TestRuleUse                                                -->
  <!-- ========================================================== -->
  <!-- extends bindRule element -->
  <complexType name="bindRuleType">
    <complexContent>
      <extension base="testRuleUse:bindRulePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="bindRule" type="profile:bindRuleType"
substitutionGroup="testRuleUse:bindRule"/>

  <!-- ========================================================== -->
  <!-- ContentControl                                             -->
  <!-- ========================================================== -->
  <!-- extends switch element -->

  <!-- switch interface element groups -->
  <group name="switchInterfaceElementGroup">
    <choice>
      <element ref="profile:switchPort"/>
    </choice>
  </group>

  <!-- extends defaultComponent element -->
  <complexType name="defaultComponentType">
    <complexContent>
      <extension base="contentControl:defaultComponentPrototype">
      </extension>
    </complexContent>
  </complexType>
```

```
  <element name="defaultComponent" type="profile:defaultComponentType"
substitutionGroup="contentControl:defaultComponent"/>

  <complexType name="switchType">
    <complexContent>
      <extension base="contentControl:switchPrototype">
          <choice minOccurs="0" maxOccurs="unbounded">
            <group ref="profile:switchInterfaceElementGroup"/>
            <element ref="profile:bindRule"/>
            <element ref="profile:switch"/>
            <element ref="profile:media"/>
            <element ref="profile:context"/>
          </choice>
        <attributeGroup ref="entityReuse:entityReuseAttrs"/>
      </extension>
    </complexContent>
  </complexType>

  <element name="switch" type="profile:switchType"
substitutionGroup="contentControl:switch"/>

  <!-- ========================================================= -->
  <!-- DescriptorControl                                         -->
  <!-- ========================================================= -->
  <!-- extends defaultDescriptor element -->
  <complexType name="defaultDescriptorType">
    <complexContent>
      <extension base="descriptorControl:defaultDescriptorPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="defaultDescriptor" type="profile:defaultDescriptorType"
substitutionGroup="descriptorControl:defaultDescriptor"/>

  <!-- extends descriptorSwitch element -->

  <complexType name="descriptorSwitchType">
    <complexContent>
      <extension base="descriptorControl:descriptorSwitchPrototype">
          <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="profile:descriptor"/>
            <element ref="profile:bindRule"/>
          </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="descriptorSwitch" type="profile:descriptorSwitchType"
substitutionGroup="descriptorControl:descriptorSwitch"/>

  <!-- ========================================================= -->
  <!-- Timing                                                    -->
  <!-- ========================================================= -->


  <!-- ========================================================= -->
  <!-- Import                                                    -->
  <!--========================================================== -->
  <complexType name="importBaseType">
```

```
    <complexContent>
      <extension base="import:importBasePrototype">
      </extension>
    </complexContent>
  </complexType>

  <complexType name="importNCLType">
    <complexContent>
      <extension base="import:importNCLPrototype">
      </extension>
    </complexContent>
  </complexType>

  <complexType name="importedDocumentBaseType">
    <complexContent>
      <extension base="import:importedDocumentBasePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="importBase" type="profile:importBaseType"
substitutionGroup="import:importBase"/>

  <element name="importNCL" type="profile:importNCLType"
substitutionGroup="import:importNCL"/>
  <element name="importedDocumentBase"
type="profile:importedDocumentBaseType"
substitutionGroup="import:importedDocumentBase"/>

  <!-- ============================================================ -->
  <!-- EntityReuse                                                  -->
  <!-- ============================================================ -->

  <!-- ============================================================ -->
  <!-- ExtendedEntityReuse                                          -->
  <!-- ============================================================ -->

  <!-- ============================================================ -->
  <!-- KeyNavigation                                                -->
  <!-- ============================================================ -->

  <!-- ============================================================ -->
  <!-- TransitionBase                                               -->
  <!-- ============================================================ -->
  <!-- extends transitionBase element -->

  <complexType name="transitionBaseType">
    <complexContent>
      <extension base="transitionBase:transitionBasePrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <element ref="profile:transition"/>
          <element ref="profile:importBase"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="transitionBase" type="profile:transitionBaseType"
substitutionGroup="transitionBase:transitionBase"/>
```

```
  <!-- ========================================================== -->
  <!-- BasicTransition                                            -->
  <!-- ========================================================== -->

  <attributeGroup name="transitionAttrs">
    <attribute ref="basicTransition:transIn"/>
    <attribute ref="basicTransition:transOut"/>
  </attributeGroup>

  <element name="transition"
substitutionGroup="basicTransition:transition"/>

  <!-- ========================================================== -->
  <!-- Metainformation                                            -->
  <!-- ========================================================== -->

  <element name="meta" substitutionGroup="metainformation:meta"/>

  <element name="metadata" substitutionGroup="metainformation:metadata"/>

</schema>
```

5.2. The Schema of the NCL 3.0 Basic DTV Profile

This section presents the definition of the NCL 3.0 Basic DTV profile using XML Schema.

NCL30BDTV.xsd

```
<!--
XML Schema for the NCL Language

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/profiles/NCL30BDTV.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:attributeAnchor="http://www.ncl.org.br/NCL3.0/AttributeAnchor"
  xmlns:compositeInterface="http://www.ncl.org.br/NCL3.0/
CompositeNodeInterface"
  xmlns:causalConnectorFunctionality="http://www.ncl.org.br/NCL3.0/
CausalConnectorFunctionality"
  xmlns:connectorBase="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  xmlns:contentControl="http://www.ncl.org.br/NCL3.0/ContentControl"
  xmlns:context="http://www.ncl.org.br/NCL3.0/Context"
  xmlns:descriptor="http://www.ncl.org.br/NCL3.0/Descriptor"
  xmlns:entityReuse="http://www.ncl.org.br/NCL3.0/EntityReuse"
  xmlns:extendedEntityReuse="http://www.ncl.org.br/NCL3.0/
ExtendedEntityReuse"
  xmlns:descriptorControl="http://www.ncl.org.br/NCL3.0/
DescriptorControl"
  xmlns:import="http://www.ncl.org.br/NCL3.0/Import"
  xmlns:keyNavigation="http://www.ncl.org.br/NCL3.0/KeyNavigation"
  xmlns:layout="http://www.ncl.org.br/NCL3.0/Layout"
  xmlns:linking="http://www.ncl.org.br/NCL3.0/Linking"
  xmlns:media="http://www.ncl.org.br/NCL3.0/Media"
  xmlns:mediaAnchor="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
  xmlns:switchInterface="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRule"
  xmlns:testRuleUse="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  xmlns:timing="http://www.ncl.org.br/NCL3.0/Timing"
  xmlns:profile="http://www.ncl.org.br/NCL3.0/BDTVProfile"
  targetNamespace="http://www.ncl.org.br/NCL3.0/BDTVProfile"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <!-- import the definitions in the modules namespaces -->
  <import namespace="http://www.ncl.org.br/NCL3.0/AttributeAnchor"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30AttributeAnchor.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30CompositeNodeInterface.xsd"/>
```

```
  <import
namespace="http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30CausalConnectorFunctionality.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorBase"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorBase.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/ContentControl"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ContentControl.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Context"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Context.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Descriptor"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Descriptor.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/DescriptorControl"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30DescriptorControl.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/EntityReuse"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30EntityReuse.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ExtendedEntityReuse.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Import"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Import.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/KeyNavigation"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30KeyNavigation.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Layout"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Layout.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Linking"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Linking.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Media"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Media.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30MediaContentAnchor.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Structure"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Structure.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/SwitchInterface"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30SwitchInterface.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/TestRule"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30TestRule.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/TestRuleUse"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30TestRuleUse.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Timing"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Timing.xsd"/>

  <!-- ========================================================== -->
```

```
  <!-- Structure                                                    -->
  <!-- ============================================================ -->
  <!-- extends ncl element -->

  <element name="ncl" substitutionGroup="structure:ncl"/>

  <!-- extends head element -->

  <complexType name="headType">
    <complexContent>
      <extension base="structure:headPrototype">
        <sequence>
          <element ref="profile:importedDocumentBase" minOccurs="0"
maxOccurs="1"/>
          <element ref="profile:ruleBase" minOccurs="0" maxOccurs="1"/>
          <element ref="profile:regionBase" minOccurs="0"
maxOccurs="unbounded"/>
          <element ref="profile:descriptorBase" minOccurs="0"
maxOccurs="1"/>
          <element ref="profile:connectorBase" minOccurs="0"
maxOccurs="1"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <element name="head" type="profile:headType"
substitutionGroup="structure:head"/>

  <!-- extends body element -->

  <complexType name="bodyType">
    <complexContent>
      <extension base="structure:bodyPrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <group ref="profile:contextInterfaceElementGroup"/>
          <element ref="profile:media"/>
          <element ref="profile:context"/>
          <element ref="profile:switch"/>
          <element ref="profile:link"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="body" type="profile:bodyType"
substitutionGroup="structure:body"/>

  <!-- ============================================================ -->
  <!-- Layout                                                       -->
  <!-- ============================================================ -->
  <!-- extends regionBase element -->

  <complexType name="regionBaseType">
    <complexContent>
      <extension base="layout:regionBasePrototype">
        <choice minOccurs="1" maxOccurs="unbounded">
          <element ref="profile:region"/>
          <element ref="profile:importBase"/>
        </choice>
```

```
      </extension>
    </complexContent>
  </complexType>

  <complexType name="regionType">
    <complexContent>
      <extension base="layout:regionPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="regionBase" type="profile:regionBaseType"
substitutionGroup="layout:regionBase"/>
  <element name="region" type="profile:regionType"
substitutionGroup="layout:region"/>

  <!-- ========================================================== -->
  <!-- Media                                                      -->
  <!-- ========================================================== -->
  <!-- extends Media elements -->

  <!-- media interface element groups -->
  <group name="mediaInterfaceElementGroup">
    <choice>
      <element ref="profile:area"/>
      <element ref="profile:attribute"/>
    </choice>
  </group>

  <complexType name="mediaType">
    <complexContent>
      <extension base="media:mediaPrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <group ref="profile:mediaInterfaceElementGroup"/>
        </choice>
        <attributeGroup ref="descriptor:descriptorAttrs"/>
        <attributeGroup ref="entityReuse:entityReuseAttrs"/>
        <attributeGroup
ref="extendedEntityReuse:extendedEntityReuseAttrs"/>
      </extension>
    </complexContent>
  </complexType>

  <element name="media" type="profile:mediaType"
substitutionGroup="media:media"/>

  <!-- ========================================================== -->
  <!-- Context                                                    -->
  <!-- ========================================================== -->
  <!-- extends context element -->

  <!-- composite node interface element groups -->
  <group name="contextInterfaceElementGroup">
    <choice>
      <element ref="profile:port"/>
      <element ref="profile:attribute"/>
    </choice>
  </group>

  <complexType name="contextType">
```

```
    <complexContent>
      <extension base="context:contextPrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <group ref="profile:contextInterfaceElementGroup"/>
          <element ref="profile:media"/>
          <element ref="profile:context"/>
          <element ref="profile:link"/>
          <element ref="profile:switch"/>
        </choice>
        <attributeGroup ref="descriptor:descriptorAttrs"/>
        <attributeGroup ref="entityReuse:entityReuseAttrs"/>
      </extension>
    </complexContent>
  </complexType>

  <element name="context" type="profile:contextType"
substitutionGroup="context:context"/>

  <!-- ========================================================= -->
  <!-- MediaContentAnchor                                        -->
  <!-- ========================================================= -->
  <!-- extends area element -->

  <complexType name="componentAnchorType">
    <complexContent>
      <extension base="mediaAnchor:componentAnchorPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="area" type="profile:componentAnchorType"
substitutionGroup="mediaAnchor:area"/>

  <!-- ========================================================= -->
  <!-- CompositeNodeInterface                                    -->
  <!-- ========================================================= -->
  <!-- extends port element -->

  <complexType name="compositeNodePortType">
    <complexContent>
      <extension base="compositeInterface:compositeNodePortPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="port" type="profile:compositeNodePortType"
substitutionGroup="compositeInterface:port"/>

  <!-- ========================================================= -->
  <!-- AtributeAnchor                                            -->
  <!-- ========================================================= -->
  <!-- extends attribute element -->

  <complexType name="attributeAnchorType">
    <complexContent>
      <extension base="attributeAnchor:attributeAnchorPrototype">
      </extension>
    </complexContent>
  </complexType>
```

```
  <element name="attribute" type="profile:attributeAnchorType"
substitutionGroup="attributeAnchor:attribute"/>

  <!-- ============================================================ -->
  <!-- SwitchInterface                                              -->
  <!-- ============================================================ -->
  <!-- extends switchPort element -->

  <complexType name="switchPortType">
    <complexContent>
      <extension base="switchInterface:switchPortPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="mapping" substitutionGroup="switchInterface:mapping"/>
  <element name="switchPort" type="profile:switchPortType"
substitutionGroup="switchInterface:switchPort"/>

  <!-- ============================================================ -->
  <!-- Descriptor                                                   -->
  <!-- ============================================================ -->

  <!-- substitutes descriptorParam element -->

  <element name="descriptorParam"
substitutionGroup="descriptor:descriptorParam"/>


  <!-- extends descriptor element -->

  <complexType name="descriptorType">
    <complexContent>
      <extension base="descriptor:descriptorPrototype">
        <attributeGroup ref="layout:regionAttrs"/>
        <attributeGroup ref="timing:explicitDurAttrs"/>
        <attributeGroup ref="timing:freezeAttrs"/>
        <attributeGroup ref="keyNavigation:keyNavigationAttrs"/>
      </extension>
    </complexContent>
  </complexType>

  <element name="descriptor" type="profile:descriptorType"
substitutionGroup="descriptor:descriptor"/>

  <!-- extends descriptorBase element -->
  <complexType name="descriptorBaseType">
    <complexContent>
      <extension base="descriptor:descriptorBasePrototype">
        <choice minOccurs="1" maxOccurs="unbounded">
          <element ref="profile:importBase"/>
          <element ref="profile:descriptor"/>
          <element ref="profile:descriptorSwitch"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="descriptorBase" type="profile:descriptorBaseType"
substitutionGroup="descriptor:descriptorBase"/>
```

```
  <!-- ============================================================ -->
  <!-- Linking                                                      -->
  <!-- ============================================================ -->

  <!-- substitutes linkParam and bindParam elements -->
  <element name="linkParam" substitutionGroup="linking:linkParam"/>
  <element name="bindParam" substitutionGroup="linking:bindParam"/>

  <!-- extends bind element and link element, as a consequence-->

  <complexType name="bindType">
    <complexContent>
      <extension base="linking:bindPrototype">
        <attributeGroup ref="descriptor:descriptorAttrs"/>
      </extension>
    </complexContent>
  </complexType>

  <element name="bind" type="profile:bindType"
substitutionGroup="linking:bind"/>

  <!-- extends link element -->
  <complexType name="linkType">
    <complexContent>
      <extension base="linking:linkPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="link" type="profile:linkType"
substitutionGroup="linking:link"/>

  <!-- ============================================================ -->
  <!-- Connector                                                    -->
  <!-- ============================================================ -->
  <!-- extends connectorBase element -->

    <complexType name="connectorBaseType">
        <complexContent>
            <extension base="connectorBase:connectorBasePrototype">
              <choice minOccurs="0" maxOccurs="unbounded">
                 <element ref="profile:importBase"/>
                 <element ref="profile:causalConnector" />
              </choice>
            </extension>
        </complexContent>
    </complexType>

  <element name="connectorBase" type="profile:connectorBaseType"
substitutionGroup="connectorBase:connectorBase"/>

  <element name="causalConnector"
substitutionGroup="causalConnectorFunctionality:causalConnector"/>

  <element name="connectorParam"
substitutionGroup="causalConnectorFunctionality:connectorParam"/>

  <element name="simpleCondition"
substitutionGroup="causalConnectorFunctionality:simpleCondition"/>
```

```
  <element name="compoundCondition"
substitutionGroup="causalConnectorFunctionality:compoundCondition"/>

  <element name="simpleAction"
substitutionGroup="causalConnectorFunctionality:simpleAction"/>

  <element name="compoundAction"
substitutionGroup="causalConnectorFunctionality:compoundAction"/>

  <element name="assessmentStatement"
substitutionGroup="causalConnectorFunctionality:assessmentStatement"/>

  <element name="attributeAssessment"
substitutionGroup="causalConnectorFunctionality:attributeAssessment"/>

  <element name="valueAssessment"
substitutionGroup="causalConnectorFunctionality:valueAssessment"/>

  <element name="compoundStatement"
substitutionGroup="causalConnectorFunctionality:compoundStatement"/>

  <!-- ============================================================ -->
  <!-- TestRule                                                     -->
  <!-- ============================================================ -->
  <!-- extends rule element -->
  <complexType name="ruleType">
    <complexContent>
      <extension base="testRule:rulePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="rule" type="profile:ruleType"
substitutionGroup="testRule:rule"/>

  <!-- extends compositeRule element -->
  <complexType name="compositeRuleType">
    <complexContent>
      <extension base="testRule:compositeRulePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="compositeRule" type="profile:compositeRuleType"
substitutionGroup="testRule:compositeRule"/>

  <!-- extends ruleBase element -->
  <complexType name="ruleBaseType">
    <complexContent>
      <extension base="testRule:ruleBasePrototype">
        <choice minOccurs="1" maxOccurs="unbounded">
          <element ref="profile:importBase"/>
          <element ref="profile:rule"/>
          <element ref="profile:compositeRule"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
```

```
  <element name="ruleBase" type="profile:ruleBaseType"
substitutionGroup="testRule:ruleBase"/>

  <!-- ============================================================ -->
  <!-- TestRuleUse                                                    -->
  <!-- ============================================================ -->
  <!-- extends bindRule element -->
  <complexType name="bindRuleType">
    <complexContent>
      <extension base="testRuleUse:bindRulePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="bindRule" type="profile:bindRuleType"
substitutionGroup="testRuleUse:bindRule"/>

  <!-- ============================================================ -->
  <!-- ContentControl                                               -->
  <!-- ============================================================ -->
  <!-- extends switch element -->

  <!-- switch interface element groups -->
  <group name="switchInterfaceElementGroup">
    <choice>
      <element ref="profile:switchPort"/>
    </choice>
  </group>

  <!-- extends defaultComponent element -->
  <complexType name="defaultComponentType">
    <complexContent>
      <extension base="contentControl:defaultComponentPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="defaultComponent" type="profile:defaultComponentType"
substitutionGroup="contentControl:defaultComponent"/>

  <complexType name="switchType">
    <complexContent>
      <extension base="contentControl:switchPrototype">
          <choice minOccurs="0" maxOccurs="unbounded">
            <group ref="profile:switchInterfaceElementGroup"/>
            <element ref="profile:bindRule"/>
            <element ref="profile:switch"/>
            <element ref="profile:media"/>
            <element ref="profile:context"/>
          </choice>
        <attributeGroup ref="entityReuse:entityReuseAttrs"/>
      </extension>
    </complexContent>
  </complexType>

  <element name="switch" type="profile:switchType"
substitutionGroup="contentControl:switch"/>

  <!-- ============================================================ -->
  <!-- DescriptorControl                                            -->
```

```
  <!-- ============================================================ -->
  <!-- extends defaultDescriptor element -->
  <complexType name="defaultDescriptorType">
    <complexContent>
      <extension base="descriptorControl:defaultDescriptorPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="defaultDescriptor" type="profile:defaultDescriptorType"
substitutionGroup="descriptorControl:defaultDescriptor"/>

  <!-- extends descriptorSwitch element -->

  <complexType name="descriptorSwitchType">
    <complexContent>
      <extension base="descriptorControl:descriptorSwitchPrototype">
          <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="profile:descriptor"/>
            <element ref="profile:bindRule"/>
          </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="descriptorSwitch" type="profile:descriptorSwitchType"
substitutionGroup="descriptorControl:descriptorSwitch"/>

  <!-- ============================================================ -->
  <!-- Timing                                                       -->
  <!-- ============================================================ -->


  <!-- ============================================================ -->
  <!-- Import                                                       -->
  <!--============================================================ -->
  <complexType name="importBaseType">
    <complexContent>
      <extension base="import:importBasePrototype">
      </extension>
    </complexContent>
  </complexType>

  <complexType name="importNCLType">
    <complexContent>
      <extension base="import:importNCLPrototype">
      </extension>
    </complexContent>
  </complexType>

  <complexType name="importedDocumentBaseType">
    <complexContent>
      <extension base="import:importedDocumentBasePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="importBase" type="profile:importBaseType"
substitutionGroup="import:importBase"/>
```

```
  <element name="importNCL" type="profile:importNCLType"
substitutionGroup="import:importNCL"/>
  <element name="importedDocumentBase"
type="profile:importedDocumentBaseType"
substitutionGroup="import:importedDocumentBase"/>

  <!-- ========================================================== -->
  <!-- EntityReuse                                                -->
  <!-- ========================================================== -->

  <!-- ========================================================== -->
  <!-- ExtendedEntityReuse                                        -->
  <!-- ========================================================== -->
  <!-- ========================================================== -->
  <!-- KeyNavigation                                              -->
  <!-- ========================================================== -->

</schema>
```

6. Authoring an NCL 2.4 Document: An Example

This section presents a step-by-step specification of an NCL document, following the language profile specification of Section 5. The example comprises a main video (a track of the MTRX film with a dialog between two characters: M. and N.), with two advertisement videos synchronized with the main video. The first one is a battery advertisement synchronized with the moment that M. raises a battery. The second one is presented when a user interacts with an icon placed over M.' eyeglasses. In this last case, the main video is paused and resized during the advertisement presentation and a text form is presented at the end of the advertisement to offer user the opportunity to initiate a transaction to buy the eyeglasses. Depending on the selected language, the English or the Portuguese text form is presented to the user.

The specification begins declaring in the body element which media objects will take part of the document, as shown in Figure 6.1. A port element in the body gives the start point for the document presentation. This entry point is bind to the maestroTV element and to its default "whole content anchor" (since the maestroTV interface is not specified), indicating the image TV800.jpg as the starting point. TV800.jpg is only an image that simulates a 3:4 aspect ratio TV device, where the presentation will take place.

The settings element (media type equals to application/x-NCL-settings) groups all environment (context aware) variables. In NCL, any variable type can be specified. Every variable used in relationship specifications must be explicitly declared in the application/x-NCL-settings type media element. In the example, the variable "language" is explicitly declared as an attribute of the settings element.

The mtrx video definition has two anchors specifying video tracks used to synchronize the battery video presentation and the icon image presentation. As mentioned before, the icon will be placed over the mtrx video presentation in a region that matches the position of M.'s eyeglasses at this time. The mtrx video definition has also explicitly declared its positioning attributes, since they will be used when resizing the video. Again, every media attribute used in relationship specifications must be explicitly declared. Note also that all media objects reference descriptors used to control their presentation. Descriptor specifications are made in the document head, as will be discussed further on.

A context element (glassesPub) is used to group all media objects related to the glasses advertisement: the video, the icon for interaction and the text forms. The grouping can be very useful, for example, if one wants to repeat (reuse) the advertisement in another part of the film. It is also useful as a structuring facility. The context element has three port elements, exporting context internal element interfaces to be used in relationships declared by links.

A switch element allows the selection of the text form element to be presented, depending on which rule is satisfied. Rule specifications are made in the document head, as will be discussed further on. Similar to the context element, switches also have port elements exporting internal element interfaces to be used in relationships declared by links.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <ncl id="mtrxExample"
     xmlns="http://www.ncl.org.br/NCL3.0/profiles/NCL30BDTV.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

03  <head>


40   </head>

41  <body>
42    <port id="entryPoint" component="maestroTV"/>

43    <media id="maestroTV" type="image" descriptor="maestroDesc"
        src="example/TV800.jpg"/>
44    <media type="settings" id="settings">
45      <attribute id="language" name="language"/>
46    </media>

47    <media id="mtrx" type="video" descriptor="mtrxDesc"
        src="example/mtrx.mpg">
48      <area id="anchorBattery" begin="18s" end="19s"/>
49      <area id="anchorGlasses" begin="27s" end="32s" />
50      <attribute id="left" name="left" />
51      <attribute id="top" name="top" />
52      <attribute id="height" name="height" />
53      <attribute id="width" name="width" />
54    </media>
55    <media id="battery" type="video" descriptor="batteryDesc"
        src="example/battery.mpg"/>

56    <context id="glassesPub">
57      <port id="glassesIconInt" component="glassesIcon"/>
58      <port id="glassesInt" component="glasses"/>
59      <port id="glassesFormInt" component="form" interface="entryForm"/>

60      <media id="glassesIcon" type="video" descriptor="glassesIconDesc"
          src="example/glasses-blinks.avi"/>
61      <media id="glasses" type="video" descriptor="glassesDesc"
          src="example/glassesAd.mpg"/>

62      <switch id="form">
63        <switchPort id="entryForm">
64          <mapping component="ptForm"/>
65          <mapping component="enForm"/>
66        </switchPort>
67        <bindRule rule="rPt" component="ptForm"/>
68        <bindRule rule="rEn" component="enForm"/>
69        <media id="ptForm" type="text" descriptor="formDesc"
            src="example/ptForm.html"/>
70        <media id="enForm" type="text" descriptor="formDesc"
            src="example/enForm.html"/>
71      </switch>


76    </context>


136 </body>
137 </ncl>
```

Figure 6.1 – Component specification

The layout, declaring the positioning of each media object presentation, is then specified in the *head* element, as shown in Figure 6.2. One region representing the device (the simulated MaestroTV) is specified, whose background is set to black. Another region (TVRegion) specifies the exhibition area (MaestroTV screen). Five (sub) regions are then specified for the document media object placements.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <ncl id="mtrxExample"
     xmlns="http://www.ncl.org.br/NCL3.0/profiles/NCL30BDTV.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

03  <head>

04    <regionBase>
05      <region id="maestroRegion" title="Mtrx Example" top="0" left="0"
         height="640" width="810" zIndex="4">
06        <region id="TVRegion" top="35" left="5"  height="600" width="800"
           zIndex="3">
07          <region id="mtrxRegion" left="0%" top="0%" height="540"
             width="800" zIndex="2"/>
08          <region id="pubBatteryRegion" left="68%" top="68%" height="32%"
             width="32%" zIndex="1"/>
09          <region id="glassesIconRegion" left="12%" top="24%" height="39"
             width="68" zIndex="1"/>
10          <region id="pubGlassesRegion" left="53%" top="43%" height="37%"
             width="47%" zIndex="1"/>
11          <region id="formRegion" left="53%" top="43%" height="37%"
             width="47%" zIndex="1"/>
12        </region>
13      </region>
14    </regionBase>


40  </head>

41  <body>


136 </body>
137 </ncl>
```

Figure 6.2 – Layout specification

In Figure 5.3, descriptor elements bind media objects and their presentation regions, besides specifying other media presentation characteristics. Descriptors are grouped in descriptor bases. Similarly, presentation rules are grouped in rule bases.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <ncl id="mtrxExample"
     xmlns="http://www.ncl.org.br/NCL3.0/profiles/NCL30BDTV.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

03  <head>



14
15    <descriptorBase>
16       <descriptor id="maestroDesc" region="maestroRegion">
17          <descriptorParam name="background" value="black"/>
18       </descriptor>
```

```
19      <descriptor id="mtrxDesc" region="mtrxRegion">
20        <descriptorParam name="soundLevel" value="1"/>
21      </descriptor>
22      <descriptor id="batteryDesc" region="pubBatteryRegion">
23        <descriptorParam name="background" value="transparent"/>
24        <descriptorParam name="soundLevel" value="1"/>
25      </descriptor>
26      <descriptor id="glassesDesc" region="pubGlassesRegion">
27        <descriptorParam name="background" value="transparent"/>
28        <descriptorParam name="soundLevel" value="1"/>
29      </descriptor>
30      <descriptor id="formDesc" region="formRegion" explicitDur="3s"/>
31      <descriptor id="glassesIconDesc" region="glassesIconRegion"/>
32    </descriptorBase>

33    <ruleBase>
34       <rule id="rEn" var="language" op="eq" value="en" />
35       <rule id="rPt" var="language" op="eq" value="pt" />
36    </ruleBase>


40  </head>

41  <body>

136 </body>
137</ncl>
```

Figure 6.3 – Descriptor and presentation rule specification

Finally, synchronization relationships among document components are specified in the context element and in the body element, through their link child elements, as shown in Figure 6.4. Connectors referenced by links (xconnector attribute) are imported from external connector bases whose URI are specified in connectorBase elements declared as child elements of the head element. The imported connector base is presented in Appendix A.

In the figure, Link00 starts the mtrx video and sets the language variable to pt (Portuguese). Link01 starts the battery video when the mtrx video enters the track specified as anchorBattery. Link 02 starts the icon presentation when the mtrx video enters the track specified as anchorGlasses. Link03 stops the icon presentation when the track corresponding to the anchorGlasses finishes its presentation. Link04 specifies that the icon selection (by the yellow key of the TV remote control) resizes and pauses the mtrx video, stops the icon presentation, and starts the glasses publicity video. Link05 defines that when the eyeglasses publicity finishes, the mtrx video is resumed with its original size. Link07 (inside the context element) starts the text form presentation when the eyeglasses publicity video finishes and Link06 finishes TVMaestro image presentation when the mtrix video also finishes. This ends the document presentation.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <ncl id="mtrxExample"
    xmlns="http://www.ncl.org.br/NCL3.0/profiles/NCL30BDTV.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

03 <head>

37   <connectorBase>
38     <importBase alias="itv" documentURI="causalConnBase.ncl"/>
```

```
39    </connectorBase>

40  </head>

41  <body>
42    <port id="entryPoint" component="maestroTV"/>


56    <context id="glassesPub">


72      <link id="Link07" xconnector="itv#onEndStart">
73        <bind role="onEnd" component="glasses"/>
74        <bind role="start" component="form" interface="entryForm"/>
75      </link>

76    </context>

77    <link id="Link00" xconnector="itv#onBeginSetStart">
78      <bind role="onBegin" component="maestroTV"/>
79      <bind role="set" component="settings" interface="language">
80        <bindParam name="var" value="pt"/>
81      </bind>
82      <bind role="start" component="mtrx"/>
83    </link>

84    <link id="Link01" xconnector="itv#onBeginStart">
85      <bind role="onBegin" component="mtrx" interface="anchorBattery"/>
86      <bind role="start" component="battery"/>
87    </link>

88    <link id="Link02" xconnector="itv#onBeginStart">
89      <bind role="onBegin" component="mtrx" interface="anchorGlasses"/>
90      <bind role="start" component="glassesPub" interface="glassesIconInt"/>
91    </link>

92    <link id="Link03" xconnector="itv#onEndStop">
93      <bind role="onEnd" component="mtrx" interface="anchorGlasses"/>
94      <bind role="stop" component="glassesPub" interface="glassesIconInt"/>
95    </link>

96    <link id="Link04" xconnector="itv#onKeySelectionStopResizePauseStart">
97      <bind role="onKeySelection" component="glassesPub"
          interface="glassesIconInt">
98        <bindParam name="keyCode" value="YELLOW"/>
99      </bind>
100     <bind role="stop" component="glassesPub" interface="glassesIconInt"/>
101     <bind role="setWidth" component="mtrx" interface="width">
102       <bindParam name="width" value="424"/>
103     </bind>
104     <bind role="setHeight" component="mtrx" interface="height">
105       <bindParam name="height" value="286"/>
106     </bind>
107     <bind role="setLeft" component="mtrx" interface="left">
108       <bindParam name="left" value="0"/>
109     </bind>
110     <bind role="setTop" component="mtrx" interface="top" >
111       <bindParam name="top" value="20"/>
112     </bind>
113     <bind role="pause" component="mtrx"/>
114     <bind role="start" component="glassesPub" interface="glassesInt"/>
115   </link>

116   <link id="Link05" xconnector="itv#onEndResizeResume">
117     <bind role="onEnd" component="glassesPub" interface="glassesFormInt"/>
118     <bind role="setLeft" component="mtrx" interface="left">
119       <bindParam name="left" value="0"/>
```

```
120     </bind>
121     <bind role="setTop" component="mtrx" interface="top">
122        <bindParam name="top" value="0"/>
123     </bind>
124     <bind role="setWidth" component="mtrx" interface="width">
125        <bindParam name="width" value="800"/>
126     </bind>
127     <bind role="setHeight" component="mtrx" interface="height">
128        <bindParam name="height" value="540"/>
129     </bind>
130     <bind role="resume" component="mtrx"/>
131   </link>

132   <link id="Link06" xconnector="itv#onEndStop">
133     <bind role="onEnd" component="mtrx"/>
134     <bind role="stop" component="maestroTV"/>
135   </link>

136 </body>
137 </ncl>
```

Figure 6.4 – Relationship specification

It is worth to be noted the orders in which binds are declared in Link04 and Link05. Indeed, these orders must be obeyed in the connector definition and are very important. The region is altered step by step, that is attribute by attribute, by the formatter. If in any time of this process the region does not fit in its parent region an inconsistency is reached and the process fail, even if the initial and final region definitions are consistent. In order to avoid this problem, group of attributes should be used. In the case of Link04 and Link05, the attribute group named *bounds* is the most appropriate. When a formatter treats an attribute group it only tests the consistency at the end of the process.

The complete document specification, with the modifications in Link04, Link05 and mtrx attributes, is shown in Figure 6.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<ncl id="mtrxExample"
  xmlns="http://www.ncl.org.br/NCL3.0/profiles/NCL30BDTV.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <head>

   <regionBase>
     <region id="maestroRegion" title="Mtrx Example" top="0" left="0"
          height="640" width="810" zIndex="3"/>
       <region id="TVRegion" top="35" left="5" height="600" width="800"
          zIndex="3">
         <region id="mtrxRegion" left="0%" top="0%" height="540" width="800"
            zIndex="2"/>
         <region id="pubBatteryRegion" left="68%" top="68%" height="32%"
            width="32%" zIndex="1"/>
         <region id="glassesIconRegion" left="12%" top="24%" height="39"
            width="68" zIndex="1"/>
         <region id="pubGlassesRegion" left="53%" top="43%" height="37%"
            width="47%" zIndex="1"/>
         <region id="formRegion" left="53%" top="43%" height="37%" width="47%"
            zIndex="1"/>
       </region>
     </region>
   </regionBase>
```

```
  <descriptorBase>
     <descriptor id="maestroDesc" region="maestroRegion"/>
       <descriptorParam name="background" value="black"/>
     </descriptor>
     <descriptor id="mtrxDesc" region="mtrxRegion">
       <descriptorParam name="soundLevel" value="1"/>
     </descriptor>
     <descriptor id="batteryDesc" region="pubBatteryRegion">
       <descriptorParam name="soundLevel" value="1"/>
     </descriptor>
     <descriptor id="glassesDesc" region="pubGlassesRegion">
       <descriptorParam name="soundLevel" value="1"/>
     </descriptor>
     <descriptor id="formDesc" region="formRegion" explicitDur="3s"/>
     <descriptor id="glassesIconDesc" region="glassesIconRegion"/>
  </descriptorBase>

  <ruleBase>
     <rule id="rEn" var="language" op="eq" value="en" />
     <rule id="rPt" var="language" op="eq" value="pt" />
  </ruleBase>

  <connectorBase>
    <importBase alias="itv" documentURI="causalConnBase.ncl"/>
  </connectorBase>

</head>

<body>
 <port id="entryPoint" component="maestroTV"/>

 <media id="maestroTV" type="image" descriptor="maestroDesc"
    src="example/TV800.jpg"/>
 <media type="settings" id="settings" >
   <attribute id="language" name="language" type="xsi:string"/>
 </media>

 <media id="mtrx" type="video" descriptor="mtrxDesc" src="example/mtrx.mpg">
   <area id="anchorBattery" begin="18s" end="19s"/>
   <area id="anchorGlasses" begin="27s" end="32s" />
   <attribute id="bounds" name="bounds" type="xsi:string"/>
 </media>

 <media id="battery" type="video" descriptor="batteryDesc"
    src="example/battery.mpg"/>

 <context id="glassesPub">
   <port id="glassesIconInt" component="glassesIcon"/>
   <port id="glassesInt" component="glasses"/>
   <port id="glassesFormInt" component="form" interface="entryForm"/>

   <media id="glassesIcon" type="video" descriptor="glassesIconDesc"
    src="example/glasses-blinks.avi"/>
   <media id="glasses" type="video" descriptor="glassesDesc"
    src="example/glassesAd.mpg"/>

   <switch id="form">
     <switchPort id="entryForm">
        <mapping component="ptForm"/>
        <mapping component="enForm"/>
     </switchPort>
     <bindRule rule="rPt" component="ptForm"/>
     <bindRule rule="rEn" component="enForm"/>
     <media id="ptForm" type="text" descriptor="formDesc"
       src="example/ptForm.html" />
     <media id="enForm" type="text" descriptor="formDesc"
       src="example/enForm.html" />
```

```
    </switch>

    <link id="Link07" xconnector="itv#onEndStart">
      <bind role="onEnd" component="glasses"/>
      <bind role="start" component="form" interface="entryForm"/>
    </link>

  </context>

  <link id="Link00" xconnector="itv#onBeginSetStart">
     <bind role="onBegin" component="maestroTV"/>
     <bind role="set" component="setting" interface="language">
        <bindParam name="var" value="pt"/>
     </bind>
     <bind role="start" component="mtrx"/>
  </link>

  <link id="Link01" xconnector="itv#onBeginStart">
     <bind role="onBegin" component="mtrx" interface="anchorBattery"/>
     <bind role="start" component="battery"/>
  </link>

  <link id="Link02" xconnector="itv#onBeginStart">
     <bind role="onBegin" component="mtrx" interface="anchorGlasses"/>
     <bind role="start" component="glassesPub" interface="glassesIconInt"/>
  </link>

  <link id="Link03" xconnector="itv#onEndStop">
     <bind role="onEnd" component="mtrx" interface="anchorGlasses"/>
     <bind role="stop" component="glassesPub" interface="glassesIconInt"/>
  </link>

  <link id="Link04" xconnector="itv#onKeySelectionStopSetPauseStart">
    <bind role="onKeySelection" component="glassesPub"
     interface="glassesIconInt">
       <bindParam name="keyCode" value="YELLOW"/>
    </bind>
    <bind role="stop" component="glassesPub" interface="glassesIconInt"/>
    <bind role="set" component="mtrx" interface="bounds">
       <bindParam name="bounds" value="0,20,424,286"/>
    </bind>
    <bind role="pause" component="mtrx"/>
    <bind role="start" component="glassesPub" interface="glassesInt"/>
  </link>

  <link id="Link05" xconnector="itv#onEndSetResume">
    <bind role="onEnd" component="glassesPub" interface="glassesFormInt"/>
    <bind role="set" component="mtrx" interface="bounds">
       <bindParam name="var" value="0,0,800,540"/>
    </bind>
    <bind role="resume" component="mtrx"/>
  </link>

  <link id="Link06" xconnector="itv#onEndStop">
    <bind role="onEnd" component="mtrx"/>
    <bind role="stop" component="maestroTV"/>
  </link>

 </body>
</ncl>
```

Figure 6.5 – Complete document specification

7. Final Remarks

In order to offer a scalable hypermedia model, with characteristics that can be progressively incorporated in hypermedia system implementations, NCM was divided in several parts. This technical report deals with the declarative language for authoring documents based on NCM, which comprises Part 8: NCL (Nested Context Language) Digital TV Profiles.

Acknowledgements

Many people have contributed to the definition of the NCL Digital TV profiles. Chief among these are Romualdo Rezende Costa, Carlos Salles Soares Netto, Marcio Ferreira Moreno e Simone Junqueira Barbosa.

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.

[CSS98] Cascading Style Sheets, level 2, Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs. W3C Recommendation 12 May 1998,
Available at http://www.w3.org/TR/REC-CSS2

[IETF98] Berners-Lee T., Fielding R., Masinter L. Uniform Resource Identifiers (URI): Generic Syntax, **RFC 2046, IETF**, August 1998.

[ISDTV-T N06 V02] Volume 2 of ISDTV-T Standard 06. *ISDTV-T Fórum*. December 2006.

[ISDTV-T N06 V04] Volume 4 of ISDTV-T Standard 06. *ISDTV-T Fórum*. December 2006.

[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.

[SMIL05] SMIL 2.1 - *Synchronized Multimedia Integration Language – SMIL 2.1 Specification,* **W3C Recommendation**, December 2005.

[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.

[W3C99] Namespaces in XML, **W3C Recommendation**, January 1999.

[W3C02] XHTML™ 1.0 2º Edition - *Extensible HyperText Markup Language*, **W3C Recommendation**, August 2002.

[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.

Appendix A – Connector Base

This appendix presents an extensive Connector Base that can be imported by any NCL 3.0 document.

```
<!--

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/connectorBases/causalConnBase.ncl
Author: TeleMidia Laboratory
Revision: 19/09/2006

-->

<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="causalConnBase"
xmlns="http://www.ncl.org.br/NCL3.0/CausalConnectorProfile">

<head>
<connectorBase>

<!-- OnBegin -->

<causalConnector id="onBeginStart">
  <simpleCondition role="onBegin"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onBeginStop">
  <simpleCondition role="onBegin"/>
  <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onBeginPause">
  <simpleCondition role="onBegin"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onBeginResume">
  <simpleCondition role="onBegin"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onBeginSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<!-- OnEnd -->

<causalConnector id="onEndStart">
  <simpleCondition role="onEnd"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onEndStop">
  <simpleCondition role="onEnd"/>
  <simpleAction role="stop"/>
</causalConnector>
```

```
<causalConnector id="onEndPause">
  <simpleCondition role="onEnd"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onEndResume">
  <simpleCondition role="onEnd"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onEndSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>


<!-- OnMouseSelection -->

<causalConnector id="onSelectionStart">
  <simpleCondition role="onSelection"/>
  <simpleAction role="start" />
</causalConnector>

<causalConnector id="onSelectionStop">
  <simpleCondition role="onSelection"/>
  <simpleAction role="stop" />
</causalConnector>

<causalConnector id="onSelectionPause">
  <simpleCondition role="onSelection"/>
  <simpleAction role="pause" />
</causalConnector>

<causalConnector id="onSelectionResume">
  <simpleCondition role="onSelection"/>
  <simpleAction role="resume" />
</causalConnector>

<causalConnector id="onSelectionSetVar">
  <connectorParam name="var" />
  <simpleCondition role="onSelection"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<!-- OnKeySelection -->

<causalConnector id="onKeySelectionStart">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onKeySelectionStop">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onKeySelectionPause">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onKeySelectionResume">
  <connectorParam name="keyCode"/>
```

```
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onKeySelectionSetVar">
  <connectorParam name="keyCode"/>
  <connectorParam name="var"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<!-- OnBegin multiple actions -->

<causalConnector id="onBeginStartStop">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartPause">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartResume">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopStart">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopPause">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopResume">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="resume"/>
```

```
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnEnd multiple actions -->

<causalConnector id="onEndStartStop">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStartPause">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>
```

```
<causalConnector id="onEndStartResume">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopStart">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopPause">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopResume">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="stet" value="$var"/>
     <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>
```

```
<causalConnector id="onEndSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnMouseSelection multiple actions -->

<causalConnector id="onSelectionStartStop">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartPause">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartResume">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopStart">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopPause">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
```

```
      <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopResume">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
      <simpleAction role="stop"/>
      <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
      <simpleAction role="stop"/>
      <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
      <simpleAction role="set" value="$var"/>
      <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
      <simpleAction role="stet" value="$var"/>
      <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
      <simpleAction role="set" value="$var"/>
      <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
      <simpleAction role="set" value="$var"/>
      <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnKeySelection multiple actions -->

<causalConnector id="onKeySelectionStartStop">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
      <simpleAction role="start"/>
      <simpleAction role="stop"/>
  </compoundAction>
```

```
</causalConnector>

<causalConnector id="onKeySelectionStartPause">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStartResume">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStartSet">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopStart">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopPause">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopResume">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopSet">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>
```

```
<causalConnector id="onKeySelectionSetStart">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetStop">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetPause">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetResume">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!--Miscellaneous-->

<causalConnector id="onKeySelectionStopResizePauseStart">
  <connectorParam name="width"/>
  <connectorParam name="height"/>
  <connectorParam name="left"/>
  <connectorParam name="top"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="setWidth" value="$width"/>
     <simpleAction role="setHeight" value="$height"/>
     <simpleAction role="setLeft" value="$left"/>
     <simpleAction role="setTop" value="$top"/>
     <simpleAction role="pause"/>
     <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndResizeResume">
  <connectorParam name="left"/>
  <connectorParam name="top"/>
  <connectorParam name="width"/>
  <connectorParam name="height"/>
  <simpleCondition role="onEnd"/>
```

```
  <compoundAction operator="seq">
     <simpleAction role="setLeft" value="$left"/>
     <simpleAction role="setTop" value="$top"/>
     <simpleAction role="setWidth" value="$width"/>
     <simpleAction role="setHeight" value="$height"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopSetPauseStart">
  <connectorParam name="bounds"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="stop"/>
     <simpleAction role="set" value="$bounds"/>
     <simpleAction role="pause"/>
     <simpleAction role="start"/>
  </compoundAction>
</causalConnector>
```

Appendix B - Media Objects in NCL Presentations

The presentation of an NCL document requires the synchronization control of several media objects specified through the <media> element. For each media object, a media player should be loaded to control the object and its NCL events. This appendix offers guidelines to the expected behavior of these players when controlling the presentation of NCL media objects.

In order to favor the incorporation of third-party media players an NCL formatter implementation must be disconnected from media players presentations.

Figure B.1 suggests a modular implementation, where media players are plug-in modules of the presentation engine. Since it may be interesting to use already existent media players that may have proprietary interfaces that are not compatible with the one required by the NCL formatter, it will be necessary to develop modules to make the necessary adaptations. In this case, the media player will be constituted of an adapter besides the player itself.

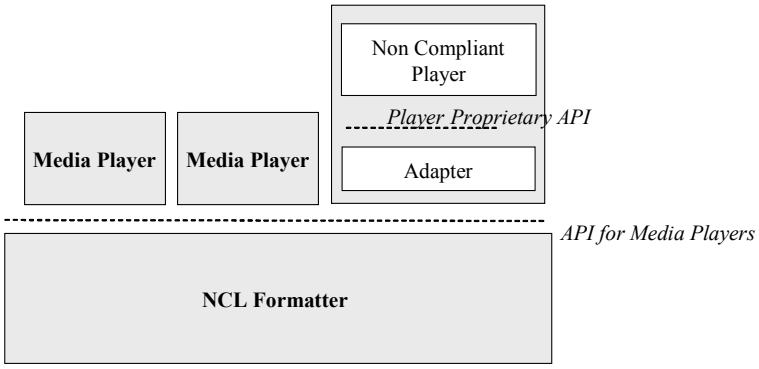

Figure B.1 – APIs for Integrating Media Players with an NCL formatter

As the NCL formatter architecture and implementation is a choice of each designer, this appendix does not intend to standardize the syntax of the formatter API. The goal is just to define the expected behavior of a media player when controlling objects that make part of an NCL document.

B.1. Expected Behavior of Media Players

A media player (or its adapter) must be able to receive presentation commands, to control the events' state machines of the media object, and answer queries from the formatter. This section describes how a media player must behave for each expected instruction issued by the formatter.

B.1.1. Start instruction

Before sending a *start* instruction, the formatter should find the more appropriate media player to be called based on the content type to be exhibited. For this sake, the formatter takes into consideration the *player* attribute of the <descriptor> element associated with the media object to be exhibit. If this attribute is not specified, the formatter must take into

account the *type* attribute of the <media> element. If this attribute is not specified either, the formatter must consider the file extension specified in the *src* attribute of the <media> element.

The *start* instruction issued by a formatter must inform to the media player the following parameters: the media object to be controlled, its associated descriptor, a list of events (presentation, selection or attribution) that need to be monitored by the media player, the presentation event that needs to be started (called main-event), an optional offset-time and an optional delay-time.

The media object must be derived from a <media> element, whose *src* attribute must be used by the media player to locate the content and start its presentation. If the content cannot be located, or if the media player does not know how to handle the content type, the media player must finish the starting operation without performing any action.

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

The list of events to be monitored by a media player must also be computed by the formatter, taking into account the NCL document specification. It must check all links where the media object and the resultant descriptor participate. When computing the events to be monitored, the formatter must 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.

The offset-time parameter is optional, it has "zero" as its default value, and it is meaningful only for continuous media or static media with explicit duration. In this case, this parameter defines a time offset from the beginning (beginning-time) of the main-event, from which the presentation of the main-event must be immediately started (i.e., it commands the player to jump to the beginning-time + offset-time). Obviously, the offset-time value must be lesser than the main-event duration. If the offset-time is greater than zero, the media player must put the main-event in the *occurring* state, but the event *starts* transition must not be notified. If the offset-time is zero, the media player must 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 must be put in the *occurring* state, but their *starts* transitions must not be notified (links that depend on this transition must 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) must have their *occurrences* attribute incremented but the *starts* and *stops* transitions must not be notified. Monitored events that have beginning-times before the start time (beginning-time + offset-time) and end time after the start time must be put in the *occurring* state, but the corresponding *starts* transition must 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 must only be considered if the offset-time parameter is equal to "zero".

If a media player receives a *start* instruction for an object already being presented it must ignore the instruction and keep on controlling the ongoing presentation.

B.1.2. Stop instruction

The *stop* instruction only needs to identify a media object already being controlled[14]. If the object is not being presented (none of the events in the object list of events is in the *occurring* or *paused* state) and the media player is not waiting due to a delayed *start* instruction, the *stop* instruction must be ignored. If the object is being presented, the main-event (the event passed as a parameter when the media object was started) and all monitored events in the *occurring* or in the *paused* state with end time equal or previous to the main-event end time must transit to the *sleeping* state, and their *stops* transitions must be notified. Monitored events in the *occurring* or in the *paused* state with end time posterior to the main-event end time must be put in the *sleeping* state, but their *stops* transitions must not be notified and their *occurrences* attribute must not be incremented. The object content presentation must stop to be played. If the *repetitions* event attribute is greater than zero, it must be decremented by one and the main-event presentation must restart after the repeat delay time (the repeat delay must having been passed to the media player as the start delay parameter). If the media object is waiting to be presented after a delayed *start* instruction and a *stop* instruction is issued, the previous *start* instruction must be removed.

B.1.3. Abort instruction

The *abort* instruction only needs to identify a media object already being controlled. If the object is not being presented and is not waiting to be presented after a delayed *start* instruction, the *abort* instruction must be ignored. If the object is being presented, the main-event and all monitored events in the *occurring* or in the *paused* state must transit to the *sleeping* state, and their *aborts* transitions must be notified. Any content presentation must stop. If the *repetitions* event attribute is greater than zero, it must be set to zero and the media object presentation must not restart. If the media object is waiting to be presented after a delayed *start* instruction and an *abort* instruction is issued, the previous *start* instruction must be removed.

---

[14] Identify the media object means identify the <media> element, the corresponding descriptor and the media-object perspective.

B.1.4. Pause instruction

The *pause* instruction only needs to identify a media object already being controlled. If the object is not being presented (the main-event, passed as a parameter when the media object was started, is not in the *occurring* state) and the media player is not waiting for the start delay, the instruction must be ignored. If the object is being presented, the main-event and all monitored events in the *occurring* state must transit to the *paused* state and their *pauses* transitions must be notified. The object presentation must be paused and the pause elapsed time must not be considered as part of the object duration. As an example, if an object has an explicit duration of 30 seconds and, after 25 seconds it is paused, even if the object stays paused for 5 minutes, after resuming the object main-event must stay occurring for 5 seconds. If the main-event is still not occurring because the media player is waiting for the start delay, the media object must wait for a resume instruction to continue waiting for the remaining start delay.

B.1.5. Resume instruction

The *resume* instruction only needs to identify a media object already being controlled. If the object is not paused (the main-event, passed as a parameter when the media object was started, is not in the *paused* state) or the media player is not paused (waiting for the start delay), the instruction must be ignored. If the media player is paused waiting for the start delay, it must resume the wait from the instant it was paused. If the main-event is in the *paused* state, the main-event and all monitored events in the *paused* state must be put in the *occurring* state and their *resumes* transitions must be notified.

B.1.6. Set instruction

The *set* instruction may be applied to an object independent from the fact it is being presented or not (in this case, although the object is not being presented, its media player must be already instantiated). In the first case, the *set* instruction needs to identify the media object being controlled, a monitored attribution event and a value to be assigned to the attribute wrapped by the event. In the second case, the instruction must also identify the <descriptor> element that will be used when presenting the object (as it is done for the *start* instruction). When setting a value to the attribute, the media player must transit the event state machine to the *occurring* and again to the *sleeping* state, generating the *starts* transition and afterwards the *stops* transition, and also must change the attribution event current value.

For every monitored attribution event, if the media player changes by itself the corresponding attribute value, it must also proceed as if it had received an external *set* instruction.

B.1.7. AddEvent instruction

The *addEvent* instruction is issued in the case of live editing the NCL document (see report Part 9: NCL Live Editing). The instruction needs to identify a media object already being controlled and a new event that must be included to be monitored. All rules applied to the intersection of monitored events with the main-event must be applied to the new event. If

the new event start time is previous to the object current time and the new event end time is posterior to the object current time, the new event must be put in the same state of the main-event (*occurring* or *paused*), without notifying the corresponding transition.

B.1.8. RemoveEvent instruction

The *removeEvent* instruction is also issued in the case of live editing the NCL document (see report Part 9: NCL Live Editing). The instruction needs to identify a media object already being controlled and a monitored event that should be no more controlled. The event state must be put in the *sleeping* state without generating any transition.

B.1.9. Natural end of a presentation

Main-events of an object, with an explicit or an intrinsic duration, normally end their presentations naturally, without needing external instructions. In this case, the media player must transit the main-event to the *sleeping* state and notify the *stops* transition. The same must be done for monitored events in the *occurring* state with the same end time of the main-event or with unknown end time. Events in the *occurring* state with end time posterior to the main-event end time must be put in the sleeping state but without generating the *stops* transition and without incrementing the *occurrences* attribute. It is important to remark that if the main-event corresponds to an object internal temporal anchor, when this anchor presentation finish, the whole media object presentation must finish.

B.2. Expected Behavior of Media Players after Instructions Applied to Composite Objects

A <simpleCondition> or <simpleAction> with *eventType* attribute value equal to "presentation" can be bound by a link to a composite node (represented by a <context> or <body> element) as a whole (i.e. without an interface being informed). As usual, the event state machine of the presentation event defined on the composite node must be controlled as specified in Section 3.2.7. Analogously, an <attributeAssessment> with *eventType* attribute value equal to "presentation" and *attributeType* equal to "state", "occurrences" or "repetitions" can be bound by a link to a composite node (represented by a <context> or <body> element) as a whole, and the attribute value should come from the event state machine of the presentation event defined on the composite node.

However, if a <simpleAction> with *eventType* attribute value equal to "presentation" is bound by a link to a composite node (represented by a <context> or <body> element) as a whole (i.e. without an interface being informed), the instruction must be also reflected to the event state machines of the composite child nodes, as explained in the following subsections.

B.2.1. Starting a composite presentation

If a <context> or <body> element participates on an action role whose action type is "start", when this action is fired, the *start* instruction must also be applied to all presentation events mapped by the <context> or <body> element's ports.

If the author wants to start the presentation using a specific port, it must in addition indicate the <port> id as the <bind> *interface* value.

B.2.2. Stopping a context presentation

If a <context> or <body> element participates on an action role whose action type is "stop", when this action is fired, the *stop* instruction must also be applied to all presentation events of the composite child nodes. For child media objects it is necessary to apply the *stop* instruction only to their main events.

If the composite contains links being evaluated (or with their evaluation paused), the evaluations must be suspended and no action must be fired.

B.2.3. Aborting a context presentation

If a <context> or <body> element participates on an action role whose action type is "abort", when this action is fired, the *abort* instruction must also be applied to all presentation events of the composite child nodes. For child media objects it is necessary to apply the *abort* instruction only to their main events.

If the composite contains links being evaluated (or with their evaluation paused), the evaluations must be suspended and no action must be fired.

B.2.4. Pausing a context presentation

If a <context> or <body> element participates on an action role whose action type is "pause", when this action is fired, the *pause* instruction must also be applied to all presentation events of the composite child nodes that are in the occurring state. For child media objects it is necessary to apply the *pause* instruction only to their main events.

If the composite contains links being evaluated, all evaluations must be suspended until a resume, stop or abort action is issued.

If the composite contains child nodes with presentation events already in the paused state when the pause action is issued, these nodes must be identified because if the composite receives a resume instruction, these events must not be resumed.

B.2.5. Resuming a context presentation

If a <context> or <body> element participates on an action role whose action type is "resume", when this action is fired, the *resume* instruction must also be applied to all presentation events of the composite child nodes that are in the paused state, except those

that were already paused before the composite has been paused. For child media objects it is necessary to apply the *resume* instruction only to their main events.

If the composite contains links with paused evaluations, they must be resumed.

If the composite contains child nodes with presentation events already in the paused state, these nodes must not be resumed.

B.3. Relation between the presentation-event state machine of a node and the presentation-event state machine of its parent-composite node

Whenever the presentation event of a node (media or composite) goes to the occurring state, the presentation event of the composite node that contains the node must also enter in the occurring state.

When all child nodes of a composite node have their presentation events in the sleeping state, the presentation event of the composite node must also be in the sleeping state.

Composite nodes do not need to infer *pauses* and *aborts* transitions from their child. These transitions in presentation events of composite nodes must occur only when instructions are applied directly to the composite node presentation event (Section B.2).

B.4. Procedural Objects in NCL Documents

Procedural objects can be inserted into NCL documents mainly to bring additional computational capabilities to declarative documents. The way to add procedural objects into NCL documents is defining a <media> element, whose content (specified by the *src* attribute) is the procedural code to be executed. EDTV and BDTV profiles of NCL 3.0 allows two media types to be associated with the <media> element: application/x-NCLua, for LUA procedural codes (extension file .lua); and application/x-NCLet, for Java (XLET) procedural codes (extension file .xlet, .xlt, or .class).

Authors may define NCL links to start, stop, pause or resume the execution of a procedural code as they do for usual presentation contents discussed in the previous sections. A procedural player (the language engine) must interface the procedural execution environment with the NCL formatter.

Analogous to conventional media content, procedural players must control the state machine of the events associated with the NCL procedural node (NCLua or NCLet). As an example, if the code finishes its execution, the player must generate the stops transition in the main-event presentation state machine corresponding to the procedural execution.

NCL allows procedural code to be synchronized with other NCL objects (procedural or not). A <media> element containing a procedural code may also define anchors (through <area> elements) and attributes (through <attribute> elements).

The <area> element must define only the *id* attribute. Procedural code may register as listener for these <area> elements. If external links start, stop, pause or resume the anchor presentation, callbacks in the procedural code must be triggered. The way these callbacks

are defined is responsibility of each procedural code associated with the NCL procedural object. On the other hand, the procedural code may command the start, stop, pause or resume of these anchors through an API offered by the procedural language. These transitions can be used as conditions of NCL links to trigger actions on other NCL objects of the same document. Thus, a two-way synchronization can be established between the procedural code and the remainder of the NCL document.

The other way a procedural code can be synchronized with other NCL objects is through NCL <attribute> elements. An <attribute> element defined as a child of a <media> element representing a procedural code can be mapped to a code function (or method) or to a code attribute. When it is mapped to a code function, a link action "set" applied to the attribute must cause the function execution, with the set values interpreted as parameters passed to the function. The attribute *name* of the <attribute> element must be used to identify the procedural code function. When the <attribute> element is mapped to a procedural code attribute the action "set" must assign the value to the attribute.

An <attribute> element defined as a child of a <media> element representing a procedural code can also be associated with an NCL link assessment role. In this case, the NCL formatter must query the attribute value in order to evaluate the link expression. If the <attribute> element is mapped to a code attribute, the code attribute value must be returned by the procedural player to the NCL formatter. If the <attribute> element is mapped to a code function, the function must be called and its output value must be returned by the procedural player to the NCL formatter.

Procedural languages should also offer an API that allows procedural code to query any pre-defined or dynamic attributes' values of the NCL settings node (<media> element of "application/x-NCL-settings" type).