



PUC

ISSN 0103-9741

Monografias em Ciência da Computação
nº 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

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 relationships 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 da PUC-Rio – Todos os direitos reservados

Impresso no Brasil

As informações contidas neste documento são de propriedade do Laboratório TeleMídia (PUC-Rio), sendo proibida a sua divulgação, reprodução ou armazenamento em base de dados ou sistema de recuperação sem permissão prévia e por escrito do Laboratório TeleMídia (PUC-Rio). As informações estão sujeitas a alterações sem notificação prévia.

Os nomes de produtos, serviços ou tecnologias eventualmente mencionadas neste documento são marcas registradas dos respectivos detentores.

Figuras apresentadas, quando obtidas de outros documentos, são sempre referenciadas e são de propriedade dos respectivos autores ou editoras referenciados.

Fazer cópias de qualquer parte deste documento para qualquer finalidade, além do uso pessoal, constitui violação das leis internacionais de direitos autorais.

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

1. Introduction	8
2. NCL Historical Evolution.....	10
3. The NCL 3.0 Modularization	13
3.1. Modular Languages and Language Profiles	13
3.1.1. Identifiers for NCL 3.0 Module and Language Profiles	15
3.1.2. NCL Version Information	17
3.2. NCL Modules: Attributes and Elements of the NCL 3.0 EnhancedDTV Profile ..	17
3.2.1. Structure Functionality	17
3.2.2. Layout Functionality	18
3.2.3. Components Functionality	21
3.2.4. Interfaces Functionality	29
3.2.5. Presentation Specification Functionality.....	46
3.2.6. Linking Functionality	48
3.2.7. Connectors Functionality	49
3.2.8. Presentation Control Functionality	58
3.2.9. Timing Functionality.....	61
3.2.10. Reuse Functionality	61
3.2.11. Navigational Key Functionality	64
3.2.12. Animation Functionality	66
3.2.13. The Transition Effects functionality.....	66
3.2.14. The Metainformation Functionality	69
3.3. Attributes and Elements of the NCL 3.0 BasicDTV Profile	70
4. NCL 3.0 Module Schemas	76
Structure Module: NCL30Structure.xsd	76

Layout Module: NCL30Layout.xsd.....	77
Media Module: NCL30Media.xsd.....	78
Context Module: NCL30Context.xsd	79
MediaContentAnchor Module: NCL30MediaContentAnchor.xsd	80
CompositeNodeInterface Module: NC30CompositeNodeInterface.xsd.....	82
PropertyAnchor Module: NCL30PropertyAnchor.xsd	83
SwitchInterface Module: NCL30SwitchInterface.xsd.....	84
Descriptor Module: NCL30Descriptor.xsd	85
Linking Module: NCL30Linking.xsd	87
ConnectorCommonPart Module: NCL30ConnectorCommonPart.xsd	88
ConnectorAssessmentExpression Module:	
NCL30ConnectorAssessmentExpression.xsd	89
ConnectorCausalExpression Module: NCL30ConnectorCausalExpression.xsd	91
CausalConnector Module: NCL30CausalConnector.xsd.....	94
ConnectorBase Module: NCL30ConnectorBase.xsd.....	95
NCL30CausalConnectorFunctionality.xsd	96
TestRule Module: NCL30TestRule.xsd.....	98
TestRuleUse Module: NCL30TestRuleUse.xsd.....	99
ContentControl Module: NCL30ContentControl.xsd.....	100
DescriptorControl Module: NCL30DescriptorControl.xsd.....	101
Timing Module: NCL30Timing.xsd	102
Import Module: NCL30Import.xsd.....	103
EntityReuse Module: NCL30EntityReuse.xsd.....	104
ExtendedEntityReuse Module: NCL30ExtendedEntityReuse.xsd	105
KeyNavigation Module: NCL30KeyNavigation.xsd.....	106
TransitionBase Module: NCL30TransitionBase.xsd	107
Animation Module: NCL30Animation.xsd.....	108

Transition module: NCL30Transition.xsd	109
Metainformation module: NCL30Metainformation.xsd.....	113
5. NCL 3.0 Language Profiles for Digital TV.....	114
5.1. The Schema of the NCL 3.0 Enhanced DTV Profile.....	115
5.2. The Schema of the NCL 3.0 Basic DTV Profile	127
6. Authoring an NCL 3.0 Document: An Example	138
7. Final Remarks	146
References	147
Appendix A – Connector Base	148
Appendix B - Media Objects in NCL Presentations.....	176
B.1. Expected Behavior of Basic Media Players.....	176
B.1.1. Start instruction for presentation events.....	177
B.1.2. Stop instruction for presentation events.....	178
B.1.3. Abort instruction for presentation events.....	179
B.1.4. Pause instruction for presentation events.....	179
B.1.5. Resume instruction for presentation events	180
B.1.6. Start instruction for attribution events	180
B.1.7. stop, abort, pause and resume instructions for attribution events.....	180
B.1.8. AddEvent instruction	181
B.1.9. RemoveEvent instruction	181
B.1.10. Natural end of a presentation.....	181
B.2. Expected Behavior of Media Players after Instructions Applied to Composite Objects.....	181
B.2.1. Starting a context presentation	182
B.2.2. Stopping a context presentation.....	182
B.2.3. Aborting a context presentation.....	182
B.2.4. Pausing a context presentation	182

B.2.5. Resuming a context presentation.....	183
B.3. Relation between the presentation-event state machine of a node and the presentation-event state machine of its parent-composite node	183
B.4. Expected behavior of imperative media players in NCL applications	184
B.5. Expected behavior of hypermedia players in NCL applications.....	185

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 with leading zeros suppressed. This report deals with the NCL version 3.0 (NCL 3.0).

From version 3.0 on, new NCL versions shall be released in accordance to the following versioning policy. If formatters¹ that conform to older versions can still receive a document based on the revised specification, due to error corrections, operational reasons, or the addition of a new concise syntax notation (“syntax sugar”) that can be translated at compile time to the old notation, the new version of NCL shall be released with the minor number updated. If receivers that conform to older versions cannot receive a document based on the revised specifications, the major number shall be updated.

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

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

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.

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 for 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 mappings for composite nodes, satisfying the document compositionality property;
- definition of hypermedia composite-node templates, allowing for 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 these new features.

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

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

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

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

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

- Part 1 – NCM Core
concerned with the main model entities, which should be present in all NCM implementations².
- Part 2 – NCM Virtual Entities
concerned mainly with the definition of virtual anchors, nodes and links.
- Part 3 – NCM Version Control
concerned with model entities and attributes to support versioning.
- Part 4 – NCM Cooperative Work
concerned with model entities and attributes to support cooperative document handling.

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

- Part 5 – NCL (Nested Context Language) Full Profile
concerned with the definition of an XML application language for authoring and exchanging NCM-based documents, using all NCL modules, including those for the definition and use of templates, and also the definition of constraint connectors, composite-connectors, temporal cost functions, transition effects and metainformation characterization.
- Part 6 – NCL (Nested Context Language) XConnector Profile Family
concerned with the definition of an XML application language for authoring connector bases. One profile is defined for authoring causal connectors, another one for authoring causal and constraint connectors, and a third one for authoring both simple and composite connectors.
- Part 7 – Composite Node Templates
concerned with the definition of the NCL Composite-Node Template functionality, and with the definition of an XML application language (TAL – Template Authoring Language or XTemplate) for authoring template bases.
- Part 8 – NCL (Nested Context Language) Digital TV Profiles (this document)
concerned with the definition of an XML application language for authoring documents

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

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

- Part 9 – NCL Live Editing Commands
concerned with editing commands used for live authoring applications based on NCL.
- Part 10 – Imperative Objects in NCL: The NCLua Scripting Language
concerned with the definition of objects that contain imperative code and how these objects may be related with other objects in an NCL application.
- Part 11 – Declarative Hypermedia Objects in NCL: Nesting Objects with NCL Code in NCL Documents
concerned with the definition of hypermedia objects that contain declarative code (including nested objects with NCL code) and how these objects may be related with other objects in an NCL application.
- Part 12 – Support to Multiple Exhibition Devices
concerned with the use of multiple devices for simultaneously presenting an NCL document.

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

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 profile. 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 a specific module are associated with the same namespace [W3C99].

A *language profile* is a combination of modules. Modules are *atomic*, i.e. they shall not 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 shall comply.

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

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

Other language profiles may be specified as subsets of the larger one, or to incorporate a combination of modules associated with different namespaces. 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 broadcast and broadband TV environment with its multiple possible presentation devices: TV set, mobile devices, etc..

The main purpose of a 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, may ignore all other (unknown) elements and attributes.

NCL 3.0 is partitioned into fifteen functional areas, which are partitioned again into modules. From the fifteen functional areas, fourteen are used to define the Enhanced DTV and the Basic DTV profiles. Two functional areas have modules with the same semantics

defined by SMIL 2.0 [SMIL05]. The fourteen 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. PropertyAnchor 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. CausalConnector Module
 - e. CausalConnectorFunctionality Module
 - f. 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. Animation

- a. Animation Module
- 13. Transition Effects
 - a. TransitionBase Module
 - b. Transition Module
- 14. 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 may 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 identifiers 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.

Modules may 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).

Table 3.1 – The NCL 3.0 Module Identifiers

Animation	http://www.ncl.org.br/NCL3.0/Animation
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
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
PropertyAnchor	http://www.ncl.org.br/NCL3.0/PropertyAnchor
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
Transition	http://www.ncl.org.br/NCL3.0/Transition
Metainformation	http://www.ncl.org.br/NCL3.0/MetaInformation

Three SMIL modules [SMIL 2.1 Specification, 2005] were used as the basis for the NCL Transition module and the NCL Metainformation module definitions. The identifiers of these SMIL 2.0 modules are shown in Table 3.2.

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

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

3.1.2. NCL Version Information

The following processing instructions shall 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 *id* attribute of an `<ncl>` element may receive any string that matches the NCName type [Namespaces in XML:1999] as a value. That is, it may receive any string value that begins with a letter or an underscore and that only contains letters, digits, "." and "_".

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 is related with version number 3.0.

3.2. NCL Modules: Attributes and Elements of the NCL 3.0 EnhancedDTV Profile

This section briefly describes the main definitions made by each NCL 3.0 module included 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 can be clarified by consulting the XML Schemas.

After discussing each module, a table is presented indicating the module elements and their attributes. For a given profile, attributes and contents (child elements) of an element may be defined in the module itself or in the language profile that groups the modules. Therefore, tables in this section show the attributes and contents that come from NCL Enhanced DTV profile, besides those defined in the modules themselves. Tables in Section 3.3 show the attributes and contents that come from NCL Basic DTV profile, besides those defined in the NCL modules themselves. Attributes of elements 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 order of child-elements 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 an NCM context node.

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

Almost all NCL elements may have the *id* attribute. This attribute may receive any string that matches the NCName type definition [Namespaces in XML: 1999], as its value. That

is, may receive any string value that begins with a letter or an underscore and that only contains letters, digits, "." and "_". The *id* attribute univocally identifies an element in an NCL document.

The <ncl> element shall have the *id* attribute defined, and the <body> element may have. As aforementioned, 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 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 shall 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> child elements should 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	<i>id</i> , <i>title</i> , <i>xmlns</i>	(head?, body?)
head		(importedDocumentBase?, ruleBase?, transitionBase?, regionBase*, descriptorBase?, connectorBase?, meta*, metadata*)
body	<i>id</i>	(port property medial context switch link meta metadata)*

3.2.2. Layout Functionality

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

In short, the <regionBase> element, which may 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 shall 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 may have the values: “systemScreen(*i*)” or “systemAudio(*i*)”, where *i* is an integer greater than zero. The chosen device defines the global environment variables: system.screenSize(*i*), system.screenGraphicSize(*i*), and system.audioType(*i*), as defined in Table 3.6 and discussed in Section 3.2.3.

There are two different types of device classes in NCL: active and passive. In an active class, a device is able to run players for NCL’s media objects. In a passive class, a device is not required to run media players, only to exhibit a bit map or a sequence of audio samples received from another device. The systemScreen (1) and systemAudio(1) are reserved to passive classes, and systemScreen (2) and systemAudio(2) are reserved to passive classes supporting all media players defined by the DTV system.

The <regionBase> element that defines a passive class may also have a *region* attribute. This attribute is used to identify a <region> element in another <regionBase> (of an active class), where the same bit map sent to the passive-class devices will be exhibited.

The interpretation of the region nesting inside a <regionBase> should be made by the software in charge of the document presentation orchestration (called *formatter*).

A <region> may also define the following attributes: *title*, *left*, *right*, *top*, *bottom*, *height*, *width*, and *zIndex*.

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 may 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 shall 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 6.1 illustrates the relative values of these attributes.

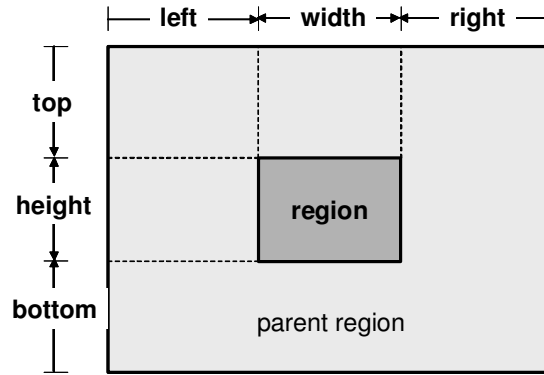


Figure 3.1 – Region positioning attributes

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 can occur. In this case, the *top* and *height* values shall 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 shall 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 shall be inherited from the corresponding parent absolute value. Another restriction is that child regions cannot stay outside the area established by their parent regions. When some portion of the child region lies outside its parent region, the child region shall be ignored (considered as if it is not specified).

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. When not specified, the *zIndex* attribute shall be set equal to zero.

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	<i>id, device, region</i>	(importBaseRegion)+
region	<i><u>id</u>, title, left, right, top, bottom, height, width, zIndex</i>	(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 introduces the <media> element. Each media object has two main attributes, besides its *id* attribute: *src*, which defines the 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 using the http protocol
https:	//server_identifier/file_path/#fragment_identifier	for remote files downloaded using the https protocol
rtsp:	//server_identifier/file_path/#fragment_identifier	for streams downloaded using the rtsp protocol
rtp:	//server_identifier/file_path/#fragment_identifier	for streams downloaded using the rtp protocol
ncl-mirror:	//media_element_identifier	for a content flow identical to the one in presentation by another <media> element
ts:	//program_number.component_tag	for elementary streams received from the transport stream

An absolute URI contains by itself 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 the URI path, as well.

The primary benefit of using relative URIs is that documents and directories containing them may 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 receivers. 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 may also have a relative directory path before the filename.

For media objects with the *src* attributes whose values identify the “ts” scheme, the specific part of the scheme, that is, the *program_number.component_tag*, can be substituted by the following reserved words:

video	Corresponding to the primary video ES being presented on the video plan.
audio	Corresponding to the primary audio ES.
text	Corresponding to the primary text ES.
video(i)	Corresponding to the <i>i</i> th smaller video ES <i>component_tag</i> listed in the PMT of the tuned services.
audio(i)	Corresponding to the <i>i</i> th smaller audio ES <i>component_tag</i> listed in the PMT of the tuned services.
text(i)	Corresponding to the <i>i</i> th smaller text ES <i>component_tag</i> listed in the PMT of the tuned services.

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 is unavailable.

The values allowed for the *type* attribute depend on the NCL profile and shall 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 in IANA, but are defined by common agreement.

Four special types are defined: *application/x-ncl-NCL*; *application/x-ncl-NCLua*, *application/x-ncl-settings*, and *application/x-ncl-time*.

The “*application/x-ncl-NCL*” type shall be applied to <media> elements with NCL code content (note that an NCL application can embed another NCL application).

The *application/x-ncl-NCLua* type shall be applied to <media> elements with Lua code content [SoSC08]³.

A <media> element of *application/x-ncl-NCL* type or of *application/x-ncl-NCLua* type may not have the *instance* and *refer* attributes (see Section 3.2.10).

³ Indeed, Lua is a multi-paradigm programming language (an imperative and functional language).

The application/x-ncl-settings shall be applied to a special <media> element (there may be only one in an NCL document, but it may be reused as child element of other <context> elements) whose properties are global variables (properties defined by the document author or reserved environment properties managed by the system) that may be manipulated by the NCL document processing. Table 3.6 states the already defined environment variables and their semantics.

Table 3.6 – Environment Variables

Group	Variable	Semantics	Possible values
system • set of variables managed by the receiver system; • they may be read, but they may not have their values changed by an NCL application, a Lua procedure or any other imperative or declarative procedure; • receiver's native applications may change the variables' values; • they shall persist during all receiver lifecycle.	system.language	Audio language	ISO 639-1 code
	system.caption	Caption language	ISO 639
	system.subtitle	Subtitle Language	ISO 639
	system.returnBitRate(i)	Bit rate of the interactive channel (i) in Kbps	real
	system.screenSize	Device screen size, in (lines, pixels/line), when a class is not defined	(integer, integer)
	system.screenGraphicSize	Resolution set for the device's screen graphics plane, in (lines, pixels/line), when a class is not defined	(integer, integer)
	system.audioType	Type of the device audio, when a class is not defined	"mono" "stereo" "5.1"
	system.screenSize (i)	Screen size of the device (i) in (lines, pixels/line)	(integer, integer)
	system.screenGraphicSize(i)	Resolution set for the screen graphics plane of the device (i) in (lines, pixels/line)	(integer, integer)
	system.audioType(i)	Type of the audio of the device (i)	"mono" "stereo" "5.1"
	system.devNumber(i)	Number of exhibition devices registered in the class (i)	integer
	system.classType(i)	Type of the class (i)	("passive" "active")
	system.parentDeviceRegion (i)	Identifies a <region> element in another <regionBase> associated with the parent device that creates the bit map sent to the passive-class devices (i); in the specified region the bit map sent must also be exhibited	Five numbers separated by comma, each one following the value rule specified for <i>left</i> , <i>top</i> , <i>width</i> , <i>height</i> , and <i>zIndex</i> properties, respectively.
	system.info(i)	List of class (i)'s media players	string
	system.classNumber	Number of classes that has been defined	integer
	system.CPU	CPU performance in MIPS	real
	system.memory	Minimum memory space in	integer

Group	Variable	Semantics	Possible values
		Mbytes provided to applications	
	system.operatingSystem	Type of the Operating System	string to be defined
	system.luaVersion	Version of the Lua engine supported by the receiver	string
	system.ncl.version	NCL language version	string
	system.GingaNCL.version	Ginga-NCL environment version	string
	system.xxx	Any variable with the “system” prefix shall be reserved for future use	
user • set of variables managed by the receiver system; • they may be read, but they may not have their values changed by an NCL application, a Lua procedure or any other imperative or declarative procedure; • receiver’s native applications may change the variables’ values; • they shall persist during all receiver lifecycle.	user.age	User age	integer
	user.location	User location	string
	user.genre	User genre	“m” “f”
	user.xxx	Any variable with the “user” prefix shall be reserved for future use	
default • set of variables managed by the receiver system; • they may be read and have their values changed by an NCL application, a Lua procedure or any other imperative or declarative procedure; • receiver’s native applications may change the variables’ values; • they shall persist	default.focusBorderColor	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”
	default.selBorderColor	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”
	default.focusBorderWidth	Default width (in pixels) applied to the border of an element in focus	integer

Group	Variable	Semantics	Possible values
during all receiver lifecycle, however, they shall be set to their initial values when a new channel is tuned.	default.focusBorderTransparency	Default transparency applied to the border of an element in focus	a real value between 0 and 1, or a real value in the range [0,100] ending with the character “%” (e.g. 30%), with “1” or “100%” meaning full transparency and “0” or “0%” meaning no transparency
	default.xxx	Any variable with the “default” prefix shall be reserved for future use	
service • set of variables managed by the NCL formatter; • they may be read and have their values changed by an NCL application of the same service; • they may be read but they may not have their values changed by a Lua procedure or any other imperative or declarative procedure of the same service; variable changes shall be done using NCL commands; • they shall persist at least during the service lifecycle.	service.currentFocus	The <i>focusIndex</i> value of the <media> element on focus	integer
	service.currentKeyMaster	Identifier (<i>id</i>) of the <media> element that controls the navigational keys; if the <media> element is not being presented or is not paused, the navigational key control pertains to the NCL Formatter	string
	service.xxx	Any variable with the “service” prefix shall follow the rules specified for the group	
si • set of variables managed by the	si.numberOfServices	Number of services available in the country for the tuned channel. ⁴	integer
	si.numberOfPartialServices	Number of 1-seg services	integer

⁴ For example, in SBTVD-T, the value for this variable should be obtained from the number of PMT tables specified in the PAT table of the transport stream received in the tuning channel. The variable value should take into account only the PMT tables whose fields *country_code*, specified in the *country_availability_descriptor* related with the table, are equivalent to the value of the *user.location* variable of the Settings node.

Group	Variable	Semantics	Possible values
middleware; • they may be read, but they may not have their values changed by an NCL application, a Lua procedure or any other imperative or declarative procedure; they shall persist at least until the next channel tuning.		available in the country for the tuned channel. ⁵	
	si.channelNumber	Number of the tuned channel. ⁶	integer
	si.xxx	Any variable with the “si” prefix shall follow the rules specified for the group	
channel • set of variables managed by the NCL formatter; • they may be read and have their values changed by an NCL application of the same channel; • they may be read but they may not have their values changed by a Lua procedure or any other imperative or declarative procedure of the same channel; variable changes shall be done using NCL commands; • they shall persist at least until the next channel tuning.	channel.keyCapture	Request of group of keys for NCL applications	“00”, significando nenhuma reserva “FF”, reservando simultaneamente o conjunto de todas as teclas possíveis valores entre 00 e FF indicando conjunto de teclas a definir pelo sistema
	channel.virtualKeyboard	Request of a virtual keyboard for NCL applications	(true false)
	channel.keyboardBounds	Virtual keyboard region (left, top, width, height)	(integer, integer, integer, integer)
	channel.xxx	Any variable with the “channel” prefix shall follow the rules specified for the group	
shared • set of variables managed by the	shared.xxx	Any variable with the “shared” prefix shall follow the rules specified for the	

⁵ For example, in SBTVD-T, the value for this variable should be obtained from the number of PMT tables specified in the PAT table of the transport stream received in the tuning channel. The variable value should take into account only the PMT tables whose country_code fields, specified in the country_availability_descriptor related with the table, are equal to the value of the user.location variable of the Settings node, and whose program_number fields are equivalent to the service_id fields of the partial_reception_descriptor related with the NIT tables.

⁶ For example, in SBTVD-T, the value for this variable should be obtained from the remote_control_key_id field of the ts_information_descriptor of the NIT table that describes the current service.

Group	Variable	Semantics	Possible values
NCL formatter; • they may be read and have their values changed by an NCL application; • they may be read but they may not have their values changed by a Lua procedure or any other imperative or declarative procedure; variable changes shall be done using NCL commands; • they shall persist at least during the lifecycle of the service that has defined them.		group	

The application/x-ncl-time type shall be applied to a special <media> element (there may be only one in an NCL document, but it may be reused as child element of other <context> elements) whose content (*src* attribute defined by default) is the Universal Time Coordinated (UTC). When it is started, the UTC content is not exhibited, however, it can be used to define content anchors by using <area> elements. Note that any continuous <media> element with no source may be used to define a clock relative to the <media> element start time.

The content of a <media> element of application/x-ncl-time type shall be specified according with the following syntax:

Year“:”Month“:”Day“:”Hours“:”Minutes“:”Seconds“.”Fraction,

where Year is an integer; Month is an integer in the [1,12] interval; Day is an integer in the [1,31] interval; Hours is an integer in the [0,23] interval; Minutes is an integer in the [0,59] interval; Seconds is an integer in the [0,59] interval; Fraction is a positive integer.

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.⁷

⁷ When there is more than one player supported by the formatter for the same type, the *player* property of the <media> element may specify which one will be used for presentation. Otherwise the formatter shall use a default player for that type of media.

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 that allows for structuring an NCL application. The <context> element may contain elements with media content (<media> elements), other nested <context> elements, elements grouping content alternatives (<switch> elements), and relationships (<link> elements) among all the objects represented by the aforementioned elements. As usual, the *id* attribute uniquely identifies the <context> element within a document.

The *instance*, *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	<i>id</i> , <i>src</i> , <i>refer</i> , <i>instance</i> , <i>type</i> , <i>descriptor</i>	(arealproperty)*

Table 3.8 – Extended Context Module in the Enhanced DTV profile

Elements	Attributes	Content
context	<i>id</i> , <i>refer</i>	(port property media context link switch meta metadata)*

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, NCL does not restrict or prescribe the media-object content types. In this sense, we may 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's media objects. Which are the media objects supported depends on the media players embedded in the NCL formatter.

A 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, LIME browsers, HbbTV 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 shall 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 for the definition of node (media object or composite objects) interfaces that will be used in relationships with other node interfaces. This functionality is partitioned into four modules:

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

The MediaContentAnchor module defines the <area> element, which allows for 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* attributes⁸; and the definition of content anchors representing temporal and spatial portions through *coords*, *begin* and *end* attributes. In addition, the <area> element allows for the definition of textual anchors, through the *beginText*, *beginPosition* and *endText*, *endPosition* attributes that define the beginning and end strings and the string's occurrences in the text. Besides, the <area> element may also define a content anchor based on the number of audio samples or video frames, through *first* and *last* attributes⁹, which shall indicate the initial and final sample/frame. Moreover, the <area> element may also define a content anchor based on the *label* attribute, which specifies a string that should be used by the media player to identify a content region (for example, a function or a method in an imperative media object, an interface in a declarative media object, like an HTML anchor, a port in a media object with an NCL application as a content, etc.), and also, for

⁸ Except for the <media> element of the application/x-ginga-time type, the *begin* and *end* attributes shall be specified according with one of the following syntax:

- i) Hours": "Minutes": "Seconds". "Fraction, Hours is an integer in the [0,23] interval; Minutes is an integer in the [0,59] interval; Seconds is an integer in the [0,59] interval; Fraction is a positive integer
- ii) Seconds"s", where Seconds is a positive real number.

⁹ The *first* and *last* attributes shall be specified according with one of the following syntax:

- i) Samples"s", where Samples is a positive integer;
- ii) Frames"f", where Frames is a positive integer;
- iii) NPT"npt", where NPT is the Normal Play Time value.

hypermedia declarative objects (see “NCL 3.0 Part 11 – Declarative Hypermedia Objects in NCL: Nesting Objects with NCL Code in NCL Documents”), a content anchor based on the *clip* attribute which specifies a triple value that should be used by the media player to identify a clip in a temporal chain defined by the hypermedia declarative object.

If the *begin* attribute is defined, but the *end* attribute is not specified, the end of the whole media content presentation shall be assumed as the anchor ending. On the other hand, if the *end* attribute is defined, but without an explicit *begin* definition, the start of the whole media content presentation shall be considered as the anchor beginning. Analogous behavior is expected from the *first* and *last* attributes. In the case of a <media> element of the application/x-ncl-time type, the *begin* and *end* attributes shall be always defined and shall assume an absolute value of the Universal Time Coordinated (UTC)¹⁰. In textual content anchors, if the end of the anchor region is not defined, the end of the text content shall be assumed. If the beginning of the content anchor region is not defined, the beginning of the text content shall be assumed.

As usual, <area> elements shall 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	<i>id, coords, begin, end, beginText, beginPosition endText, endPosition, first, last, label, clip</i>	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).

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	<i>id, component, interface</i>	empty

¹⁰ For the <media> element of the application/x-ginga-time type, the *begin* and *end* attributes shall be specified according with the following syntax: Year“:”Month“:”Day“:”Hours“:”Minutes“:”Seconds“.”Fraction, according to the country time zone. The NCL user agent is responsible for translating the value for the country time zone to the one corresponding to the UTC.

In NCM, every node (media or context node) shall 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 documents. Except for media objects with imperative code content (<media type=“application/x-ncl-NCLua” ...>, for example), every time an NCL component is referred without specifying one of its anchors, the *whole content anchor* is assumed.

The PropertyAnchor module defines an element named <property>, which may be used for defining a node property or a group of node properties as one of its interfaces (anchors)¹¹. The <property> element defines the *name* attribute, which indicates the name of the property or of the property group, and the *value* attribute, an optional attribute that defines an initial value for the *name* property. The parent element cannot have <property> elements with the same *name* attribute values. When a formatter treats a change in a property group it shall only test the process consistency at its end.

The <body>, <context>, and <media> elements may have several embedded properties, which are not explicitly declared by <property> elements. Some predefined properties and group of properties for <media> elements are:

Predefined properties for all <media> elements:

Name	Value	Description	Default
style	URI	The locator of a stylesheet file that specifies other properties for the object.	—
player	string	Identifies the presentation tool to be used.	If not specified, the player is determined by the object type, defined by the type attribute of the <media> element, or inferred by the file extension of the <i>src</i> attribute value of the <media> element.
reusePlayer	“true” “false”	Specifies if a new player must be instantiated (“false”) or if a player already instantiated must be used (“true”).	false
playerLife	“close” “keep”	Specifies what will happen to the player instance at the end of the object presentation.	close
deviceClass	“systemScreen(i)” or “systemAudio(i)”, where <i>i</i> is a non-negative	Specifies the device class where the media object will be presented. There are two different types of	The same device class that runs the NCL player.

¹¹ It is possible to have NCL document players (formatters) that define some node properties as node interfaces, implicitly. However, in general, it is a good practice to explicitly define the interfaces.

	integer	device classes: active and passive. In an active class, a device is able to run some media players for <media> element presentations. In a passive class, a device is not required to run media players, only to exhibit a bit map or a sequence of audio samples received from another device. The “systemScreen(1)” and “systemAudio(1)” are reserved to passive classes, and “systemScreen(2)” and “systemAudio(2)” are reserved to active classes.	
explicitDur	The <i>explicitDur</i> property is specified according with one of the following syntax: i) Hours“:.”Minutes“:.”Seconds“.”Fraction, where Hours is an integer in the [0,23] interval; Minutes is an integer in the [0,59] interval; Seconds is an integer in the [0,59] interval; and Fraction is a positive integer ii) Seconds“s”, where Seconds is a positive real number.	Gives the duration of an object.	For continuous media, the default value is set to the natural content presentation duration, otherwise, it must be set to null
focusIndex	Positive integer	Specifies an index for the <media> element to which the focus may be applied	— When this property is not defined, the object is considered as if no focus could be set to it
moveLeft	Positive integer	Specifies a value equal to the <i>focusIndex</i> value associated to the element to which the focus should be applied when the “left arrow key” is pressed.	—
moveRight	Positive integer	Specifies a value equal to the <i>focusIndex</i> value associated to the element to which the focus should be applied when the “right arrow key” is pressed.	—
moveUp	Positive integer	Specifies a value equal to the <i>focusIndex</i> value associated to the element to which the focus should be applied when the “up arrow key” is pressed.	—
moveDown	Positive integer	Specifies a value equal to the <i>focusIndex</i> value associated to an element to which the focus should be applied when the “down arrow key” is pressed.	—

Predefined properties for <media> elements with visual contents:

Name	Value	Description	Default
top	A real number in the range [0,100] ending with the character “%” (e.g. 30%), or an integer non-negative value specifying the attribute in pixels. For pixel values, the “px” unit qualifier may be omitted.	Top position of the content, expressed as a percentage of the exhibition device <i>height</i> dimension, or as an absolute value. It places the top position of the content to be exhibited in the specified distance away from the top edge of the exhibition device.	0%, if the <i>bottom</i> value is not specified.
bottom	If the <i>top</i> and <i>height</i> values are specified, it must be computed based on these values. Otherwise, it must be a real number in the range [0,100] ending with the character “%” (e.g. 30%), or an integer non-negative value specifying the attribute in pixels. For pixel values, the “px” unit qualifier may be omitted.	Bottom position of the content, expressed as a percentage of the exhibition device <i>height</i> dimension, or as an absolute value. It places the bottom position of the content to be exhibited in the specified distance away from the bottom edge of the exhibition device.	0%, if the attribute value cannot be computed from the <i>height</i> and <i>top</i> attributes.
left	A real number in the range [0,100] ending with the character “%” (e.g. 30%), or an integer non-negative value specifying the attribute in pixels. For pixel values, the “px” unit qualifier may be omitted.	Left position of the content, expressed as a percentage of the exhibition device <i>width</i> dimension, or as an absolute value. It places the left position of the content to be exhibited in the specified distance away from the left edge of the exhibition device.	0%, if the <i>right</i> value is not specified.
right	If the <i>left</i> and <i>width</i> values are specified, it must be computed based on these values. Otherwise, it must be a real number in the range [0,100] ending with the character “%” (e.g. 30%), or an integer non-negative value specifying the attribute in pixels. For pixel values, the “px” unit qualifier may be omitted.	Right position of the content, expressed as a percentage of the exhibition device <i>width</i> dimension, or as an absolute value. It places the right position of the content to be exhibited in the specified distance away from the right edge of the exhibition device.	0%, if the attribute value cannot be computed from the <i>width</i> and <i>left</i> attributes.
width	A real number in the range [0,100] ending with the character “%” (e.g. 30%), or an integer non-negative value specifying the attribute in pixels. For pixel values, the “px” unit qualifier may be omitted.	Width of the content to be exhibited, expressed as a percentage of the exhibition device <i>width</i> dimension, or as an absolute value.	0%
height	A real number in the range [0,100] ending with the	Height of the content to be exhibited, expressed as a	0%

	character “%” (e.g. 30%), or an integer non-negative value specifying the attribute in pixels. For pixel values, the “px” unit qualifier may be omitted.	percentage of the exhibition device <i>height</i> dimension, or as an absolute value.	
location	See <i>left</i> and <i>top</i> properties	Two numbers separated by comma, each one following the rule specified for <i>left</i> and <i>top</i> properties, respectively	See <i>left</i> and <i>top</i> properties
size	See <i>width</i> and <i>height</i> properties	Two numbers separated by comma, each one following the rule specified for <i>width</i> and <i>height</i> properties, respectively	See <i>width</i> and <i>height</i> properties
bounds	See <i>left</i> , <i>top</i> , <i>width</i> and <i>height</i> properties	Four numbers separated by comma, each one following the rule specified for <i>left</i> , <i>top</i> , <i>width</i> and <i>height</i> properties, respectively	See <i>left</i> , <i>top</i> , <i>width</i> and <i>height</i> properties
background	“white”, “black”, “silver”, “gray”, “red”, “maroon”, “fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, or “teal” (see table of colors after the tables of pre-defined properties) It may also be the reserved value “transparent”.	The background color used to fill the area of a region for displaying the media content that is not filled by the content itself.	transparent
rgbChromaKey	RGB color defined using a hexadecimal (hex) notation for the combination of Red, Green, and Blue color values (RGB). The lowest value that can be given to one of the light sources is 0 (hex 00). The highest value is 255 (hex FF). It may also be defined by the reserved words: “white”, “black”, “silver”, “gray”, “red”, “maroon”, “fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, or “teal” (see table of colors after the tables of pre-defined properties).	The RGB color to be set as transparent.	—
visible	“true”	Allows the object presentation to be seen or	true

	“false”	hidden. When the <i>visible</i> property associated with a <context> or <body> element is equal to “true”, the <i>visible</i> property of each child element of the composition is taken into account to define how each of these child element will be exhibited. When the <i>visible</i> property associated with a <context> or <body> element is equal to “false”, all child elements of the composition are exhibited as hidden. It should be remarked that an object with a <i>visible</i> property equal to “false”, that is, an object exhibited as hidden, may not have any of its content anchors selected while the <i>visible</i> property value persists as “false”. When a document has its <body> element with its <i>visible</i> property set to “false” and its presentation event is in the <i>paused</i> state, the document is said to be in <i>stand-by</i>. In ISDB-T standard, when there is only one application in execution and this application is in stand-by, the service’s main video is dimensioned to 100% of the screen, and the main audio is set to 100% of its volume.	
transparency	A real number in the range [0,1] or a real number in the range [0,100] ending with the character “%” (e.g. 30%),	Indicates the degree of transparency of an object presentation: “1” or “100%” means full transparency and “0” or “0%” means opaque.	0%
fit	“fill” “hidden” “meet” “meetBest” “slice”	Indicates how an object will be presented. “fill”: scale the object's media content so that it touches all edges of the box defined by the object’s <i>width</i> and <i>height</i> properties. “hidden”: if the intrinsic height (width) of the media content is smaller than the	fill

		<i>height</i> (<i>width</i>) property, 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 <i>height</i> (<i>width</i>) property, the object must be rendered starting from the top (left) edge until the height (width) defined in the property is reached; the part of the media content below (to right of) the height (width) is 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 <i>height</i> or <i>width</i> properties. 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 <i>height</i> and <i>width</i> properties 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”	Allows for the specification of how the scroll is configured in an exhibition region.	none
zIndex	An integer number in the range [0, 255].	The <i>zIndex</i> attribute specifies the superposition precedence, where objects	0

		with greater <i>zIndex</i> values are stacked on top of objects with smaller <i>zIndex</i> values.	
plan	“video”, “graphic” “background”	Defines in which plan of a structured screen an object will be placed.	“video”, for media with <i>src</i> attribute referring to a PES of a Transport Stream. ”graphic”, for all other cases.
focusBorderColor	“white”, “black”, “silver”, “gray”, “red”, “maroon”, “fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, or “teal” (see table of colors after the tables of pre-defined properties)	Defines the color to decorate the screen region where the object on focus is being presented, before it is selected.	System dependent: the value defined by the <i>default.focusBorderColor</i> property of the <i>settings</i> object.
selBorderColor	“white”, “black”, “silver”, “gray”, “red”, “maroon”, “fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, or “teal” (see table of colors after the tables of pre-defined properties).	Defines the color to decorate the screen region where the object on focus is being presented, after it is selected.	System dependent: the value defined by the <i>default.selBorderColor</i> property of the <i>settings</i> object.
focusBorderWidth	integer	Width (in pixels) applied to the border of an element in focus.	System dependent: the value defined by the <i>default.focusBorderWidth</i> property of the <i>settings</i> object.
focusBorderTransparency	A real value between 0 and 1, or a real value in the range [0,100] ending with the character “%” (e.g. 30%)	Transparency applied to the border of an element in focus, with “1” or “100%” meaning full transparency and “0” or “0%” meaning no transparency.	System dependent: the value defined by the <i>default.focusTransparency</i> property of the <i>settings</i> object.
focusSrc	URI	Specifies 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 <i>src</i> attribute of the <media> element.	—
focusSelSrc	URI	Specifies an alternative media source to be presented, instead of the	—

		current presentation, when an element on focus is selected by pressing the activation (select or enter) key. This attribute follows the same rules of the <i>src</i> attribute of the <media> element.	
freeze	“true” “false”	Defines what will happen with an object at the end of its presentation, if the end content is frozen (“true”) or not (“false”).	“false”
transIn, transOut	String list of (type, subtype, dur, startProgress, endProgress, direction, fadeColor, horzRepeat, vertRepeat, borderWidth, borderColor) separated by semicolons. Where: - type can be: “barWipe”, “irisWipe”, “clockWipe”, “snakeWipe”, or “fade”. - subtype can be: - dur is specified according with one of the following syntax: i) Hours“:”Minutes“:”Seconds“.”Fraction, where Hours is an integer in the [0,23] interval; Minutes is an integer in the [0,59] interval; Seconds is an integer in the [0,59] interval; and Fraction is a positive integer ii) Seconds“s”, where Seconds is a positive real number. - startProgress is a real number in the range [0.0,1.0]. - endProgress is a real number in the range [0.0,1.0]. - direction values are “forward” and “reverse”. - fadeColor is “white”, “black”, “silver”, “gray”, “red”, “maroon”, “fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, or “teal” (see	The value of the <i>transIn</i> and <i>transOut</i> is a semicolon-separated list of transitions. 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. The <i>type</i> and <i>subtype</i> define the transition. The <i>dur</i> attribute specifies the duration of the transition. The <i>startProgress</i> attribute specifies the amount of progress through the transition at which to begin execution. For instance, we can want to begin a crossfade with the destination image being already 30% faded in. For this case, <i>startProgress</i> would be 0.3. The <i>endProgress</i> attribute specifies the amount of progress through the transition at which to end execution. The value of this attribute must be greater than or equal to the value of the <i>startProgress</i> attribute. If <i>endProgress</i> is equal to <i>startProgress</i> , then the transition remains at a fixed	empty string If a transition type is specified then: - The <i>dur</i> default duration is 1s. - The <i>startProgress</i> default value is 0.0. - The <i>endProgress</i> default value is 1.0. - The <i>direction</i> default value is “forward”. - The <i>fadeColor</i> default value is “black”. - The <i>horzRepeat</i> default value is 1. - The <i>vertRepeat</i> default value is 1. - The <i>borderWidth</i> default value is 0. - The <i>borderColor</i> default value for is “black”

	table of colors after the tables of pre-defined properties) • horzRepeat is a positive integer • vertRepeat is a positive integer • borderWidth is a non-negative integer • borderColor is "white", "black", "silver", "gray", "red", "maroon", "fuchsia", "purple", "lime", "green", "yellow", "olive", "blue", "navy", "aqua", or "teal" (see table of colors after the tables of pre-defined properties)	progress for the duration of the transition. The <i>direction</i> attribute specifies the direction the transition will run. 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 <i>direction</i> attribute ignored and the default value of "forward" assumed. If the value of the <i>type</i> attribute is "fade" and the value of the <i>subtype</i> attribute is "fadeToColor" or "fadeFromColor" (values that are not required to be supported in a Ginga implementation of ISDB-T), then the <i>fadeColor</i> attribute specifies the ending or starting color of the fade. If the value of the <i>type</i> attribute is not "fade", or the value of the <i>subtype</i> attribute is not "fadeToColor" or "fadeFromColor", then the <i>fadeColor</i> attribute is ignored. If the value of the <i>type</i> attribute is not "fade", then the following attributes can be specified: The <i>horzRepeat</i> attribute specifies how many times to perform the transition pattern along the horizontal axis. The <i>vertRepeat</i> attribute specifies how many times to perform the transition pattern along the vertical axis. The <i>borderWidth</i> attribute specifies the width of a generated border along a wipe edge. The <i>borderColor</i> 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.	
--	--	--	--

Predefined properties for <media> elements with continuous-media content (Read-only properties):

Name	Value	Description	Default
contentId	string (Set only by the NCL player)	The <i>contentId</i> property is associated to continuous-media objects whose content is defined referring to an elementary stream. It has "null" as its initial value and it is set to the identifier value transported in the NPT reference descriptor (in a field of the same name: contentId), as soon as the associated continuous-media object is started.	"null"
standby	"true" "false" (Set only by the NCL player)	The <i>standby</i> property is set to "true" while an already started continuous-media object content referring to an elementary stream is temporarily interrupted by another interleaved content, in the same elementary stream.	"false"

Predefined properties for <media> elements with audible contents:

Name	Value	Description	Default
soundLevel	A real number in the range [0, 1] or a real number in the range [0,100] ending with the character "%" (e.g. 30%).	Audio volume relative to the recorded audio.	1
balanceLevel	A real number in the range [-1, 1] or a real number in the range [-100,100] ending with the character "%" (e.g. 30%).	Balance level of the audio (-1 to the left and +1 to the right)	0
trebleLevel	A real number in the range [0, 1] or a real number in the range [0,100] ending with the character "%" (e.g. 30%).	Treble setting relative to the recorded audio.	1
bassLevel	A real number in the range [0, 1] or a real number in the range [0,100] ending with the character "%" (e.g. 30%).	Bass setting relative to the recorded audio.	1

freeze	“true” “false”	This property defines what will happen with an object at the end of its presentation, if the end content is frozen (“true”) or not (“false”).	“false”
transIn, transOut	String (dur, startProgress, endProgress) Where: - type is “fade” by default. - subtype is “crossfade” by default. - dur is specified according with one of the following syntax: i)Hours:.”Minutes:.”Second s”.Fraction, where Hours is an integer in the [0,23] interval; Minutes is an integer in the [0,59] interval; Seconds is an integer in the [0,59] interval; and Fraction is a positive integer ii) Seconds“s”, where Seconds is a positive real number. - startProgress is a real number in the range [0.0,1.0]. - endProgress is a real number in the range [0.0,1.0].	The value of the The <i>type</i> of the <i>transIn</i> and <i>transOut</i> is always “fade” and the <i>subtype</i> is always “crossfade” and is defined by default. The <i>dur</i> attribute specifies the duration of the transition. The <i>startProgress</i> attribute specifies the amount of progress through the transition at which to begin execution. The <i>endProgress</i> attribute specifies the amount of progress through the transition at which to end execution. The value of this attribute must be greater than or equal to the value of the <i>startProgress</i> attribute. If <i>endProgress</i> is equal to <i>startProgress</i> , then the transition remains at a fixed progress for the duration of the transition.	empty string If a transition is specified then: - The <i>dur</i> default duration is 1s. - The <i>startProgress</i> default value is 0.0. - The <i>endProgress</i> default value is 1.0.

Predefined properties for <media> elements with textual contents:

Name	Value	Description	Default
fontColor	“white”, “black”, “silver”, “gray”, red”, “maroon”, fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, or “teal” (see table of colors after the tables of pre-defined properties)	Defines the font color.	white
fontFamily	A string: a prioritized list of font family names and/or generic family names. There are two types of font family names: - family-name: the name of a font family of choice. - generic-family: the following generic	The list values are separated by a comma to indicate that they are alternatives.	System dependent

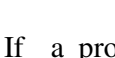
	families are defined: “serif”, “sans-serif”, “cursive”, “fantasy”, “monospace”		
textAlign	“left”, “right”, “center”, “justify”	This property describes how text content is aligned.	left
fontStyle	“normal”, italic”	Defines the font style.	normal
fontSize	positive integer ending with the character “px” or “pt”, for relative font size expressed in pixels, or absolute font size expressed in points, respectively.	Sets the font-size to a fixed size in pixel or in points.	System dependent
fontVariant	“small-caps”, “normal”	Defines a variation type within a font family	normal
fontWeight	“normal”, “bold”	This property selects the weight of the font.	normal
freeze	“true” “false”	This property defines what will happen with an object at the end of its presentation, if the end content is frozen (“true”) or not (“false”).	“false”
transIn transOut	String list of (type, subtype, dur, startProgress, endProgress, direction, fadeColor, horzRepeat, vertRepeat, borderWidth, borderColor) separated by semicolons. Where: • type can be: “barWipe”, “irisWipe”, “clockWipe”, “snakeWipe”, or “fade”. • subtype can be: • dur is specified according with one of the following syntax: i) Hours:”Minutes”:”Seconds”.”Fraction, where Hours is an integer in the [0,23] interval; Minutes is an integer in the [0,59] interval; Seconds is an integer in the [0,59] interval; and Fraction is a positive integer ii) Seconds”s”, where Seconds is a positive real number. • startProgress is a real number in the range [0.0,1.0]. • endProgress is a real number in the range [0.0,1.0]. • direction values are “forward” and “reverse”. • fadeColor is “white”, “black”, “silver”, “gray”, “red”, “maroon”, fuchsia”,	The value of the <i>transIn</i> and <i>transOut</i> is a semicolon-separated list of transitions. 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. The <i>type</i> and <i>subtype</i> define the transition. The <i>dur</i> attribute specifies the duration of the transition. The <i>startProgress</i> attribute specifies the amount of progress through the transition at which to begin execution. For instance, we can want to begin a crossfade with the destination image being already 30% faded in. For this case, <i>startProgress</i>	empty string If a transition type is specified then: • The <i>dur</i> default duration is 1s. • The <i>startProgress</i> default value is 0.0. • The <i>endProgress</i> default value is 1.0. • The <i>direction</i> default value is “forward”. • The <i>fadeColor</i> default value is “black”. • The <i>horzRepeat</i> default value is 1. • The <i>vertRepeat</i> default value

	“purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, or “teal” (see table of colors after the tables of pre-defined properties) • horzRepeat is a positive integer • vertRepeat is a positive integer • borderWidth is a non-negative integer • borderColor is “white”, “black”, “silver”, “gray”, “red”, “maroon”, “fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, or “teal” (see table of colors after the tables of pre-defined properties)	would be 0.3. The <i>endProgress</i> attribute specifies the amount of progress through the transition at which to end execution. The value of this attribute must be greater than or equal to the value of the <i>startProgress</i> attribute. If <i>endProgress</i> is equal to <i>startProgress</i>, then the transition remains at a fixed progress for the duration of the transition. The <i>direction</i> attribute specifies the direction the transition will run. 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 <i>direction</i> attribute ignored and the default value of “forward” assumed. If the value of the <i>type</i> attribute is “fade” and the value of the <i>subtype</i> attribute is “fadeToColor” or “fadeFromColor” (values that are not required to be supported in a Ginga implementation of ISDB-T), then the <i>fadeColor</i> attribute specifies the ending or starting color of the fade. If the value of the <i>type</i> attribute is not “fade”, or the value of the <i>subtype</i> attribute is not “fadeToColor” or “fadeFromColor”, then the <i>fadeColor</i> attribute is ignored. If the value of the <i>type</i> attribute is not “fade”, then the following attributes can be specified: The <i>horzRepeat</i> attribute specifies how many times	is 1. • The <i>borderWidth</i> default value is 0. • The <i>borderColor</i> default value for is “black”
--	--	--	--

		to perform the transition pattern along the horizontal axis. The <i>vertRepeat</i> attribute specifies how many times to perform the transition pattern along the vertical axis. The <i>borderWidth</i> attribute specifies the width of a generated border along a wipe edge. The <i>borderColor</i> 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.	
--	--	---	--

Properties that have reserved color string names as values follow the CSS1 color standard, as defined in next table.

Reserved names for color definition

	Name	Hexadecimal	R	G	B	Hue	Satur.	Light	Satur.	Value
	White	#FFFFFF	100%	100%	100%	0°	0%	100%	0%	100%
	Silver	#C0C0C0	75%	75%	75%	0°	0%	75%	0%	75%
	Gray	#808080	50%	50%	50%	0°	0%	50%	0%	50%
	Black	#000000	0%	0%	0%	0°	0%	0%	0%	0%
	Red	#FF0000	100%	0%	0%	0°	100%	50%	100%	100%
	Maroon	#800000	50%	0%	0%	0°	100%	25%	100%	50%
	Yellow	#FFFF00	100%	100%	0%	60°	100%	50%	100%	100%
	Olive	#808000	50%	50%	0%	60°	100%	25%	100%	50%
	Lime	#00FF00	0%	100%	0%	120°	100%	50%	100%	100%
	Green	#008000	0%	50%	0%	120°	100%	25%	100%	50%
	Aqua	#00FFFF	0%	100%	100%	180°	100%	50%	100%	100%
	Teal	#008080	0%	50%	50%	180°	100%	25%	100%	50%
	Blue	#0000FF	0%	0%	100%	240°	100%	50%	100%	100%
	Navy	#000080	0%	0%	50%	240°	100%	25%	100%	50%
	Fuchsia	#FF00FF	100%	0%	100%	300°	100%	50%	100%	100%
	Purple	#800080	50%	0%	50%	300°	100%	25%	100%	50%

If a property is used in a relationship, it shall be explicitly declared as a <property> (interface) element. Therefore, each property has a hidden Boolean attribute named *externable*, which is defined as “false” by *default* and as “true” when the property is explicitly declared by a <property> element.

Note that all properties (and their initial values) of an NCL object may be defined using only <property> elements. The <descriptor>, <descriptorParam>, and <region> elements are only additional options for defining initial values for properties.

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

Table 3.11 – Extended PropertyAnchor Module in the Enhanced DTV profile

Elements	Attributes	Content
property	<u>name</u> , <i>value</i> , <i>externable</i>	empty

The SwitchInterface module allows for the creation of interfaces for <switch> elements. These interfaces may 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).

Finishing this section, it is important to remark that every element representing an object interface (<area>, <port>, <property>, and <switchPort>) shall have an identifier (*id* attribute or *name* attribute).

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

Table 3.12 – Extended SwitchInterface Module in the Enhanced DTV profile

Elements	Attributes	Content
switchPort	<u>id</u>	mapping+
mapping	<u>component</u> , <i>interface</i>	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 start each object of an NCL document. This information is modeled by *descriptor* objects, according to NCM [SoRo05].

The Descriptor module allows for the definition of <descriptor> elements, which define the initial values for properties of <media> elements. These initial values are defined in attributes of the <descriptor> element, or in child <descriptorParam> elements.

It should be stressed that if several values are specified for the same property, the value defined in a <property> element has precedence over the one defined in a <descriptorParam> element, which has precedence over the value defined in an attribute of the corresponding <descriptor> element (including the *region* attribute).

The definition of <descriptor> elements shall be included in the document head, inside the <descriptorBase> element, which specifies the set of descriptors of a document. The

<descriptor> element shall 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); and transition attributes: *transIn* and *transOut* (see Section 3.2.13).

A <descriptor> element may also have <descriptorParam> child elements, which define initial values for any property of <media> elements, including all those pre-defined properties defined in Section 3.2.4. The property is identified in the *name* attribute of the <descriptorParam> element and the value is defined in the *value* attribute.

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.

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 media and context objects of a document. Following NCM directives, NCL 3.0 EDTV and NCL 3.0 BDTV profile establish that the *descriptor* attribute may be an attribute of <media> elements or of <bind> elements, allowing the association of descriptors to <media> elements directly or through relationships represented by <link> elements.

The descriptor shall be chosen by the formatter in agreement with the following directives. If the *start* instruction over a <media> element results from a link action that has a descriptor explicitly declared in its <bind> element, the resulting descriptor shall merge the attributes of the bind descriptor with the attributes of the descriptor specified in the corresponding <media> element, if this attribute was specified. For the common attributes, the <bind> descriptor information shall superpose the <media> descriptor data. If the <bind> element does not contain an explicit descriptor, the descriptor evaluated by the formatter shall be the <media> descriptor, if this attribute was specified. Otherwise, a default descriptor for that *type* of <media> may be chosen by the formatter.

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	<i>id</i> , <i>player</i> , <i>explicitDur</i> , <i>region</i> , <i>freeze</i> , <i>moveLeft</i> , <i>moveRight</i> , <i>moveUp</i> , <i>moveDown</i> , <i>focusIndex</i> , <i>focusBorderColor</i> , <i>focusBorderWidth</i> , <i>focusBorderTransparency</i> , <i>focusSrc</i> , <i>focusSelSrc</i> , <i>selBorderColor</i> , <i>transIn</i> , <i>transOut</i>	(descriptorParam)*
descriptorParam	<u><i>name</i></u> , <u><i>value</i></u>	
descriptorBase	<i>id</i>	(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 shall have an *xconnector* attribute, which refers to a hypermedia connector URI. The reference shall 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 a node. The third is an optional attribute called *interface*¹², used for making reference to an interface of the node. 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 having *name* and *value* attributes. In this case the *name* attribute shall refer to the name of a connector parameter while the *value* attribute shall define a value to be assigned to the respective parameter.

¹² Note that the *interface* attribute may refer to any node interface, that is, an anchor, a property, a port (if it is a composite node), or a switchPort (if it is a switch node). The interface attribute is optional. When it is not specified, the association will be done with the whole node content, as explained in 3.2.4, except for media objects with imperative code content.

If a link defines the same parameter through using the <linkParam> and <bindParam> elements, the definition by using <bindParam> element has precedence.

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	<i>role, component, interface, descriptor</i>	(bindParam)*
bindParam	<i>name, value</i>	empty
linkParam	<i>name, value</i>	empty
link	<i>id, xconnector</i>	(linkParam*, bind+)

3.2.7. Connectors Functionality

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

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 may be used to specify spatio-temporal synchronization relations, treating reference (user interaction) relations as a particular case of temporal synchronization relations.

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

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

A <causalConnector> element 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, may reuse the

same connector, reusing all previous relation 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 may 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 information units of a media object, specified in NCL by the `<area>` element, or by the media node itself (its whole content presentation). Presentation events may also be defined on composite nodes (represented by a `<body>`, `<context>`, or `<switch>` element), representing the presentation of information units of any node inside a composite node;
- *selection event*, which is defined by the selection of a subset of information units of a media object being presented, specified in NCL by the `<area>` element, or by the media node itself (its whole content presentation);
- *attribution event*, which is defined by the attribution of a value to a property of a node (represented by a `<media>`, `<body>`, `<context>`, or `<switch>` element), which shall be declared in a `<property>` 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 (the composite organization).

Each event defines a state machine that should 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 of presentation and attribution types have also an attribute named *repetitions*, which counts how many times the event shall be automatically restarted (transited from sleeping to occurring states) by the formatter. This attribute may contain the “indefinite” value, leading to an endless loop of the event occurrences until some external interruption.

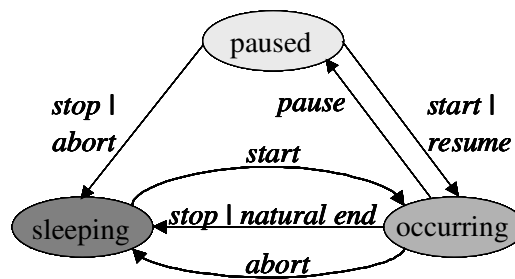


Figure 3.2 - Event state machine

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
<i>sleeping</i> → <i>occurring</i> (<i>start</i>)	<i>starts</i>
<i>occurring</i> → <i>sleeping</i> (<i>stop or natural end</i>)	<i>stops</i>
<i>occurring</i> → <i>sleeping</i> (<i>abort</i>)	<i>aborts</i>
<i>occurring</i> → <i>paused</i> (<i>pause</i>)	<i>pauses</i>
<i>paused</i> → <i>occurring</i> (<i>resume or start</i>)	<i>resumes</i>
<i>paused</i> → <i>sleeping</i> (<i>stop</i>)	<i>stops</i>
<i>paused</i> → <i>sleeping</i> (<i>abort</i>)	<i>aborts</i>

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 may 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 may 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 anyone 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 anyone of the composite child nodes is in the paused state and all other presentation events associated with the composite child nodes are in the sleeping or paused state. Otherwise, the presentation event is in the sleeping state.¹³

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.

Attribution events stay in the occurring state while the corresponding property values are being modified. Obviously, instantaneous events, like attribution events for simple value assignments, may stay in the occurring state only during an infinitesimal period of time.

¹³ More details about the behavior of presentation event state machines for media and composite nodes are given in Appendix B.

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) property values. The CausalConnectorFunctionality module allows only for 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 shall be executed. The <causalConnector> element shall 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 shall 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 may 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.

Table 3.17 – Reserved condition *role* values associated to event state machines¹⁴

Role Value	Transition Value	Event Type
<i>onBegin</i>	<i>starts</i>	<i>presentation</i>
<i>onEnd</i>	<i>stops</i>	<i>presentation</i>
<i>onAbort</i>	<i>aborts</i>	<i>presentation</i>
<i>onPause</i>	<i>pauses</i>	<i>presentation</i>
<i>onResume</i>	<i>resumes</i>	<i>presentation</i>
<i>onSelection</i>	<i>stops</i>	<i>selection</i>
<i>onBeginSelection</i>	<i>starts</i>	<i>selection</i>
<i>onEndSelection</i>	<i>stops</i>	<i>selection</i>
<i>onAbortSelection</i>	<i>aborts</i>	<i>selection</i>
<i>onPauseSelection</i>	<i>pauses</i>	<i>selection</i>
<i>onResumeSelection</i>	<i>resumes</i>	<i>selection</i>
<i>onBeginAttribution</i>	<i>starts</i>	<i>attribution</i>
<i>onEndAttribution</i>	<i>stops</i>	<i>attribution</i>
<i>onAbortAttribution</i>	<i>aborts</i>	<i>Attribution</i>
<i>onPauseAttribution</i>	<i>pauses</i>	<i>Attribution</i>
<i>onResumeAttribution</i>	<i>resumes</i>	<i>Attribution</i>

¹⁴ Several reserved words are defined for condition role values. However, it should be noted that there are more values than those defined by the reserved words. Reserved words are only aliases and in the future new aliases can be added. For example, in the case of a selection event, it could be defined reserved words both for the beginning and for the end of a selection.

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. At least the following values (case sensitive) shall be accept for the *key* attribute: “0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”, “A”, “B”, “C”, “D”, “E”, “F”, “G”, “H”, “I”, “J”, “K”, “L”, “M”, “N”, “O”, “P”, “Q”, “R”, “S”, “T”, “U”, “V”, “W”, “X”, “Y”, “Z”, “*”, “#”, “MENU”, “INFO”, “GUIDE”, “CURSOR_DOWN”, “CURSOR_LEFT”, “CURSOR_RIGHT”, “CURSOR_UP”, “CHANNEL_DOWN”, “CHANNEL_UP”, “VOLUME_DOWN”, “VOLUME_UP”, “ENTER”, “RED”, “GREEN”, “YELLOW”, “BLUE”, “BACK”, “EXIT”, “POWER”, “REWIND”, “STOP”, “EJECT”, “PLAY”, “RECORD”, “PAUSE”.

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>. The minimal cardinality value shall always be a positive finite value, greater than zero and lesser than or equal to the maximal cardinality value. If minimal and maximal cardinalities are not informed, “1” shall be assumed as the default value for both parameters. 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 may be set to the *max* attribute, if the role may 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” shall be assumed.

Table 3.18 – Simple condition *qualifier* values

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

A *delay* attribute may 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”¹⁵ or “or”) relating its child elements: <simpleCondition>, <compoundCondition>, <assessmentStatement> and <compoundStatement>. A *delay* attribute may 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.

¹⁵ 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 shall be considered true in the instant immediately after all the trigger conditions are satisfied.

An action expression captures actions that may be executed in causal relations. It may be 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 may 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.

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

Role Value	Action Type	Event Type
<i>start</i>	<i>start</i>	<i>presentation</i>
<i>stop</i>	<i>stop</i>	<i>presentation</i>
<i>abort</i>	<i>abort</i>	<i>presentation</i>
<i>pause</i>	<i>pause</i>	<i>presentation</i>
<i>resume</i>	<i>resume</i>	<i>presentation</i>
<i>set</i>	<i>start</i>	<i>attribution</i>
<i>startAttribution</i>	<i>start</i>	<i>Attribution</i>
<i>stopAttribution</i>	<i>stop</i>	<i>Attribution</i>
<i>abortAttribution</i>	<i>abort</i>	<i>Attribution</i>
<i>pauseAttribution</i>	<i>pause</i>	<i>Attribution</i>
<i>resumeAttribution</i>	<i>resume</i>	<i>Attribution</i>

If an *eventType* value is “attribution”, the `<simpleAction>` shall also define the value that shall be assigned, through its *value* attribute. If this value cannot be retrieved, no attribution shall be made. If the *value* is specified as “\$anyName” (where \$ is a reserved symbol and anyName is any string, except reserved role names), the assigned value shall be retrieved from the property associated with the *role*=“anyName” and defined by a `<bind>` child element of the `<link>` element that refers the connector.¹⁶ Declaring the *role*=“anyName” attribute in a `<bind>` element of a `<link>` implies having a role implicitly declared as `attributeAssessment role=“anyName” eventType=“attribution” attributeType=“nodeProperty”/`. This is the only possible case of a `<bind>` element referring to a role that is not explicitly declared in a connector.

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 shall be specified. Table 3.20 presents possible qualifier values.

¹⁶ In the case that *value*=“\$anyName”, the value to be attributed shall be the value of a property (`<property>` element) of a component of the same composition where the link (`<link>` element) that refers to the event is defined, or a property of the composition where the link is defined, or a property of an element that can be reached through a `<port>` element of the composition where the link is defined, or even a property of an element that can be reached through a port (elements `<port>` or `<switchPort>`) of a composition nested in the same composition where the link is defined.

Table 3.20 – Action *qualifier* values

Role Element	Qualifier	Semantics
simpleAction	<i>par</i>	All actions shall be executed in parallel
simpleAction	<i>seq</i>	All actions shall be executed in the bind sequence

A *delay* attribute may also be defined for a <simpleAction> specifying that the action shall be fired only after waiting for the specified time. Besides, the <simpleAction> may 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.

Besides all aforementioned attributes, the <simpleAction> element may also have attributes defined in the Animation Functionality (*duration* and *by* attributes), if its *eventType* value is “attribution”, as specified in Section 3.2.12.

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 shall be performed in any order or in a specific order¹⁷, respectively. A *delay* attribute may also be defined specifying that the compound action shall 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 “nodeProperty” (default), “occurrences”, “repetition” or “state”. In the first case, the event represents a node property to be evaluated; in the other ones the event represents the evaluation of the corresponding attribution-event attribute (“occurrences” or “repetition”) or the attribution-event state. An *offset* value may 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”.

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

¹⁷ 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 <assessmentStatement> element has a *comparator* attribute that compares the values inferred from its child elements (<attributeAssessment> element and <valueAssessment> element).

- a) In the case of <attributeAssessment>: a node property value [*eventType* = “attribution” and the *attributeType* = “nodeProperty”]; 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 *comparator* attribute shall have one of the values: “eq”, “ne”, “gt”, “lt”, “gte”, or “lte”.

The <compoundStatement> element has a Boolean *operator* attribute (“and” or “or”) relating its child elements: <assessmentStatement> or <compoundStatement>. An *isNegated* attribute may also be defined to specify that the <compoundStatement> child element shall 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.

The ConnectorBase module defines an element named <connectorBase>, which allows for 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 non-expert users, the idea is to have expert users defining connectors, storing them in libraries (connector bases) that may be imported, and making them available to others for creating links. Appendix E gives an extensive example of connector definitions that may be imported.

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

Table 3.21 – Extended CausalConnectorFunctionality module in the Enhanced DTV profile

Elements	Attributes	Content
causalConnector	<i><u>id</u></i>	(connectorParam*, (simpleCondition compoundCondition), (simpleAction compoundAction))
connectorParam	<i><u>name</u>, type</i>	empty
simpleCondition	<i><u>role</u>, delay, eventType, key, transition, min, max, qualifier</i>	empty
compoundCondition	<i><u>operator</u>, delay</i>	((simpleCondition compoundCondition)+, (assessmentStatement compoundStatement)*)
simpleAction	<i><u>role</u>, delay, eventType, actionType, value, min, max, qualifier, repeat, repeatDelay, duration, by</i>	empty
compoundAction	<i><u>operator</u>, delay</i>	(simpleAction compoundAction)+
assessmentStatement	<i><u>comparator</u></i>	(attributeAssessment, (attributeAssessment valueAssessment))
attributeAssessment	<i><u>role</u>, eventType, key, attributeType, offset</i>	empty
valueAssessment	<i><u>value</u></i>	empty
compoundStatement	<i><u>operator</u>, isNegated</i>	(assessmentStatement compoundStatement)+

Table 3.22 – Extended ConnectorBase module in the Enhanced DTV profile

Elements	Attributes	Content
connectorBase	<i><u>id</u></i>	(importBaseIcausalConnector)*

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 for 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 are useful for defining either alternative components (nodes) or alternative descriptors.

The <ruleBase> element specifies a set of rules, and shall be defined as a child element of the <head> element. These rules may be simple, defined by the <rule> element, or compound, defined by the <compositeRule> element. Simple rules define the optional identifier (*id* attribute), a variable (*var* attribute), a value (*value* attribute), and a comparator (*comparator* attribute) relating the variable to the value. The variable shall be a property of the settings node (<media> element of application/x-ncl-settings type), that is, the *var* attribute shall have the same value of a *name* attribute of a <property>element, defined as a child of the <media> element of application/x-ncl-settings type. The *comparator* attribute shall have one of the values: “eq”, “ne”, “gt”, “lt”, “gte”, or “lte”. Composite rules have the 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> and <compositeRule> 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	<i>id</i>	(importBase rule compositeRule)+
rule	<i>id</i> , <i>var</i> , <i>comparator</i> , <i>value</i>	empty
compositeRule	<i>id</i> , <i>operator</i>	(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	<u>constituent</u> , <u>rule</u>	empty

The ContentControl module specifies the <switch> element, allowing for the definition of alternative nodes (document components) to be chosen during presentation time. Test rules used to choose the switch's child-component to be presented are defined by the TestRule module, as previously mentioned. The ContentControl module also defines the <defaultComponent> element, whose *component* attribute (also of IDREF type) identifies the default element that shall 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 for the definition of special interfaces, named <switchPort>.

A <switchPort> child element may define a mapping to a subset of the switch's child components. When a link is bound to a <switchPort> element, the rules bound to mapped components are evaluated in the order they are defined. The first rule evaluated as true determines the chosen object. If all rules bound to mapped components are evaluated as false, a default component, defined by the <defaultComponent> child element, is chosen; if the <defaultComponent> element is not defined, no component is selected for presentation and the NCL player behaves as if the selected component does not exist.

All references to a switch's child component are made through <switchPort> elements or, by default, to the <switch> element without specifying any <switchPort>. In this case, it is considered as if the reference is made to a default <switchPort> that contains mapping elements to each child object of the switch and referring to its whole content anchor.

As usual, <switch> elements shall 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> shall 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 shall 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	<i><u>id</u></i> , <i>refer</i>	defaultComponent?, (switchPort bindRule media context switch)*)
defaultComponent	<i><u>component</u></i>	empty

The DescriptorControl module specifies the <descriptorSwitch> element, which contains a set of alternative descriptors to be associated with an object. The <descriptorSwitch> elements shall 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. The DescriptorControl module also defines the <defaultDescriptor> element, whose *descriptor* attribute (also of IDREF type) identifies the default element that shall 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	<i><u>id</u></i>	(defaultDescriptor?, (bindRule descriptor)*)
defaultDescriptor	<i><u>descriptor</u></i>	empty

During a document presentation, from the moment on a <switch> is evaluated, it is considered resolved until the end of its current 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.¹⁸

¹⁸ 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.

3.2.9. Timing Functionality

The Timing functionality defines the Timing module. The Timing module allows for the definition of temporal attributes 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*).

3.2.10. Reuse Functionality

NCL allows for 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 shall 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*. The import operation is transitive, that is, if *baseA* imports *baseB* that imports *baseC*, then *baseA* imports *baseC*. However, the *alias* defined for *baseC* inside *baseB* shall not be considered by *baseA*.

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

- 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. As the referred document URI can have more than one region base, the base to be imported must be identified by assigning its *id* to the *baseId* attribute. On importing a `<regionBase>`, an optional attribute, named *region*, may be specified within the `<importBase>` element. When present, the attribute shall identify the *id* of a `<region>` element declared in a `<regionBase>` element of the host document (the document that did the importing operation). As a consequence, all child `<region>` elements

of the imported <regionBase> shall 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> shall be considered children of the host document <regionBase> element.

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

An NCL document may 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. It should be stressed that the <importNCL> element does not “include” the referred NCL document but only makes the referred document visible to have its components reused by the document that has defined the <importNCL> element. Thus, imported <body>, as well as any of its contained nodes, may be reused inside the <body> element of the importing NCL document.

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 with the <importBase> element, the name shall 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*. 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* shall 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	<u><i>alias</i></u> , <u><i>documentURI</i></u> , <i>region</i> , <i>baseId</i>	empty
importedDocumentBase	<i>id</i>	(importNCL)+
importNCL	<u><i>alias</i></u> , <u><i>documentURI</i></u>	empty

The EntityReuse module allows for an NCL element to be reused. This module defines the *refer* attribute, which refers to an element *id*¹⁹ that will be reused. Only <media>, <context>, <body> and <switch> may be reused. An element that refers to another element cannot be reused; that is, its *id* cannot be the value of any *refer* attribute.

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

When an element declares the *refer* attribute, all attributes and child elements defined by the referred element are inherited. All attributes and child elements defined by the referring element shall be ignored by the NCL formatter, except the *id* attribute that shall be defined. The only other exception is for <media> elements, in which new child <area> and <property> elements may be added, and a new attribute, *instance*, may be defined. If the new added <property> element has the same *name* attribute of an already existing <property> element (defined in the reused <media> element), the new added <property> shall be ignored. Similarly, if the new added <area> element has the same *id* attribute of an already existent <area> element (defined in the reused <media> element), the new added <area> shall be ignored. The *instance* attribute is defined in the ExtendedEntityReuse module and has “new” as its default string value.

The referred element and the element that refers to it shall be considered the same, regarding its data specification. In other words it means that a single NCM node may be represented by more than one NCL element. As nodes contained in an NCM composite node defines a set, an NCM node may be represented by no more than one NCL element inside a composition²⁰. 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.

The referred element and the element that refers to it shall also be considered the same regarding their presentation, if the *instance* attribute receives the “instSame” or the “gradSame” value. Therefore, the following semantics shall be respected. Assume the set of <media> elements composed of the referred <media> element and all the referring

¹⁹ If the referred node is defined within an imported document *D*, the *refer* attribute value shall have the format “alias#id”, where “alias” is the value of the *alias* attribute associated with the *D* import.

²⁰ 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* shall each be declared inside a different composition.

<media> elements; assume also the subset formed by the referred <media> element and all other <media> elements having the *instance* attribute equal to “instSame” or “gradSame”.

- If any element of the subset is scheduled to be presented, all other elements in the subset, which are not child descendants of a <switch> element, are also assumed to be scheduled for presenting, and more than that, when they are being presented, they shall be represented by the same presentation instance. Descendent elements of a <switch> element shall also have the same behavior if all rules needed to present them are satisfied; otherwise they shall not be scheduled for presenting. As each <media> element starts its presentation, the common instance in presentation shall notify events associated with its <area> and <property> elements.
- If the *instance* attribute is equal to “instSame”, all scheduled nodes of the subset shall be immediately presented through a unique instance (as if a start action is applied on all subset elements)²¹. If the *instance* attribute is equal to “gradSame”, all scheduled nodes of the subset shall be presented through a unique instance, but now gradually, as while start instructions are applied, coming from a link, etc.
- The <media> elements in the set that have *instance* attribute values equal to “new” shall not be scheduled for presenting. When they are individually scheduled for presenting, no other element in the set is affected. Moreover, new independent presentation instances shall be created at each individual presentation starting.

It must be stressed that when a composite node is referred, always a new instance of the composition is created. In the new created composition the child <media> elements maintain the same referring behavior of their counterpart in the referred composition.

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 may be incorporated by <descriptor> elements, or defined in <property> elements.

The *focusIndex* attribute specifies an index for the <media> element to which the focus may be applied, when this element is in exhibition. The *focusIndex* may be defined using a <property> or a <descriptor> element. When this property is not defined, the object 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 applied to the element that is being presented with smallest index value. Values of *focusIndex* attribute shall be unique in an NCL document. Otherwise, the repeated attributes will be ignored if in a certain moment there is more than one <media> element to gain the focus. Moreover, when a <media> element refers to another <media> element (using the *refer* attribute specified in Section 3.2.10), it shall ignore the *focusIndex* associated with the referred <media> element.

²¹ NCM compositionality is not affected by this behavior, if we assumed that there are “virtual” <port> elements in every composition mapping to interfaces that reach a nested element that pertains to the mentioned subset, and if we assumed “virtual” links binding every element in the subset, such that when an element’s interface starts its presentation, all other elements start the presentation of their same corresponding interface at the same time.

The *moveUp* attribute specifies a value equal to the *focusIndex* value associated to an element to which the focus should 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 should 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 should 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 should be applied when the “left arrow key” is pressed.

When the focus is applied to an element with the visible property set to false, or to an element that it is not being presented, the current focus does not move.

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

When an element receives the focus, the square box defined by the element positioning attributes shall be decorated. The *focusBorderColor* attribute defines the decorative color and may 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* shall be a real value between 0 and 1, or a real value in the range [0,100] ending with the character “%” (e.g. 30%), with “1” or “100%” meaning full transparency and “0” or “0%” meaning no transparency. When the *focusBorderColor*, the *focusBorderWidth*, or the *focusBorderTransparency* are not defined, default values shall be assumed. These values are specified in properties of the <media> element of application/x-ncl-settings type: *default.focusBorderColor*, *default.focusBorderWidth*, *default.focusTransparency*, respectively.

When an element on focus is selected by pressing the activation (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> element. When selected, the square box defined by the element positioning attributes shall be decorated with the color defined by the *selBorderColor* attribute (default value specified by the *default.selBorderColor* of the <media> element of application/x-ncl-settings type), 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 shall be passed to the <media> element renderer (player). The player may then follow its own rules for navigation. The focus control shall be passed back to the NCL formatter when the “back key” is pressed²². In this case, the focus goes to the element

²² The focus control can also be passed by setting the service.currentKeyMaster attribute of the *settings* node (<media> element of application/x-ginga-settings type). This can be done through a link action, through an

identified by the the *service.currentFocus* attribute of the *settings* node (<media> element of application/x-ncl-settings type).

3.2.12. Animation Functionality

Animation in the cartoon sense is actually a combination of two factors: support for object drawing and support for object motion — or more precisely, support for object alteration as a function of time.

NCL is not a content format and, as such, does not have support for creating media objects and it does not have a generalized method for altering media object content. Instead, NCL is a scheduling and orchestration format. This means that NCL cannot be used to make cartoons, but can be used to render cartoon objects in the context of a general presentation, and to change the timing and rendering properties of a cartoon (or any other) object as a whole, while it is being displayed.

The animation primitives of NCL allow values of node properties to be changed during an explicitly declared duration. Since NCL animation can be computationally intensive, only the properties that define numerical values and colors may be animated.

The Animation Functionality defines the Animation module that provides the extensions necessary to describe what happens when a node property value is changed. Basically, the module defines attributes that may be incorporated to <simpleAction> elements of a connector), if its *eventType* value is “attribution”. Two new attributes are defined: *duration* and *by*.

When setting a new value to a property the change is instantaneous by default (*duration*=“0”), but the change may also be carried out during an explicitly declared duration, specified by the *duration* attribute.

Also, when setting a new value to a property the change from the old value to the new one may be linear by default (*by*=“indefinite”), or carried out step by step, with the pace specified by the *by* attribute.

The combination of the *duration* and *by* attribute definitions gives how (discretely or linearly) the change shall be performed and its transforming interval.

3.2.13. The Transition Effects functionality

The Transition Effects functionality is divided into two modules: TransitionBase and Transition.

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

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

NCL editing command executed by an imperative-code node (NCLua or NCLet object, for example). The player of a node that has the current control cannot directly change the *service.currentKeyMaster* property.

Table 3.28 – Extended TransitionBase Module in the Enhanced DTV profile

Elements	Attributes	Content
transitionBase	<i>id</i>	(importBase, transition)+

The Transition module is based on SMIL 2.1 specifications [SMIL05]. It has just one element called <transition>.

In NCL 3.0 Enhanced DTV profile, the <transition> element is specified in the <transitionBase> element and allows for a transition template to be defined. Each <transition> element defines a single transition template and shall have the *id* attribute so that it may be referred.

Seven <transition> element attributes come from SMIL BasicTransitions module specification: *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, shall 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 subtypes for the five required transition types listed in Table 3.29 shall be supported, the others, defined in SMIL specifications [SMIL 2.1 Specification, 2005], are optional.

Table 3.29 – Required transition types and subtypes.

Transition type	Default Transition subtype
barWipe	leftToRight
irisWipe	rectangle
clockWipe	clockwiseTwelve
snakeWipe	topLeftHorizontal
fade	crossfade (fade for audio)

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

can want to begin a crossfade with the destination image being already 30% faded in. For this case, *startProgress* would be 0.3. 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 shall 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” (values that are not required to be supported in a Ginga implementation), then the *fadeColor* attribute specifies the ending or starting 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 shall be ignored. The default value is “black”.

The Transition 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 shall be performed. The properties can also be defined using <property> elements.

The value of *transIn* and *transOut* attributes is a semicolon-separated list of transition identifiers. Each of the identifiers shall 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* attribute or the *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 transitions defined in Transition module accept four additional attributes (coming from the SMIL TransitionModifiers module specification) that may be used to control the visual appearance of the transitions. The *horzRepeat* 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 Transition Module, its child elements, and its attributes.

Table 3.30 – Extended Transition module in the Enhanced DTV profile

Elements	Attributes	Content
transition	<i>id, type, subtype, dur, startProgress, endProgress, direction, fadeColor, horzRepeat, vertRepeat, borderWidth, borderColor</i>	empty

3.2.14. The Metainformation Functionality

Metainformation does not contain content information that is used or displayed by the NCL formatter during a presentation. Instead, it contains information about content that is used or displayed. The Metainformation Functionality is composed of the Metainformation module that comes from SMIL Metainformation module specification.

The Metainformation module contains two elements that allow for the 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 metainformation of the document. It acts as the root element of the RDF tree. The <metadata> element may have as child elements: RDF element and its sub-elements (refers 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	<i><u>name</u>, <u>content</u></i>	empty
metadata	<i>empty</i>	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 may 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), (l) 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	<u>id</u> , title, xmlns	(head?, body?)
head		(importedDocumentBase? ruleBase?, regionBase*, descriptorBase?, connectorBase?),
body	id	(portl propertyl medialcontextlswitchllink)*

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

Elements	Attributes	Content
regionBase	id, device, region	(importBase region)+
region	<u>id</u> , 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	<u>id</u> , src, refer, instance, type, descriptor	(area property)*

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

Elements	Attributes	Content
context	<i><u>id</u></i> , <i>refer</i>	(port property media context link switch)*

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

Elements	Attributes	Content
area	<i><u>id</u></i> , <i>coords</i> , <i>begin</i> , <i>end</i> , <i>beginText</i> , <i>beginPosition</i> , <i>endText</i> , <i>endPosition</i> , <i>first</i> , <i>last</i> , <i>label</i> , <i>clip</i>	empty

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

Elements	Attributes	Content
port	<i><u>id</u></i> , <i><u>component</u></i> , <i>interface</i>	empty

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

Elements	Attributes	Content
property	<i><u>name</u></i> , <i>value</i>	empty

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

Elements	Attributes	Content
switchPort	<i><u>id</u></i>	mapping+
mapping	<i><u>component</u></i> , <i>interface</i>	empty

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

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

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

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

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

Elements	Attributes	Content
causalConnector	<u>id</u>	(connectorParam*, (simpleCondition compoundCondition), (simpleAction compoundAction))
connectorParam	<u>name</u> , <i>type</i>	empty
simpleCondition	<i>role</i> , <i>delay</i> , <i>eventType</i> , <i>key</i> , <i>transition</i> , <i>min</i> , <i>max</i> , <i>qualifier</i>	empty
compoundCondition	<u>operator</u> , <i>delay</i>	((simpleCondition compoundCondition)+, (assessmentStatement compoundStatement)*)
simpleAction	<i>role</i> , <i>delay</i> , <i>eventType</i> , <i>actionType</i> , <i>value</i> , <i>min</i> , <i>max</i> , <i>qualifier</i> , <i>repeat</i> , <i>repeatDelay</i>	empty
compoundAction	<u>operator</u> , <i>delay</i>	(simpleAction compoundAction)+
assessmentStatement	<u>comparator</u>	(attributeAssessment, (attributeAssessment valueAssessment))
attributeAssessment	<i>role</i> , <i>eventType</i> , <i>key</i> , <i>attributeType</i> , <i>offset</i>	empty
valueAssessment	<u>value</u>	empty
compoundStatement	<u>operator</u> , <i>isNegated</i>	(assessmentStatement compoundStatement)+

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

Elements	Attributes	Content
connectorBase	<i>id</i>	(importBaseIcausalConnector)*

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

Elements	Attributes	Content
ruleBase	<i>id</i>	(importBaseIruleIcompositeRule)+
rule	<i>id</i> , <u><i>var</i></u> , <u><i>comparator</i></u> , <u><i>value</i></u>	empty
compositeRule	<u><i>id</i></u> , <u><i>operator</i></u>	(rule compositeRule)+

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

Elements	Attributes	Content
bindRule	<u><i>constituent</i></u> , <u><i>rule</i></u>	empty

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

Elements	Attributes	Content
switch	<u><i>id</i></u> , <i>refer</i>	(defaultComponent?,(switchPortI bindRuleImediaI context switch)*)
defaultComponent	<u><i>component</i></u>	empty

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

Elements	Attributes	Content
descriptorSwitch	<i>id</i>	(defaultDescriptor?, (bindRule descriptor)*)
defaultDescriptor	<i><u>descriptor</u></i>	empty

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

Elements	Attributes	Content
importBase	<i><u>alias</u>, <u>documentURI</u>, <u>region</u></i>	empty
importedDocumentBase	<i>id</i>	(importNCL)+
importNCL	<i><u>alias</u>, <u>documentURI</u></i>	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="device" type="string" use="optional"/>
    <attribute name="region" 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="beginText" type="string" use="optional"/>
    <attribute name="beginPosition" type="unsignedLong" use="optional"/>
    <attribute name="endText" type="string" use="optional"/>
    <attribute name="endPosition" type="unsignedLong" use="optional"/>
  </attributeGroup>

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

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

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

  <!-- define the clipAttrs attribute group -->
  <attributeGroup name="clipAttrs">
    <attribute name="clip" 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:labelAttrs" />
  </complexType>
</schema>
```

```
<attributeGroup ref="mediaAnchor:clipAttrs" />
</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>
```

PropertyAnchor Module: NCL30PropertyAnchor.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/NCL30PropertyAnchor.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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

  <complexType name="propertyAnchorPrototype">
    <attribute name="name" type="string" use="required" />
    <attribute name="value" type="string" use="optional" />
    <attribute name="externable" type="boolean" use="optional" />
  </complexType>

  <!--
  The following reserved words are used for properties' names.
  * For media-objects in general: player; reusePlayer, which determines if a new
  player shall be instantiated or if a player already instantiated shall be used;
playerLife, which specifies what will happen to the player instance at the end of
  the presentation; the focus movement parameters: moveLeft, moveRight, moveUp,
moveDown, focusIndex; the timing parameter explicitDur; style, which refers to a
  style sheet with information for content presentation; and the multiple device
  parameter deviceClass, which defines the device class where the media object will be
  presented.
  * For visual media-objects: 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 shall be between 0 and 1, or a real value in the range
  [0,100] ending with the character "%" (e.g. 30%)); visible, indicating if the
  presentation is to be seen or hidden; rgbChromaKey; the object positioning
  parameters: top, left, bottom, right, width, height, zIndex, plan, location, size
  and bounds; the other related focus parameters: focusBorderColor, selBorderColor,
focusBorderWidth, focusBorderTransparency, focusSrc, and focusSelSrc; the
  transition parameters: transIn and transOut; and the timing parameter: freeze.
  * For media-objects with continuous media content: contentId and standby.
  * For audio media-objects: soundLevel; balanceLevel; trebleLevel; bassLevel; the
  transition parameters: transIn and transOut; and the timing parameter: freeze.
  * For text media-objects: textAlign; fontColor; fontFamily; fontStyle; fontSize;
fontVariant; fontWeight; the transition parameters: transIn and transOut; and the
  timing parameter: freeze.
  -->

  <!-- declare global elements in this module -->
  <element name="property" type="propertyAnchor:propertyAnchorPrototype"/>

</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 players in general: player; reusePlayer, which determines if a new player
  shall be instantiated or if a player already instantiated shall be used;
playerLife, which specifies what will happen to the player instance at the end of
  the presentation; the focus movement parameters: moveLeft, moveRight, moveUp,
moveDown, focusIndex; the timing parameter explicitDur; style, which refers to a
  style sheet with information for content presentation; and the multiple device
  parameter deviceClass, which defines the device class where the media object will be
  presented.
  * For visual 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 shall
  be between 0 and 1, or a real value in the range [0,100] ending with the
  character "%" (e.g. 30%)); visible, indicating if the presentation is to be seen
  or hidden; rgbChromakey; the object positioning parameters: top, left, bottom,
right, width, height, zIndex, plan, location, size and bounds; the other related
  focus parameters: focusBorderColor, selBorderColor, focusBorderWidth,
focusBorderTransparency, focusSrc, and focusSelSrc; the transition parameters:
transIn and transOut; and the timing parameter: freeze.
  * For audio players: soundLevel; balanceLevel; trebleLevel; bassLevel; the
  transition parameters: transIn and transOut; and the timing parameter: freeze.
  * For text players: textAlign; fontColor; fontFamily; fontStyle; fontSize;
fontVariant; fontWeight; the transition parameters: transIn and transOut; and the
  timing parameter: freeze.
  -->
  <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" />
    </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/ConnectorAssessmentExpression"
  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="nodeProperty" />
  </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" default="false"/>
</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" />
  </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" default="0s"/>
    <attribute name="min" type="positiveInteger" use="optional" default="1"/>
    <attribute name="max" type="connectorCausalExpression:maxUnion" use="optional"
default="1"/>
    <attribute name="qualifier" type="connectorCommonPart:logicalOperatorPrototype"
use="optional" default="or"/>
</complexType>

<complexType name="compoundConditionPrototype">
    <attribute name="operator" type="connectorCommonPart:logicalOperatorPrototype"
use="required"/>
    <attribute name="delay" type="string" use="optional" default="0s"/>
</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" default="0s"/>
    <attribute name="value" type="string" use="optional"/>
    <attribute name="repeat" type="positiveInteger" use="optional"/>
    <attribute name="repeatDelay" type="string" use="optional" default="0s"/>
    <attribute name="min" type="positiveInteger" use="optional" default="1"/>
    <attribute name="max" type="connectorCausalExpression:maxUnion" use="optional"
default="1"/>
    <attribute name="qualifier"
type="connectorCausalExpression:actionOperatorPrototype" use="optional"
default="par"/>
</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" default="0s"/>
</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="optional"/>
    <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="IDREF" 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="required"/>
    <attribute name="region" type="IDREF" use="optional"/>
    <attribute name="documentURI" type="anyURI" use="required"/>
    <attribute name="baseId" type="IDREF" use="optional"/>
  </complexType>

  <complexType name="importNCLPrototype">
    <attribute name="alias" type="ID" use="required"/>
    <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="instance" type="string" 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="positiveInteger" use="optional"/>
    <attribute name="moveRight" type="positiveInteger" use="optional"/>
    <attribute name="moveUp" type="positiveInteger" use="optional"/>
    <attribute name="moveDown" type="positiveInteger" use="optional"/>
    <attribute name="focusIndex" type="positiveInteger" 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="focusSrc" type="anyURI" use="optional"/>
    <attribute name="focusSelSrc" type="anyURI" 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>
```

Animation Module: NCL30Animation.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/NCL30Animation.xsd  
Author: TeleMidia Laboratory  
Revision: 19/09/2006  
  
Schema for the NCL Timing module namespace.  
-->  
<schema xmlns="http://www.w3.org/2001/XMLSchema"  
  xmlns:animation="http://www.ncl.org.br/NCL3.0/Animation"  
  targetNamespace="http://www.ncl.org.br/NCL3.0/Animation"  
  elementFormDefault="qualified" attributeFormDefault="unqualified" >  
  
  <!-- declare global attributes in this module -->  
  
  <!-- define the animation attribute group -->  
  <attributeGroup name="animationAttrs">  
    <attribute name="duration" type="string" use="optional" default="0s"/>  
    <attribute name="by" type="string" use="optional" default="indefinite"/>  
  </attributeGroup>  
  
</schema>
```

Transition module: NCL30Transition.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/NCL30Transition.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

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

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

  <!-- define the type attribute prototype -->
  <simpleType name="typePrototype">
    <restriction base="string">
      <enumeration value="in"/>
      <enumeration value="barWipe"/>
      <enumeration value="boxWipe"/>
      <enumeration value="fourBoxWipe"/>
      <enumeration value="barnDoorWipe"/>
      <enumeration value="diagonalWipe"/>
      <enumeration value="bowTieWipe"/>
      <enumeration value="miscDiagonalWipe"/>
      <enumeration value="veeWipe"/>
      <enumeration value="barnVeeWipe"/>
      <enumeration value="zigZagWipe"/>
      <enumeration value="barnZigZagWipe"/>
      <enumeration value="irisWipe"/>
      <enumeration value="triangleWipe"/>
      <enumeration value="arrowHeadWipe"/>
      <enumeration value="pentagonWipe"/>
      <enumeration value="hexagonWipe"/>
      <enumeration value="ellipseWipe"/>
      <enumeration value="eyeWipe"/>
      <enumeration value="roundRectWipe"/>
      <enumeration value="starWipe"/>
      <enumeration value="miscShapeWipe"/>
      <enumeration value="clockWipe"/>
      <enumeration value="pinWheelWipe"/>
      <enumeration value="singleSweepWipe"/>
      <enumeration value="fanWipe"/>
      <enumeration value="doubleFanWipe"/>
      <enumeration value="doubleSweepWipe"/>
      <enumeration value="saloonDoorWipe"/>
      <enumeration value="windshieldWipe"/>
      <enumeration value="snakeWipe"/>
      <enumeration value="spiralWipe"/>
      <enumeration value="parallelSnakesWipe"/>
      <enumeration value="boxSnakesWipe"/>
      <enumeration value="waterfallWipe"/>
      <enumeration value="pushWipe"/>
      <enumeration value="slideWipe"/>
      <enumeration value="fade"/>
    </restriction>
  </simpleType>
</schema>
```

```

        <enumeration value="audioFade"/>
        <enumeration value="audioVisualFade"/>
    </restriction>
</simpleType>

<!-- define subType attribute prototype-->
<simpleType name="subTypePrototype">
    <restriction base="string">
        <enumeration value="bottom"/>
        <enumeration value="bottomCenter"/>
        <enumeration value="bottomLeft"/>
        <enumeration value="bottomLeftClockwise"/>
        <enumeration value="bottomLeftCounterClockwise"/>
        <enumeration value="bottomLeftDiagonal"/>
        <enumeration value="bottomRight"/>
        <enumeration value="bottomRightClockwise"/>
        <enumeration value="bottomRightCounterClockwise"/>
        <enumeration value="bottomRightDiagonal"/>
        <enumeration value="centerRight"/>
        <enumeration value="centerTop"/>
        <enumeration value="circle"/>
        <enumeration value="clockwiseBottom"/>
        <enumeration value="clockwiseBottomRight"/>
        <enumeration value="clockwiseLeft"/>
        <enumeration value="clockwiseNine"/>
        <enumeration value="clockwiseRight"/>
        <enumeration value="clockwiseSix"/>
        <enumeration value="clockwiseThree"/>
        <enumeration value="clockwiseTop"/>
        <enumeration value="clockwiseTopLeft"/>
        <enumeration value="clockwiseTwelve"/>
        <enumeration value="cornersIn"/>
        <enumeration value="cornersOut"/>
        <enumeration value="counterClockwiseBottomLeft"/>
        <enumeration value="counterClockwiseTopRight"/>
        <enumeration value="crossfade"/>
        <enumeration value="diagonalBottomLeft"/>
        <enumeration value="diagonalBottomLeftOpposite"/>
        <enumeration value="diagonalTopLeft"/>
        <enumeration value="diagonalTopLeftOpposite"/>
        <enumeration value="diamond"/>
        <enumeration value="doubleBarnDoor"/>
        <enumeration value="doubleDiamond"/>
        <enumeration value="down"/>
        <enumeration value="fadeFromColor"/>
        <enumeration value="fadeToColor"/>
        <enumeration value="fanInHorizontal"/>
        <enumeration value="fanInVertical"/>
        <enumeration value="fanOutHorizontal"/>
        <enumeration value="fanOutVertical"/>
        <enumeration value="fivePoint"/>
        <enumeration value="fourBlade"/>
        <enumeration value="fourBoxHorizontal"/>
        <enumeration value="fourBoxVertical"/>
        <enumeration value="fourPoint"/>
        <enumeration value="fromBottom"/>
        <enumeration value="fromLeft"/>
        <enumeration value="fromRight"/>
        <enumeration value="fromTop"/>
        <enumeration value="heart"/>
        <enumeration value="horizontal"/>
        <enumeration value="horizontalLeft"/>
        <enumeration value="horizontalLeftSame"/>
        <enumeration value="horizontalRight"/>
        <enumeration value="horizontalRightSame"/>
        <enumeration value="horizontalTopLeftOpposite"/>
        <enumeration value="horizontalTopRightOpposite"/>
    </restriction>
</simpleType>

```

```

    <enumeration value="keyhole"/>
    <enumeration value="left"/>
    <enumeration value="leftCenter"/>
    <enumeration value="leftToRight"/>
    <enumeration value="oppositeHorizontal"/>
    <enumeration value="oppositeVertical"/>
    <enumeration value="parallelDiagonal"/>
    <enumeration value="parallelDiagonalBottomLeft"/>
    <enumeration value="parallelDiagonalTopLeft"/>
    <enumeration value="parallelVertical"/>
    <enumeration value="rectangle"/>
    <enumeration value="right"/>
    <enumeration value="rightCenter"/>
    <enumeration value="sixPoint"/>
    <enumeration value="top"/>
    <enumeration value="topCenter"/>
    <enumeration value="topLeft"/>
    <enumeration value="topLeftClockwise"/>
    <enumeration value="topLeftCounterClockwise"/>
    <enumeration value="topLeftDiagonal"/>
    <enumeration value="topLeftHorizontal"/>
    <enumeration value="topLeftVertical"/>
    <enumeration value="topRight"/>
    <enumeration value="topRightClockwise"/>
    <enumeration value="topRightCounterClockwise"/>
    <enumeration value="topRightDiagonal"/>
    <enumeration value="topToBottom"/>
    <enumeration value="twoBladeHorizontal"/>
    <enumeration value="twoBladeVertical"/>
    <enumeration value="twoBoxBottom"/>
    <enumeration value="twoBoxLeft"/>
    <enumeration value="twoBoxRight"/>
    <enumeration value="twoBoxTop"/>
    <enumeration value="up"/>
    <enumeration value="vertical"/>
    <enumeration value="verticalBottomLeftOpposite"/>
    <enumeration value="verticalBottomSame"/>
    <enumeration value="verticalLeft"/>
    <enumeration value="verticalRight"/>
    <enumeration value="verticalTopLeftOpposite"/>
    <enumeration value="verticalTopSame"/>
  </restriction>
</simpleType>

<attributeGroup name="transAttrs">
  <attribute name="transIn" type="string" use="optional"/>
  <attribute name="transOut" type="string" use="optional"/>
</attributeGroup>

<!-- define the transition attribute group -->
<attributeGroup name="transitionAttrs">
  <attribute name="type" type="transition:typePrototype" use="required"/>
  <attribute name="subtype" type="transition:subTypePrototype" use="optional"/>
  <attribute name="fadeColor" type="string" use="optional" default="black"/>
  <attribute name="dur" type="string" use="optional"/>
  <attribute name="startProgress" use="optional" default="0.0">
    <simpleType>
      <restriction base="decimal">
        <minInclusive value="0.0"/>
        <maxInclusive value="1.0"/>
      </restriction>
    </simpleType>
  </attribute>
  <attribute name="endProgress" use="optional" default="1.0">
    <simpleType>
      <restriction base="decimal">
        <minInclusive value="0.0"/>

```

```

        <maxInclusive value="1.0"/>
    </restriction>
</simpleType>
</attribute>
<attribute name="direction" use="optional" default="forward">
    <simpleType>
        <restriction base="string">
            <enumeration value="forward"/>
            <enumeration value="reverse"/>
        </restriction>
    </simpleType>
</attribute>
</attributeGroup>

<!-- define the transition-modifier attribute group -->
<attributeGroup name="transitionModifierAttrs">
    <attribute name="horzRepeat" type="decimal" use="optional" default="1.0"/>
    <attribute name="vertRepeat" type="decimal" use="optional" default="1.0"/>
    <attribute name="borderWidth" type="nonNegativeInteger" use="optional"
default="0"/>
    <attribute name="borderColor" type="string" use="optional" default="black"/>
</attributeGroup>

<complexType name="transitionPrototype">
    <attribute name="id" type="ID" use="required"/>
    <attributeGroup ref="transition:transitionAttrs"/>
    <attributeGroup ref="transition:transitionModifierAttrs"/>
</complexType>

<!-- declare global element in this module -->
<element name="transition" type="transition:transitionPrototype"/>

</schema>

```

Metainformation module: NCL30Metainformation.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/NCL30Metainformation.xsd  
Author: TeleMidia Laboratory  
Revision: 19/09/2006  
  
Schema for the NCL Metainformation module namespace.  
-->  
<schema xmlns="http://www.w3.org/2001/XMLSchema"  
  xmlns:metainformation="http://www.ncl.org.br/NCL3.0/Metainformation"  
  targetNamespace="http://www.ncl.org.br/NCL3.0/Metainformation"  
  elementFormDefault="qualified" attributeFormDefault="unqualified" >  
  
  <complexType name="metaPrototype">  
    <attribute name="name" type="string" use="required"/>  
    <attribute name="content" type="string" use="required"/>  
  </complexType>  
  
  <complexType name="metadataPrototype">  
    <sequence>  
      <any minOccurs="0"/>  
    </sequence>  
  </complexType>  
  
  <!-- declare global elements in this module -->  
  <element name="meta" type="metainformation:metaPrototype"/>  
  
  <!-- declare global elements in this module -->  
  <element name="metadata" type="metainformation:metadataPrototype"/>  
  
</schema>
```

5. NCL 3.0 Language Profiles for Digital TV

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

Any document in conformance with NCL profiles shall 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, PropertyAnchor, SwitchInterface, Descriptor, Linking, CausalConnectorFunctionality, ConnectorBase, TestRule, TestRuleUse, ContentControl, DescriptorControl, Timing, Import, EntityReuse, ExtendedEntityReuse, KeyNavigation, Animation, TransitionBase, Transition and Metainformation modules of NCL 3.0.

The tables of Section 3.2 show each module element, already extended with 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, PropertyAnchor, 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 with 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:animation="http://www.ncl.org.br/NCL3.0/Animation"
  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:connectorCausalExpression="http://www.ncl.org.br/NCL3.0/
ConnectorCausalExpression"
  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:propertyAnchor="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  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.ncl.org.br/NCL3.0/Metainformation"
  xmlns:transition="http://www.ncl.org.br/NCL3.0/Transition"
  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/Animation"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Animation.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/ConnectorCausalExpression"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorCausalExpression.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/PropertyAnchor"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30PropertyAnchor.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.ncl.org.br/NCL3.0/Metainformation"
      schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Metainformation.xsd"/>
    <import namespace="http://www.ncl.org.br/NCL3.0/Transition"
      schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Transition.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"/>
        <element ref="profile:meta"/>
        <element ref="profile:metadata"/>
      </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:property"/>
  </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:property"/>
    </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"/>
          <element ref="profile:meta"/>
          <element ref="profile:metadata"/>
        </choice>
        <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"/>

  <!-- ===== -->
  <!-- PropertyAnchor -->
  <!-- ===== -->
  <!-- extends property element -->

  <complexType name="propertyAnchorType">
    <complexContent>
      <extension base="propertyAnchor:propertyAnchorPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="property" type="profile:propertyAnchorType"
substitutionGroup="propertyAnchor:property"/>

  <!-- ===== -->
  <!-- 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="transition:transAttrs"/>
    </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>

<complexType name="simpleActionType">
  <complexContent>
    <extension
base="connectorCausalExpression:simpleActionPrototype">
      <attributeGroup ref="animation:animationAttrs"/>
    </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" type="profile:simpleActionType"
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"/>

<!-- ===== -->
<!-- Transition -->
<!-- ===== -->

<element name="transition" substitutionGroup="transition: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: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:propertyAnchor="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  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/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/PropertyAnchor"
      schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30PropertyAnchor.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>

```

```

    </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:property"/>
    </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:property"/>
    </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="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"/>

<!-- ===== -->
<!-- PropertyAnchor -->
<!-- ===== -->
<!-- extends property element -->

<complexType name="propertyAnchorType">
  <complexContent>
    <extension base="propertyAnchor:propertyAnchorPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="property" type="profile:propertyAnchorType"
substitutionGroup="propertyAnchor:property"/>

```

```

<!-- ===== -->
<!-- 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 3.0 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 mister M. raises a battery. The second one is presented when a user interacts with an icon placed over mister M.'s 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 viewers the opportunity to initiate a transaction to buy the eyeglasses. Depending on the selected viewer 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 gingaTV element and to its default “whole content anchor” (since the gingaTV interface is not specified), indicating the image TV1920.jpg as the starting point. TV1920.jpg is only an image that simulates a 16:9 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. Every variable used in relationship specifications shall be explicitly declared in the application/x-ncl-settings type media element. In the example, the variable “shared.language” is explicitly declared as a property 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 mister M.'s eyeglasses at this time. The mtrx video definition has also its positioning and size properties explicitly declared, since they will be used when resizing the video. Again, every media property used in relationship specifications shall be explicitly declared. Note also that all media objects refer to 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 for 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="gingaTV"/>

43   <media id="gingaTV" type="image" descriptor="gingaDesc"
    src="example/TV1920.png"/>
44   <media type="application/x-ncl-settings" id="settings">
45     <property name="shared.language"/>
46   </media>

47   <media id="mtrx" type="video" descriptor="mtrxDesc"
    src="example/mtrx.mp4">
48     <area id="anchorBattery" begin="18s"/>
49     <area id="anchorGlasses" begin="27s" end="32s" />
50     <property name="left" />
51     <property name="top" />
52     <property name="height" />
53     <property name="width" />
54   </media>
55   <media id="battery" type="video" descriptor="batteryDesc"
    src="example/battery.mp4"/>

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.mp4"/>
61     <media id="glasses" type="video" descriptor="glassesDesc"
    src="example/glassesAd.mp4"/>

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 GingaTV) is specified, whose background is set to black. Another region (TVRegion) specifies the exhibition area (GingaTV 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="gingaRegion" title="Mtrx Example" top="0" left="0"
        height="1130" width="1940" zIndex="1">
06       <region id="TVRegion" top="30" left="10" height="1080" width="1920"
        zIndex="2">
07         <region id="mtrxRegion" left="0%" top="0%" height="100%"
            width="100%" zIndex="3"/>
08         <region id="pubBatteryRegion" left="68%" top="68%" height="32%"
            width="32%" zIndex="4"/>
09         <region id="glassesIconRegion" left="12%" top="24%" height="39"
            width="68" zIndex="4"/>
10         <region id="pubGlassesRegion" left="53%" top="43%" height="37%"
            width="47%" zIndex="4"/>
11         <region id="formRegion" left="53%" top="43%" height="37%"
            width="47%" zIndex="4"/>
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="gingaDesc" region="gingaRegion">
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="shared.language" op="eq" value="en" />
35     <rule id="rPt" var="shared.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 referred 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 TVGinga image presentation when the mtrx 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="gingaTV"/>

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#onBeginSet_varStart">
78     <bind role="onBegin" component="gingaTV"/>
79     <bind role="set" component="settings" interface="shared.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"
98       interface="glassesIconInt">
99       <bindParam name="keyCode" value="YELLOW"/>
100     </bind>
101     <bind role="stop" component="glassesPub" interface="glassesIconInt"/>
102     <bind role="setWidth" component="mtrx" interface="width">
103       <bindParam name="width" value="1018"/>
104     </bind>
105     <bind role="setHeight" component="mtrx" interface="height">
106       <bindParam name="height" value="509"/>
107     </bind>
108     <bind role="setLeft" component="mtrx" interface="left">
109       <bindParam name="left" value="0"/>
110     </bind>
111     <bind role="setTop" component="mtrx" interface="top" >
112       <bindParam name="top" value="20"/>
113     </bind>
114     <bind role="pause" component="mtrx"/>
115     <bind role="start" component="glassesPub" interface="glassesInt"/>
116   </link>

117   <link id="Link05" xconnector="itv#onEndResizeResume">
118     <bind role="onEnd" component="glassesPub" interface="glassesFormInt"/>
119     <bind role="setLeft" component="mtrx" interface="left">
120       <bindParam name="left" value="0"/>
121     </bind>
122     <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="1920"/>
126   </bind>
127   <bind role="setHeight" component="mtrx" interface="height">
128     <bindParam name="height" value="1080"/>
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="gingaTV"/>
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 shall be obeyed in the connector definition and are very important. The region is altered step by step, that is, attribute by attribute, by the NCL 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 properties should be used. In the case of Link04 and Link05, the property group named *bounds* is the most appropriate. When a formatter treats a property group it only tests the consistency at the end of the process.

The complete document specification, with the modifications in Link04, Link05 and in mtrx properties, 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="gingaRegion" title="Mtrx Example" top="0" left="0"
        height="1130" width="1940" zIndex="1"/>
      <region id="TVRegion" top="30" left="10" height="1080" width="1920"
        zIndex="2">
        <region id="mtrxRegion" left="0%" top="0%" height="1080" width="1920"
          zIndex="3"/>
        <region id="pubBatteryRegion" left="68%" top="68%" height="32%"
          width="32%" zIndex="4"/>
        <region id="glassesIconRegion" left="12%" top="24%" height="39"
          width="68" zIndex="4"/>
        <region id="pubGlassesRegion" left="53%" top="43%" height="37%"
          width="47%" zIndex="4"/>
        <region id="formRegion" left="53%" top="43%" height="37%" width="47%"
          zIndex="4"/>
      </region>
    </region>
  </regionBase>

  <descriptorBase>
    <descriptor id="gingaDesc" region="gingaRegion"/>
  </descriptorBase>

```

```

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

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

</head>

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

  <media id="gingaTV" type="image" descriptor="gingaDesc"
    src="example/TV800.png"/>
  <media type="application/x-ncl-settings" id="settings">
    <property name="shared.language"/>
  </media>

  <media id="mtrx" type="video" descriptor="mtrxDesc" src="example/mtrx.mp4">
    <area id="anchorBattery" begin="18s"/>
    <area id="anchorGlasses" begin="27s" end="32s" />
    <property name="bounds"/>
  </media>

  <media id="battery" type="video" descriptor="batteryDesc"
    src="example/battery.mp4"/>

  <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.mp4"/>
    <media id="glasses" type="video" descriptor="glassesDesc"
      src="example/glassesAd.mp4"/>

    <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>
  </context>

```

```

<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#onBeginSet_varStart">
  <bind role="onBegin" component="gingaTV"/>
  <bind role="set" component="setting" interface="shared.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#onKeySelectionStopSet_varPauseStart">
  <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="var" value="0,20,1018,509"/>
  </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="0,0,1920,1080"/>
  </bind>
  <bind role="resume" component="mtrx"/>
</link>

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

</body>
</ncl>

```

Figure 6.5 – Complete document specification

7. Final Remarks

In order to offer a scalable hypermedia model, with characteristics that may be progressively incorporated in hypermedia system implementations, NCM was divided in several parts, and also its declarative XML application language: NCL. This technical report deals with the declarative language for authoring documents for DTV environments, 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 them are Romualdo Rezende Costa, Carlos Salles Soares Netto, Marcio Ferreira Moreno and Simone Junqueira Barbosa.

References

- [Anto00] Antonacci M.J. NCL: Uma Linguagem Declarativa para Especificação de Documentos Hipermedia 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 Hipermedia na Web, **VI Simpósio Brasileiro de Sistemas Multimídia e Hipermedia - 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.
- [Soar09] Soares L.F.G. Nested Context Language 3.0: Part 11 - Declarative Hypermedia Objects in NCL: Nesting Objects with NCL Code in NCL Documents. **Technical Report, Departamento de Informática PUC-Rio**. Mars 2009, ISSN: 0103-9741.
- [SoSC08] Soares L.F.G.; Sant’Anna F.F.; Cerqueira, R.F.G. Nested Context Language 3.0: Part 10 – Imperative Objects in NCL. **Technical Report, Departamento de Informática PUC-Rio**. January 2008, ISSN: 0103-9741.
- [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] XHTMLTM 1.0 2^o 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 may 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" max="unbounded"/>  
      </causalConnector>  
  
      <causalConnector id="onBeginStart_delay">  
        <connectorParam name="delay"/>  
        <simpleCondition role="onBegin"/>  
        <simpleAction role="start" max="unbounded" delay="$delay"/>  
      </causalConnector>  
  
      <causalConnector id="onBeginStop">  
        <simpleCondition role="onBegin"/>  
        <simpleAction role="stop" max="unbounded"/>  
      </causalConnector>  
  
      <causalConnector id="onBeginStop_delay">  
        <connectorParam name="delay"/>  
        <simpleCondition role="onBegin"/>  
        <simpleAction role="stop" max="unbounded" delay="$delay"/>  
      </causalConnector>  
  
      <causalConnector id="onBeginPause">  
        <simpleCondition role="onBegin"/>  
        <simpleAction role="pause" max="unbounded"/>  
      </causalConnector>  
  
      <causalConnector id="onBeginPause_delay">  
        <connectorParam name="delay"/>  
        <simpleCondition role="onBegin"/>  
        <simpleAction role="pause" max="unbounded" delay="$delay"/>  
      </causalConnector>  
  
      <causalConnector id="onBeginResume">  
        <simpleCondition role="onBegin"/>  
        <simpleAction role="resume" max="unbounded"/>  
      </causalConnector>
```

```

<causalConnector id="onBeginResume_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onBegin"/>
  <simpleAction role="resume" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onBeginSet_var">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<causalConnector id="onBeginSet_var_delay">
  <connectorParam name="var"/>
  <connectorParam name="delay"/>
  <simpleCondition role="onBegin"/>
  <simpleAction role="set" value="$var" delay="$delay"/>
</causalConnector>

<!-- OnEnd -->

<causalConnector id="onEndStart">
  <simpleCondition role="onEnd"/>
  <simpleAction role="start" max="unbounded"/>
</causalConnector>

<causalConnector id="onEndStart_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onEnd"/>
  <simpleAction role="start" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onEndStop">
  <simpleCondition role="onEnd"/>
  <simpleAction role="stop" max="unbounded"/>
</causalConnector>

<causalConnector id="onEndStop_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onEnd"/>
  <simpleAction role="stop" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onEndPause">
  <simpleCondition role="onEnd"/>
  <simpleAction role="pause" max="unbounded"/>
</causalConnector>

<causalConnector id="onEndPause_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onEnd"/>
  <simpleAction role="pause" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onEndResume">
  <simpleCondition role="onEnd"/>
  <simpleAction role="resume" max="unbounded"/>
</causalConnector>

<causalConnector id="onEndResume_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onEnd"/>
  <simpleAction role="resume" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onEndSet_var">
  <connectorParam name="var"/>

```

```

    <simpleCondition role="onEnd"/>
    <simpleAction role="set" value="$var"/>
</causalConnector>

<causalConnector id="onEndSet_var_delay">
  <connectorParam name="var"/>
  <connectorParam name="delay"/>
  <simpleCondition role="onEnd"/>
  <simpleAction role="set" value="$var" delay="$delay"/>
</causalConnector>

<!-- OnMouseSelection -->

<causalConnector id="onSelectionStart">
  <simpleCondition role="onSelection"/>
  <simpleAction role="start" max="unbounded"/>
</causalConnector>

<causalConnector id="onSelectionStart_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onSelection"/>
  <simpleAction role="start" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onSelectionStop">
  <simpleCondition role="onSelection"/>
  <simpleAction role="stop" max="unbounded"/>
</causalConnector>

<causalConnector id="onSelectionStop_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onSelection"/>
  <simpleAction role="stop" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onSelectionPause">
  <simpleCondition role="onSelection"/>
  <simpleAction role="pause" max="unbounded"/>
</causalConnector>

<causalConnector id="onSelectionPause_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onSelection"/>
  <simpleAction role="pause" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onSelectionResume">
  <simpleCondition role="onSelection"/>
  <simpleAction role="resume" max="unbounded"/>
</causalConnector>

<causalConnector id="onSelectionResume_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onSelection"/>
  <simpleAction role="resume" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onSelectionSet_var">
  <connectorParam name="var" />
  <simpleCondition role="onSelection"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<causalConnector id="onSelectionStet_var_delay">
  <connectorParam name="var" />
  <connectorParam name="delay"/>

```

```

    <simpleCondition role="onSelection"/>
    <simpleAction role="set" value="$var" delay="$delay"/>
  </causalConnector>

  <!-- OnKeySelection -->

  <causalConnector id="onKeySelectionStart">
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="start" max="unbounded"/>
  </causalConnector>

  <causalConnector id="onKeySelectionStart_delay">
    <connectorParam name="keyCode"/>
    <connectorParam name="delay"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="start" max="unbounded" delay="$delay"/>
  </causalConnector>

  <causalConnector id="onKeySelectionStop">
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="stop" max="unbounded"/>
  </causalConnector>

  <causalConnector id="onKeySelectionStop_delay">
    <connectorParam name="keyCode"/>
    <connectorParam name="delay"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="stop" max="unbounded" delay="$delay"/>
  </causalConnector>

  <causalConnector id="onKeySelectionPause">
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="pause" max="unbounded"/>
  </causalConnector>

  <causalConnector id="onKeySelectionPause_delay">
    <connectorParam name="keyCode"/>
    <connectorParam name="delay"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="pause" max="unbounded" delay="$delay"/>
  </causalConnector>

  <causalConnector id="onKeySelectionResume">
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="resume" max="unbounded"/>
  </causalConnector>

  <causalConnector id="onKeySelectionResume_delay">
    <connectorParam name="keyCode"/>
    <connectorParam name="delay"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="resume" max="unbounded" delay="$delay"/>
  </causalConnector>

  <causalConnector id="onKeySelectionSet_var">
    <connectorParam name="keyCode"/>
    <connectorParam name="var"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="set" value="$var"/>
  </causalConnector>

  <causalConnector id="onKeySelectionSet_var_delay">
    <connectorParam name="keyCode"/>

```

```

    <connectorParam name="var"/>
    <connectorParam name="delay"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="set" value="$var" delay="$delay"/>
</causalConnector>

<!-- OnBeginAttribution -->

<causalConnector id="onBeginAttributionStart">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="start" max="unbounded"/>
</causalConnector>

<causalConnector id="onBeginAttributionStart_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="start" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onBeginAttributionStop">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="stop" max="unbounded"/>
</causalConnector>

<causalConnector id="onBeginAttributionStop_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="stop" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onBeginAttributionPause">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="pause" max="unbounded"/>
</causalConnector>

<causalConnector id="onBeginAttributionPause_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="pause" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onBeginAttributionResume">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="resume" max="unbounded"/>
</causalConnector>

<causalConnector id="onBeginAttributionResume_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="resume" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onBeginAttributionSet_var">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<causalConnector id="onBeginAttributionSet_var_delay">
  <connectorParam name="var"/>
  <connectorParam name="delay"/>
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="set" value="$var" delay="$delay"/>
</causalConnector>

<!-- OnEndAttribution -->

```

```

<causalConnector id="onEndAttributionStart">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="start" max="unbounded"/>
</causalConnector>

<causalConnector id="onEndAttributionStart_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="start" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onEndAttributionStop">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="stop" max="unbounded"/>
</causalConnector>

<causalConnector id="onEndAttributionStop_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="stop" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onEndAttributionPause">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="pause" max="unbounded"/>
</causalConnector>

<causalConnector id="onEndAttributionPause_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="pause" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onEndAttributionResume">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="resume" max="unbounded"/>
</causalConnector>

<causalConnector id="onEndAttributionResume_delay">
  <connectorParam name="delay"/>
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="resume" max="unbounded" delay="$delay"/>
</causalConnector>

<causalConnector id="onEndAttributionSet_var">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<causalConnector id="onEndAttributionSet_var_delay">
  <connectorParam name="var"/>
  <connectorParam name="delay"/>
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="set" value="$var" delay="$delay"/>
</causalConnector>

<!-- OnBegin multiple actions -->

<causalConnector id="onBeginStartStop">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" />
    <simpleAction role="stop" max="unbounded" />
  </compoundAction>
</causalConnector>

```

```

<causalConnector id="onBeginStart_delay1Stop_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="start" delay="$delay1" max="unbounded" />
    <simpleAction role="stop" delay="$delay2" max="unbounded" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartPause">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" />
    <simpleAction role="pause" max="unbounded" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStart_delay1Pause_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="start" delay="$delay1" max="unbounded" />
    <simpleAction role="pause" delay="$delay2" max="unbounded" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartResume">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" />
    <simpleAction role="resume" max="unbounded" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStart_delay1Resume_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="start" delay="$delay1" max="unbounded" />
    <simpleAction role="resume" delay="$delay2" max="unbounded" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" />
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStart_delay1Set_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="start" delay="$delay1" max="unbounded" />
    <simpleAction role="set" value="$var" delay="$delay2"/>
  </compoundAction>
</causalConnector>

```

```

<causalConnector id="onBeginStopStart">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" />
    <simpleAction role="start" max="unbounded" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStop_delay1Start_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" delay="$delay1" max="unbounded" />
    <simpleAction role="start" delay="$delay2" max="unbounded" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopPause">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" />
    <simpleAction role="pause" max="unbounded" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStop_delay1Pause_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" delay="$delay1" max="unbounded" />
    <simpleAction role="pause" delay="$delay2" max="unbounded" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopResume">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" />
    <simpleAction role="resume" max="unbounded" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStop_delay1Resume_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" delay="$delay1" max="unbounded" />
    <simpleAction role="resume" delay="$delay2" max="unbounded" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" />
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStop_delay1Set_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>

```

```

    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="stop" delay="$delay1" max="unbounded" />
        <simpleAction role="set" delay="$delay2" value="$var"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetStart">
    <connectorParam name="var"/>
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="start" max="unbounded" />
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginSet_delay1Start_delay2">
    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var" delay="$delay1" />
        <simpleAction role="start" delay="$delay2" max="unbounded"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetStop">
    <connectorParam name="var"/>
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="stop" max="unbounded"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginSet_delay1Stop_delay2">
    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var" delay="$delay1" />
        <simpleAction role="stop" delay="$delay2" max="unbounded"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetPause">
    <connectorParam name="var"/>
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="pause" max="unbounded"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginSet_delay1Pause_delay2">
    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var" delay="$delay1" />
        <simpleAction role="pause" delay="$delay2" max="unbounded"/>
    </compoundAction>
</causalConnector>

```

```

<causalConnector id="onBeginSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSet_delay1Resume_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1" />
    <simpleAction role="resume" delay="$delay2" max="unbounded"/>
  </compoundAction>
</causalConnector>

<!-- OnEnd multiple actions -->

<causalConnector id="onEndStartStop">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="stop" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStart_delay1Stop_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="stop" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStartPause">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="pause" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStart_delay1Pause_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStartResume">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

```

```

<causalConnector id="onEndStart_delay1Resume_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStart_delay1Set_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="set" value="$var" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopStart">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="start" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStop_delay1Start_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="start" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopPause">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="pause" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStop_delay1Pause_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopResume">

```

```

    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
      <simpleAction role="stop" max="unbounded"/>
      <simpleAction role="resume" max="unbounded"/>
    </compoundAction>
  </causalConnector>

  <causalConnector id="onEndStop_delay1Resume_delay2">
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
      <simpleAction role="stop" max="unbounded" delay="$delay1"/>
      <simpleAction role="resume" max="unbounded" delay="$delay2"/>
    </compoundAction>
  </causalConnector>

  <causalConnector id="onEndStopSet">
    <connectorParam name="var"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
      <simpleAction role="stop" max="unbounded"/>
      <simpleAction role="set" value="$var"/>
    </compoundAction>
  </causalConnector>

  <causalConnector id="onEndStop_delay1Set_delay2">
    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
      <simpleAction role="stop" max="unbounded" delay="$delay1"/>
      <simpleAction role="set" delay="$delay2" value="$var"/>
    </compoundAction>
  </causalConnector>

  <causalConnector id="onEndSetStart">
    <connectorParam name="var"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
      <simpleAction role="set" value="$var"/>
      <simpleAction role="start" max="unbounded"/>
    </compoundAction>
  </causalConnector>

  <causalConnector id="onEndSet_delay1Start_delay2">
    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
      <simpleAction role="set" value="$var" delay="$delay1" />
      <simpleAction role="start" max="unbounded" delay="$delay2"/>
    </compoundAction>
  </causalConnector>

  <causalConnector id="onEndSetStop">
    <connectorParam name="var"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
      <simpleAction role="set" value="$var"/>
      <simpleAction role="stop" max="unbounded"/>
    </compoundAction>
  </causalConnector>

  <causalConnector id="onEndSet_delay1Stop_delay2">

```

```

    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var" delay="$delay1" />
        <simpleAction role="stop" max="unbounded" delay="$delay2"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onEndSetPause">
    <connectorParam name="var"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="pause" max="unbounded"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onEndSet_delay1Pause_delay2">
    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var" delay="$delay1" />
        <simpleAction role="pause" max="unbounded" delay="$delay2"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onEndSetResume">
    <connectorParam name="var"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="resume" max="unbounded"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onEndSet_delay1Resume_delay2">
    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var" delay="$delay1" />
        <simpleAction role="resume" max="unbounded" delay="$delay2"/>
    </compoundAction>
</causalConnector>

<!-- OnMouseSelection multiple actions -->

<causalConnector id="onSelectionStartStop">
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
        <simpleAction role="start" max="unbounded"/>
        <simpleAction role="stop" max="unbounded"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStart_delay1Stop_delay2">
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
        <simpleAction role="start" max="unbounded" delay="$delay1"/>

```

```

    <simpleAction role="stop" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartPause">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="pause" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStart_delay1Pause_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartResume">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStart_delay1Resume_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStart_delay1Set_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="set" value="$var" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopStart">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="start" max="unbounded"/>
  </compoundAction>
</causalConnector>

```

```

<causalConnector id="onSelectionStop_delay1Start_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="start" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopPause">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="pause" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStop_delay1Pause_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopResume">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStop_delay1Resume_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStop_delay1Set_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="set" delay="$delay2" value="$var"/>
  </compoundAction>
</causalConnector>

```

```

<causalConnector id="onSelectionSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSet_delay1Start_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1" />
    <simpleAction role="start" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="stop" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSet_delay1Stop_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1" />
    <simpleAction role="stop" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSet_delay1Pause_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1" />
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>

```

```

</causalConnector>

<causalConnector id="onSelectionSet_delay1Resume_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1" />
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<!-- OnKeySelection multiple actions -->

<causalConnector id="onKeySelectionStartStop">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="stop" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStart_delay1Stop_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="stop" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStartPause">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="pause" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStart_delay1Pause_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStartResume">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStart_delay1Resume_delay2">
  <connectorParam name="delay1"/>

```

```

    <connectorParam name="delay2"/>
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <compoundAction operator="seq">
        <simpleAction role="start" max="unbounded" delay="$delay1"/>
        <simpleAction role="resume" max="unbounded" delay="$delay2"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStartSet">
    <connectorParam name="var"/>
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <compoundAction operator="seq">
        <simpleAction role="start" max="unbounded"/>
        <simpleAction role="set" value="$var"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStart_delay1Set_delay2">
    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <compoundAction operator="seq">
        <simpleAction role="start" max="unbounded" delay="$delay1"/>
        <simpleAction role="set" value="$var" delay="$delay2"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopStart">
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <compoundAction operator="seq">
        <simpleAction role="stop" max="unbounded"/>
        <simpleAction role="start" max="unbounded"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStop_delay1Start_delay2">
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <compoundAction operator="seq">
        <simpleAction role="stop" max="unbounded" delay="$delay1"/>
        <simpleAction role="start" max="unbounded" delay="$delay2"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopPause">
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <compoundAction operator="seq">
        <simpleAction role="stop" max="unbounded"/>
        <simpleAction role="pause" max="unbounded"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStop_delay1Pause_delay2">
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <compoundAction operator="seq">
        <simpleAction role="stop" max="unbounded" delay="$delay1"/>

```

```

    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopResume">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStop_delay1Resume_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopSet">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStop_delay1Set_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="set" delay="$delay2" value="$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" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSet_delay1Start_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1" />
    <simpleAction role="start" max="unbounded" delay="$delay2"/>
  </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" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSet_delay1Stop_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1" />
    <simpleAction role="stop" max="unbounded" delay="$delay2"/>
  </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" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSet_delay1Pause_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1" />
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </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" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSet_delay1Resume_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1" />
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

```

```

<!-- OnBeginAttribution multiple actions -->

<causalConnector id="onBeginAttributionStartStop">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="stop" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStart_delay1Stop_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="stop" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStartPause">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="pause" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStart_delay1Pause_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStartResume">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStart_delay1Resume_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

```

```

<causalConnector id="onBeginAttributionStart_delay1Set_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="set" value="$var" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopStart">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="start" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStop_delay1Start_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="start" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopPause">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="pause" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStop_delay1Pause_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopResume">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStop_delay1Resume_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopSet">
  <connectorParam name="var"/>

```

```

    <simpleCondition role="onBeginAttribution"/>
    <compoundAction operator="seq">
        <simpleAction role="stop" max="unbounded"/>
        <simpleAction role="set" value="$var"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStop_delay1Set_delay2">
    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onBeginAttribution"/>
    <compoundAction operator="seq">
        <simpleAction role="stop" max="unbounded" delay="$delay1"/>
        <simpleAction role="set" delay="$delay2" value="$var"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSetStart">
    <connectorParam name="var"/>
    <simpleCondition role="onBeginAttribution"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="start" max="unbounded"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSet_delay1Start_delay2">
    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onBeginAttribution"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var" delay="$delay1"/>
        <simpleAction role="start" max="unbounded" delay="$delay2"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSetStop">
    <connectorParam name="var"/>
    <simpleCondition role="onBeginAttribution"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="stop" max="unbounded"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSet_delay1Stop_delay2">
    <connectorParam name="var"/>
    <connectorParam name="delay1"/>
    <connectorParam name="delay2"/>
    <simpleCondition role="onBeginAttribution"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var" delay="$delay1"/>
        <simpleAction role="stop" max="unbounded" delay="$delay2"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSetPause">
    <connectorParam name="var"/>
    <simpleCondition role="onBeginAttribution"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="pause" max="unbounded"/>
    </compoundAction>
</causalConnector>

```

```

<causalConnector id="onBeginAttributionSet_delay1Pause_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1"/>
    <simpleAction role="pause" max="unbounded" delay="$delay2" />
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSet_delay1Resume_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1"/>
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<!-- OnEndAttribution multiple actions -->

<causalConnector id="onEndAttributionStartStop">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="stop" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStart_delay1Stop_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="stop" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStartPause">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="pause" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStart_delay1Pause_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

```

```

    </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStartResume">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStart_delay1Resume_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStart_delay1Set_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start" max="unbounded" delay="$delay1"/>
    <simpleAction role="set" value="$var" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStopStart">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="start" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStop_delay1Start_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="start" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStopPause">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="pause" max="unbounded"/>
  </compoundAction>
</causalConnector>

```

```

<causalConnector id="onEndAttributionStop_delay1Pause_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStopResume">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStop_delay1Resume_delay2">
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStop_delay1Set_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded" delay="$delay1"/>
    <simpleAction role="set" delay="$delay2" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSet_delay1Start_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1"/>
    <simpleAction role="start" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

```

```

<causalConnector id="onEndAttributionSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="stop" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSet_delay1Stop_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1"/>
    <simpleAction role="stop" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSet_delay1Pause_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1"/>
    <simpleAction role="pause" max="unbounded" delay="$delay2"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume" max="unbounded"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSet_delay1Resume_delay2">
  <connectorParam name="var"/>
  <connectorParam name="delay1"/>
  <connectorParam name="delay2"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var" delay="$delay1"/>
    <simpleAction role="resume" max="unbounded" delay="$delay2"/>
  </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="onKeySelectionStopSet_varPauseStart">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelection_orStopSet_varPauseStart">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode" max="unbounded"/>
  <compoundAction operator="seq">
    <simpleAction role="stop" max="unbounded"/>
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause" max="unbounded"/>
    <simpleAction role="start" max="unbounded"/>
  </compoundAction>
</causalConnector>

</connectorBase>
</connectorBase>
</head>

</ncl>

```

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 shall control the content presentation 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 should be disconnected from media player presentations.

Figure B.1 suggests a modular implementation, where media players are plug-in modules of the presentation engine. Since it can be interesting to use already existing 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.

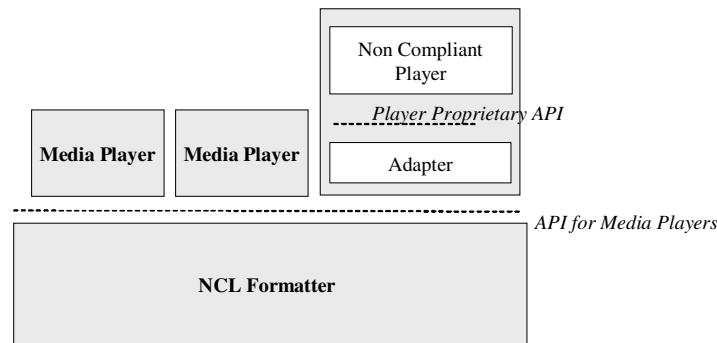


Figure B.1 – APIs for Integrating Media Players with an NCL formatter

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 Basic Media Players

This section deals with media players for <media> elements whose types are different from those of media object containing hypermedia declarative code (for example, “application/x-ncl-NCL” type and text/html type) and different from any media object containing imperative or functional code (for example, “application/x-ncl-NCLua” type).

A media object being presented is identified by the *id* attribute of the corresponding <media> element and the *ids* of the <descriptor> elements that were associated with the media object. This identification is called in this section *representationObjectId*.

For each media object, the associated media player (or its adapter) shall be able to receive presentation commands, to control all event state machines, and answer queries from the formatter.

B.1.1. Start instruction for presentation events

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 associated with the media object to be exhibit. If this attribute is not specified, the formatter shall take into account the *type* attribute of the <media> element. If this attribute is not specified either, the formatter shall consider the file extension specified in the *src* attribute of the <media> element.

The *start* instruction issued by a formatter shall inform the following parameters to the media player: the locator of the content of the media object to be controlled, a list of all properties associated with the media object, the media object identification during execution (*representationObjectId*), a list of events (presentation, selection or attribution) that need to be monitored by the media player (defined by the <media> element's <area> and <property> child elements, and by the default content anchor), the presentation event that needs to be started (called here main-event), an optional offset-time and an optional delay-time.

The *src* attribute of the <media> element shall 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 shall finish the starting operation without performing any action.

The descriptors shall be chosen by the formatter following the directives specified in the NCL document. If the *start* instruction results from a link action that has a descriptor explicitly declared in its <bind> element (*descriptor* attribute of the child <bind> element of the <link> element), the resulting descriptor shall merge the attributes of the bind descriptor with the attributes of the descriptor specified in the corresponding <media> element, if this attribute was specified. For the common attributes, the <bind> descriptor information shall superpose the <media> descriptor data. If the <bind> element does not contain an explicit descriptor, the evaluated descriptor shall be the <media> descriptor, if this attribute was specified. Otherwise, a default descriptor for that *type* of <media> shall be chosen by the formatter. Based on this procedure, the list of <descriptor> elements' *id* that are associated with the media object is computed, and, unifying the properties defined by the resulting descriptor with those properties explicitly defined by the <media> element, the list of properties associated with the media object is evaluated.

The list of events to be monitored by a media player should also be computed by the formatter, taking into account the NCL document specification. It shall check all links in which the media object and the resulting descriptor participate (that is, in which the *representationObjectId* object participates). When computing the events to be monitored, the formatter shall take into account the media-object perspective, i.e., the depth 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* parameter²³ 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 shall be immediately started (i.e., it commands the player to jump to the beginning-time + offset-time). Obviously, the offset-time value shall be lower than the main-event duration; otherwise, that *start* instruction shall be ignored. If the offset-time is greater than zero, the media player shall put the main-event in the *occurring* state, but the event *starts* transition shall not be notified. If the offset-time is zero, the media player shall put the main-event in the *occurring* state and notify the *starts* transition occurrence.

Events that would have their end-times previous to the beginning-time of the main-event and events that would have their beginning times after the end-time of the main-event do not need to be monitored by the media player (the formatter should do this verification when building the monitored event list).

Monitored events that would have beginning-times before the start time (beginning-time + offset-time) of the main-event and end-times after the start time (beginning-time + offset-time) of the main-event shall be put in the *occurring* state, but their *starts* transitions shall not be notified (links that depend on this transition shall not be fired).

Monitored events that would have their end times after the main-event beginning-time, but before the start time (beginning-time + offset-time) shall have their *occurrences* attribute incremented but the *starts* and *stops* transitions shall not be notified.

The *delay-time* is also an optional parameter and its default value is “0s” too. If greater than zero, this parameter contains a time to be waited by the media player before starting the presentation.

If a media player receives a *start* instruction for an object already being presented (paused or not), it shall ignore the instruction and keep on controlling the ongoing presentation. . In this case, the <simpleAction> element that has caused the *start* instruction shall not cause any transition on the corresponding event state machine.

B.1.2. Stop instruction for presentation events

The *stop* instruction only needs to identify a media object already being controlled (*representationObjectId*). To identify the media object means to identify the <media> element, and the corresponding descriptors. Therefore, if a <simpleAction> element with an *actionType* attribute equal to “stop” is bound through a link to an object’s interface, the interface shall be ignored when the action is performed.

If the object is not being presented (none of the events in the object’s list of events is in the *occurring* or *paused* state) and the media player is not waiting due to a delayed *start*

²³ The *offset-time* and the *delay-time* parameters are computed by the NCL formatter, before triggering the start instruction, based on the application life cycle control, as for example, based on an NCLEditingCommand received.

instruction, the *stop* instruction shall 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 shall transit to the *sleeping* state, and their *stops* transitions shall be notified. Monitored events in the *occurring* or in the *paused* state with end time posterior to the main-event end time shall be put in the *sleeping* state, but their *stops* transitions shall not be notified and their *occurrences* attribute shall not be incremented. The object content presentation shall be stopped. If the *repetitions* event attribute is greater than zero, it shall be decremented by one and the main-event presentation shall restart after the repeat delay time (the repeat delay shall have been passed to the media player as the start delay parameter). If the media object is waiting to be presented after a delayed *start* instruction and a *stop* instruction is issued, the previous *start* instruction shall be removed.

The *stop* instruction shall transit the monitored events to the *sleeping* state no matter if a transition effect is being applied to the media object. In other words, the transition effect shall also be stopped. Transition effects are never applied after an object suffers a *stop* instruction.

B.1.3. Abort instruction for presentation events

The *abort* instruction only needs to identify a media object already being controlled (*representationObjectId*). If a <simpleAction> element with an *actionType* attribute equal to “abort” is bound through a link to an object’s interface, the interface shall be ignored when the action is applied.

If the object is not being presented and is not waiting to be presented after a delayed *start* instruction, the *abort* instruction shall be ignored. If the object is being presented, the main-event and all monitored events in the *occurring* or in the *paused* state shall transit to the *sleeping* state, and their *aborts* transitions shall be notified. Any content presentation shall stop. If the *repetitions* event attribute is greater than zero, it shall be set to zero and the media object presentation shall 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 shall be removed.

B.1.4. Pause instruction for presentation events

The *pause* instruction only needs to identify a media object already being controlled (*representationObjectId*). If a <simpleAction> element with an *actionType* attribute equal to “pause” is bound through a link to an object’s interface, the interface shall be ignored when the action is applied.

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 shall be ignored. If the object is being presented, the main-event and all monitored events in the *occurring* state shall transit to the *paused* state and their *pauses* transitions shall be notified. The object presentation shall be paused and the pause elapsed time shall 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 shall 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 shall wait for a resume instruction to continue waiting for the remaining start delay.

B.1.5. Resume instruction for presentation events

The *resume* instruction only needs to identify a media object already being controlled (*representationObjectId*). If a <simpleAction> element with an *actionType* attribute equal to “resume” is bound through a link to an object’s interface, the interface shall be ignored when the action is applied.

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 shall be ignored. If the media player is paused waiting for the start delay, it shall 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 shall be put in the *occurring* state and their *resumes* transitions shall be notified.

B.1.6. Start instruction for attribution events

The *start* instruction may be applied to an object property independent from the fact if the object is being presented or not (in this last case, although the object is not being presented, its media player shall have already been instantiated). The *start* instruction needs to identify the media object being controlled (*representationObjectId*), and a monitored attribution event, and also needs to determine a value to be assigned to the property wrapped by the event, the duration of the attribution process, and the attribution step. When setting a value to the property, the media player shall set the event state machine to the *occurring* state and after finishing the attribution, again to the *sleeping* state, generating the *starts* transition and afterwards the *stops* transition.

For every monitored attribution event, if the media player changes by itself the corresponding attribute value, it shall also proceed as if it had received an external start instruction.

B.1.7. stop, abort, pause and resume instructions for attribution events

The *stop*, *abort*, *pause* and *resume* instructions need to identify a media object already being controlled (*representationObjectId*) and an attribution event being monitored.

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

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

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

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

B.1.8. AddEvent instruction

The *addEvent* instruction is issued in the case of receiving an *addInterface* NCL editing command (see Technical Report Part 9: NCL Live Editing). The instruction needs to identify a media object already being controlled and a new event that shall be included to be monitored. All rules applied to the intersection of monitored events with the main-event shall 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 shall be put in the same state of the main-event (*occurring* or *paused*), without notifying the corresponding transition.

B.1.9. RemoveEvent instruction

The *removeEvent* instruction is issued in the case of receiving a *removeInterface* NCL editing command (see the Technical Report: NCL 3.0: 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 shall be put in the *sleeping* state without generating any transition.

B.1.10. Natural end of a presentation

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 shall transit the event to the *sleeping* state and notify the *stops* transition. The same shall be done for monitored events in the *occurring* state with the same end time of the main-event or with unknown end time, when the main-event ends. Events in the *occurring* state with end time posterior to the main-event end time shall be put in the sleeping state but without generating the *stops* transition and without incrementing the *occurrences* attribute. It is important to remark that when the main-event finishes, the whole media object presentation shall finish.

B.2. Expected Behavior of Media Players after Instructions Applied to Composite Objects

This section applies only for objects represented by <context>, <body> and <switch> elements.

A <simpleCondition> or <simpleAction> with *eventType* attribute value equal to “presentation” may be bound by a link to a composite node (represented by a <context>, <switch>, 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 shall 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” may be bound by a link to a composite node (represented by

a <context>, <switch>, 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.

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 shall also be reflected to the event state machines of the composite child nodes, as explained in the following subsections.

B.2.1. Starting a context presentation

If a <context> or <body> element participates on an action role whose action type is “start”, when this action is fired without referring to any specific interface, the *start* instruction shall 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 shall in addition indicate the <port> id as the *interface* value of the corresponding <bind> element.

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 without referring to any specific interface, the *stop* instruction shall also be applied to all presentation events of the composite child nodes.

If the composite node contains links being evaluated (or with their evaluation paused), the evaluations shall be suspended and no action shall 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 without referring to any specific interface, the *abort* instruction shall also be applied to all presentation events of the composite child nodes.

If the composite contains links being evaluated (or with their evaluation paused), the evaluations shall be suspended and no action shall 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 without referring to any specific interface, the *pause* instruction shall also be applied to all presentation events of the composite child nodes that are in the occurring state.

If the composite contains links being evaluated, all evaluations shall 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 shall be identified because if the composite receives a resume instruction, these events shall 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 without referring to any specific interface, the *resume* instruction shall 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.

If the composite contains links with paused evaluations, they shall be resumed.

B.3. Relation between the presentation-event state machine of a node and the presentation-event state machine of its parent-composite node²⁴

This section applies for objects represented by <context>, <body>, and <switch> elements, and <media> elements of “application/x-ncl-NCL” type.

Whenever a presentation event of a child node (media or composite) goes to the occurring state, the presentation event of the composite node (or of the NCL node of “application/x-ncl-NCL” type) that contains the node shall also enter in the occurring state.

When all child nodes of a composite node (or of an NCL node of “application/x-ncl-NCL” type) have their presentation events in the sleeping state, the presentation event of the composite node (or of the NCL node of “application/x-ncl-NCL” type) shall also be in the sleeping state.

Composite nodes (or NCL nodes of “application/x-ncl-NCL” type) do not need to infer *aborts* transitions from their child nodes. These transitions in presentation events of composite nodes (or of NCL nodes of “application/x-ncl-NCL” type) shall occur only when instructions are applied directly to composite node presentation events (see 8.4).

When all child nodes of a composite node (or of an NCL node of “application/x-ncl-NCL” type) have their presentation events in a state different from the *occurring* state and at least one child node has its main-event in the *paused* state, the presentation event of the composite node (or of the NCL node of “application/x-ncl-NCL” type) shall also be in the paused state.

If a <switch> element is started, but it does not define a default component and none of the <bindRule> referred rules is evaluated as true, the switch presentation shall not enter in the occurring state.

²⁴ The concepts provided in this section also applies to a <media> element of “application/x-ginga-NCL” type that will behave as if it is a composite node made up by its <body> element and shall be treated accordingly.

B.4. Expected behavior of imperative media players in NCL applications

Imperative objects may be inserted into NCL documents mainly to bring additional computational capabilities to declarative documents. The way to add imperative objects into NCL documents is to define a `<media>` element, whose content (located through the *src* attribute) is the imperative code to be executed. Both EDTV and BDTV profiles of NCL 3.0 allows the `<media>` element of application/x-ncl-NCLua type, for Lua codes (file extension `.lua`).

Authors may define NCL links to start, stop, pause, resume or abort the execution of an imperative code. An imperative player (the language engine) shall interface the imperative execution environment with the NCL formatter.

NCL allows imperative code execution to be synchronized with other NCL objects (imperative or not). A `<media>` element containing an imperative code may also define anchors (through `<area>` elements) and properties (through `<property>` elements).

Analogous to conventional media content players, imperative players shall control event state machines associated with the imperative media-object. As an example, if the code finishes its execution, the player shall generate the *stops* transition in the event presentation state machine corresponding to the imperative code execution. The difference in this case is that the notifications are not automatic. Author of Lua code must place commands in the code to make these notifications.

Imperative code span may be associated with an `<area>` element (using the *label* attribute). If external links start, stop, pause, resume or abort the anchor presentation, callbacks in the imperative code span shall be triggered. The way these callbacks are defined is responsibility of each imperative code associated with the NCL imperative object.

On the other hand, the imperative code span may also notify the start, stop, pause or resume of its `<area>` elements through an API offered by the imperative language. These transitions may 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 imperative code and the remainder of the NCL document.

The other way an imperative code may be synchronized with other NCL objects is through `<property>` elements. A `<property>` element defined as a child of a `<media>` element representing an imperative code may be mapped to a code span (function, method, etc.) or to an object's property (an usual media object property or a code attribute). When it is mapped to a code span, a link action "set" applied to the property shall cause the code span execution, with the set values interpreted as input parameters. The *name* attribute of the `<property>` element shall be used to identify the imperative code span. When the `<property>` element is mapped to a property, the action "set" shall only assign the value to the property. As usual, the event state machine associated with the property shall be controlled by the imperative player. It should be stressed again the difference in this case is that the notifications about event state transitions are not automatic. Author of Lua code must place commands in the code to make these notifications.

A `<property>` element defined as a child of a `<media>` element representing an imperative code may also be associated with an NCL link assessment role. In this case, the NCL formatter shall query the property value in order to evaluate the link expression. If the

<property> element is mapped to a property, the property value shall be returned by the imperative player to the NCL formatter. If the <property> element is mapped to a code span, it shall be called and the output value resulting from its execution shall be returned by the imperative player to the NCL formatter.

Imperative players for NCL media objects with imperative code content shall follow the guidelines established in Technical Report “Nested Context Language 3.0: Part 10” [SoSC08].

There are two additional main differences regarding media players for imperative code content and the basic media players discussed in Section B.1.

First, if an imperative player receives a *start* instruction for an event associated with an <area> element and this event is in the *sleeping* state, it shall start the execution of the imperative code span associated with the element, even though other portion of the object’s imperative code is being in execution (paused or not). However, if the event associated with the target <area> element is in the *occurring* or *paused* state, the *start* instruction shall be ignored by the imperative code player that keeps on controlling the ongoing execution. As a consequence, different from what happens for other <media> elements, a <simpleAction> element with an *actionType* attribute equal to “stop”, “pause”, “resume” or “abort” shall be bound through a link to an imperative media-object’s interface, which shall not be ignored when the action is applied.

Second, an imperative object shall have a content anchor called the *whole content anchor* and it is declared by default in NCL documents. This content anchor, however, has a special meaning. It represents the execution of any code span inside the imperative-code object. Another content anchor is also defined by default, called *main* content anchor. Every time an imperative object is started without specifying one of its content anchors or properties, the *main* content anchor is assumed and, as a consequence, the code span associated to it. In all other references to the imperative object without specifying one of its content anchors or properties, the *whole content anchor* shall be assumed.

B.5. Expected behavior of hypermedia players in NCL applications

NCL media objects (<media> elements of “application/x-ginga-NCL” or “application/x-ncl-NCL” type) and HTML-based objects (<media> elements of “text/html” type) embedded in NCL applications shall follow the guidelines established in “Nested Context Language 3.0: Part 11” [Soar09], as well as media objects with other declarative code content.