PUC

Nested Context Model 3.0
Part 1 – NCM Core

Luiz Fernando Gomes Soares
Rogério Ferreira Rodrigues

Departamento de Informática

Nested Context Model 3.0[1]

Part 1 – NCM Core

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 basic entities of the Nested Context Model (NCM) version 3.0. NCM is a conceptual model focused on the representation and handling of hypermedia documents.*

---

[1] This report has a version in Portuguese, under the same title.

Nested Context Model 3.0

Part 1 – NCM Core

Table of Contents

Nested Context Model 3.0: Part 1 – NCM Core

1. Introduction

This report describes the basic entities of NCM (*Nested Context Model*) version 3.0. NCM is a conceptual model focused on the representation and handling of hypermedia documents.

A hypermedia conceptual model should represent the data structural-concepts, as well as events and relationships concerning these data. The model should also define the structuring rules and the operations on data for manipulating and updating the structures.

The first NCM definitions [SoCR91] relied mostly on aspects related to the data structures of the basic model and the structuring rules. The structures and operations for version control were subjects of the subsequent specification [SoCR95, SSRM99]. Afterwards, the previous definitions were revised and updated, adding to the basic model operations for creating, editing and presenting the document structure [Soar00]. The NCM 2.2 specification also added new data structures for defining spatial and temporal document synchronization relationships, as well as operations for defining and presenting these synchronization relationships.

Version 2.2 reviewed the original class hierarchy entirely, incorporating some new classes [SoCR95]. Moreover, some original classes were redefined, especially the *link* class, through the definition of new attributes and the redefinition of existing ones. Actually, all model classes that could be instantiated were partially or significantly modified.

This new version of NCM, called NCM 3.0, makes a new review of the NCM class hierarchy. Again, the link class is one of the main model entities affected by the revision. Indeed, it is completely redefined. New support for document and presentation adaptations is also introduced.

In NCM 3.0, a novel specification approach is also adopted. In order to offer a scalable hypermedia model, with characteristics that shall be progressively incorporated in hypermedia system implementations, NCM was divided in several parts:

- Part 1 – NCM Core
  concerned with the main model entities, which should be present in all NCM implementations[2].

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

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

---

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

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

NCM is the model underlying NCL (*Nested Context Language*), an XML application language for authoring hypermedia documents. The NCL specification is composed of Parts 5 to 12 of the collection:

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

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

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

- Part 8 – NCL (Nested Context Language) Digital TV Profiles
  concerned with the definition of an XML application language for authoring documents aiming at the digital TV domain. Two profiles are defined: the Enhanced Digital TV (EDTV) profile and the Basic Digital TV (BDTV) profile.

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

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

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

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

This technical report deals with basic model entities that comprise the NCM core, as discussed throughout all subsections of Section 2.

2. NCM Core Entities

NCM is based on the usual concepts of nodes and links. *Nodes* represent information fragments and *links* are used to define relationships among nodes. However, links are not the only entity available for defining relationships, as will be clarified in this section.

The model distinguishes two basic classes of nodes, called content nodes and composite nodes, the latter being the NCM central point to define other types of relationships. Figure 1 offers a basic overview of the model class hierarchy[3], which will be detailed along the rest of this section following a top-down approach.
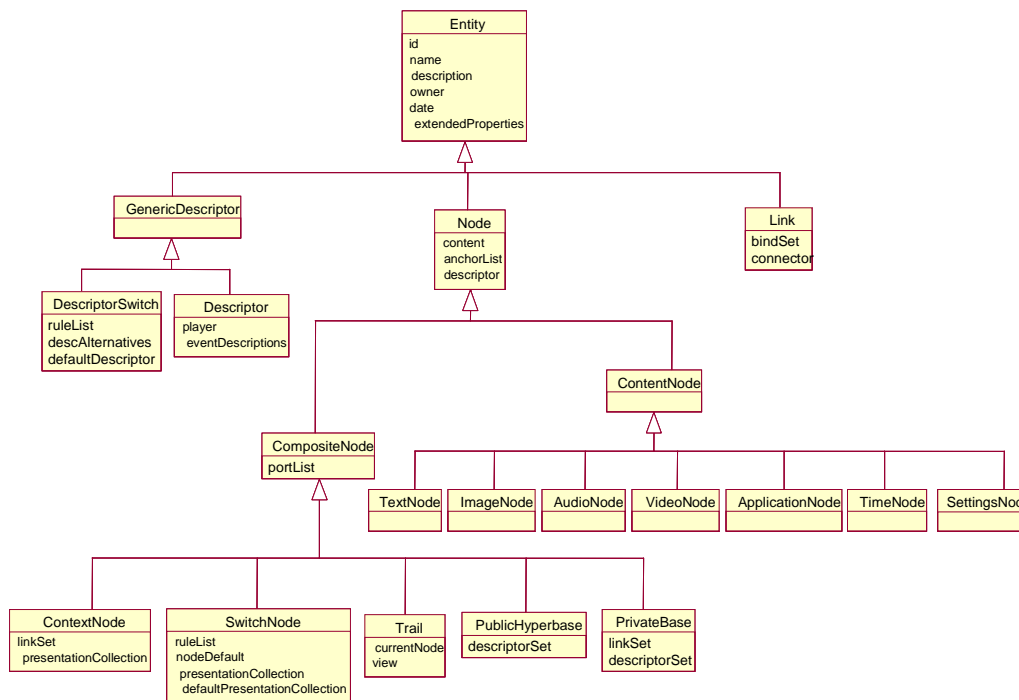
Figure 1 – Overview of NCM class hierarchy[4].

2.1. Entities and Properties

Every NCM *entity* has as attributes: a *unique identifier* (ID), a *name*, a *description*, a *date of creation* and an *owner*[5]. Besides this basic collection of attributes, an NCM entity maintains a list of extended attributes to allow extensions not only by means of class inheritance. In NCM, most of the attributes are called *properties* and should be wrapped by an NCM *property* class. The reason for this is that NCM foresees the support for maintaining, for each entity property (*basic* or *extended*), information about access rights, the user that last modified its value, the date of this modification, whether

---

[3] It must be emphasized that, although it follows an object-oriented specification, NCM does not oblige an object-oriented implementation. NCM is just a hypermedia model.

[4] In order not to pollute the class diagram figures, class operations were hidden.

[5] NCM just defines a *user* type, whose implementation is left for hypermedia systems using the model.

value changes imply on entity versioning, etc. In other words, NCM predicts a very fine granular control when implementing security and version support, obliging attributes to maintain other attributes. Nevertheless, systems that are not interested in exploring all NCM capabilities may choose to model class fields as traditional attributes, instead of using the property wrapper. Even for those systems implementing access or version control, class fields that do not need to be monitored with such granularity may also be represented without using wrappers.

Entities should offer specific *getter* and *setter* methods for each basic property (e.g. getId, setId, getName, setName, etc.)[6], methods for adding/removing extended properties and two generic methods to get or set extended property values.

2.2. Nodes and Anchors

A *node* is an NCM entity that has the following additional properties: content; generic descriptor (optional property); and ordered list of anchors.

The node *content* is composed by a collection of information units. The exact notion of what constitutes an information unit is part of the node definition and depends on the node specialization, as will be exemplified further on.

NCM descriptors will be explained in Sections 2.9 and 2.10[7]. The definition of a descriptor as a node property is optional. When specified, it will contain information determining how the node should be temporally and spatially exhibited.

Each element of the *ordered list of anchors* is called a node anchor, or simply an anchor. An *anchor* is an NCM entity that can be further specialized, as illustrated in Figure 2[8]. The model defines two types of anchors (or interfaces). The first one is the *content anchor* (or *area anchor*), which has an attribute called *region*. The anchor *region* specifies a collection of information units of the node content. Any subset of content information units may define an anchor and the exact notion of what is an anchor region depends on the node content definition. However, every node shall have an anchor with a region representing the whole content of the node. This anchor is called the *all content anchor* and its corresponding region is represented by the symbol λ. The all content anchor shall always be the first anchor in the node anchor list. Anchors will be very important when specifying relationships among nodes.

---

[6] From here on, it is assumed that subclasses have to specify getter and setter methods for manipulating each entity property.

[7] Unfortunately some NCM class definitions are interdependent, what makes it difficult to write a linear text describing NCM. This model definition is a typical example where the hypertext paradigm would be very useful.

[8] Actually, NCM anchors inherit from an interface point class, which inherits from entity, as it will be clarified in Section 2.4.

Entity

Interface

Anchor

Port
node
nodeIntPt

SwitchPort
nodeList
intPointList

ContentAnchor
region

AttributeAnchor
attributeName

AllContentAnchor
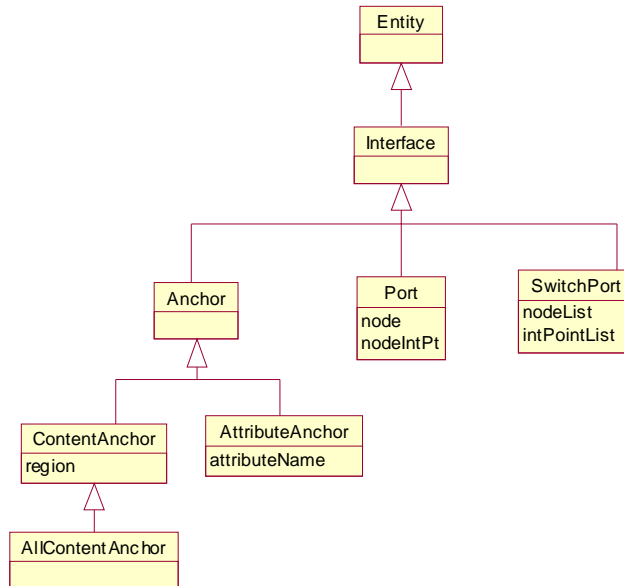
Figure 2 – NCM interface class hierarchy.

The second type of anchor defined in NCM is the *attribute anchor*. An attribute anchor may point to a node attribute (or property), or to an attribute defined in the descriptor associated with the node. During document presentations, NCM nodes are associated with NCM descriptors, as it will be explained in Section 2.9. In fact, the attribute anchor may identify any attribute recursively contained in the node, through qualified references. For example, one can use the qualified name *selectedDescriptor.xPosition* to create an attribute anchor pointing to an attribute *xPosition* defined inside the descriptor selected to present the node.

Besides the already mentioned getter and setter methods, nodes shall offer methods for manipulating their anchor lists (e.g. add, remove, get, iterate, etc.).

2.3. Content Nodes

Content nodes (also called *media nodes* or *media objects*) are NCM nodes representing the usual media objects. They should be specialized in subclasses to better define the content interpretation (e.g. text, audio, video, image, application, etc.). As aforementioned, the exact notion of what constitutes an information unit is part of the node class definition. For example, the information unit of a video content node could be a frame, while the information unit of a text content node could be a character or a word. The content of a media node may be defined via a reference (e.g. a URI) to the content itself, or by a content byte array (raw data). Moreover, each content node subclasses can be refined; for example, text nodes can be an HTML node, a PDF node, etc.

A special type of content node is the *time node*. This node represents an absolute time (as, for example, the Greenwich hour), or a relative time (based on some event, as for example, the start of a document presentation). Each instant of time is considered an information unit for the content of this kind of node. Thus, time intervals may be defined as time-node content anchors. Time node will allow triggering events based on a specific time (see Section 2.7 on events).

Another special type of content node is the *settings* node. This node represents all variables that are controlled by the formatter (the tool responsible for the NCM document presentation). These variables are represented by properties (attribute anchors) of the settings node. As will be discussed in Section 2.8, these properties may have their values changed by link actions. As will be discussed in Section 2.6 and 2.10, these properties may have their values tested by switch and descriptor switch rules to choose exhibition alternatives.

2.4. Composite Nodes

A *composite node* (or simply *composition*) C is an NCM node whose content is a collection C_L of nodes (content nodes or compositions), which constitute the composite node information units. When a node N belongs to C_L, we say that N is defined as *a component of C,* or simply *N is contained in C* (and *C contains N*). We also say that a node A is *recursively contained in C* (and *C recursively contains A*) if and only if *A* is contained in *C* or *A* is contained in a composite node recursively contained in *C*. Note that *C* components can be ordered (ordered list), which will be useful in the definition of navigation operations. Note also that a node may be contained more than once in C_L. However, there is an important restriction: a node cannot recursively contain itself.

Different compositions may contain the same node and composite nodes may be nested to any depth, if the restriction of a composite node not recursively containing itself is respected. In order to identify through which sequence of nested composite nodes the instance of a node N is being observed, NCM introduces the notion of node perspective. The *perspective* of a node N is a sequence $P = (N_m,....,N_1)$, with $m \geq 1$, such that $N_1 = N$, N_{i+1} is a composite node, N_i is contained in N_{i+1}, for $i \in [1,m)$, and N_m is not contained in any other node. Note that it is possible to define several perspectives for the same node *N*, if this node is contained in more than one composition. The *current perspective* of a node is the one traversed by the most recent navigation to the node (navigation possibilities will be defined further on). Given the perspective $P = (N_m,....,N_1)$, N_1 is called the *base node* of the perspective (or the *anchor node*).

Composite nodes are objects whose semantics is well known by the model. A conceptual model should represent not only the data structural concepts, but also define operations on that data for maintaining the structures. Thus, every composite node *C* should have the following methods:

- addNode: abstract (implementation dependent on the composite node subclass);

- removeNode: removes a specific node from the collection of nodes of the composition.

Based on the definitions of composite-node content and content-anchor regions (Section 2.2), we conclude that the content-anchor region of a composite node anchor should specify a subset of the composite node components. A special subset is that with all nodes (all content anchor of the composition).

Besides the ordered anchor list, composite nodes have another property named *ordered port list*. Ports and anchors have similar purpose and extend a common type named *interface*. A *port* of a composition *C* is an NCM entity that has two attributes: a node *N* and a node interface *ip*; where *N* shall be contained in *C*, and *ip* shall be an

interface defined in N, that is, contained in its anchor or port list[9]. As it can be observed, a composite node port allows defining mappings between the composition and its internal nodes. As a consequence, the composite node may make an interface of a constituent node visible for external references (for hypermedia links, for example). The set of interfaces (anchors or ports) works as an external protection for references to a node, in the sense that, to access a node content or attribute, it shall be available at the node interface. The goal is to use interfaces to prevent internal modifications (mainly in media content) from affecting other entities making reference to the node. Take as an example a text content node with a content-anchor region marking the second paragraph of the text content. Any change inside the text, for example the deletion of the first paragraph, should be hidden from external references, thus maintaining the references to the correct content part (i.e. the old second paragraph now positioned as the first one). The node shielding through interfaces will also bring to the model the concept of compositionality, allowing formal proof of document properties like the temporal consistency.

We define a *port mapping sequence* of a port p_k in a composition N_k the sequence of nodes and interfaces $(N_k, ip_k,....,N_1, ip_1)$ with $k > 1$, such that, for $i \in [1,k)$:

i) N_{i+1} is a composite node, and N_i is contained in N_{i+1};

ii) ip_i is an interface of N_i, and N_i and ip_i are the values of the node and interface attributes, respectively, of a port ip_{i+1} of node N_{i+1}. We say that *ip_i is in* the port mapping sequence of *p*.

Note that defining two types of composite node interfaces (anchors and ports) allows two kinds of presentation actions, desirable when building a hypermedia document. One can want to play a composition to view the composition structure (for instance, a drawing showing the composition structural graph). Composition anchors are interfaces to express this kind of behavior, and the anchor region will enumerate the components that should be drawn. However, to play a composition, sometimes, means to play its constituents from an entry point, without showing the structural view of the composition. Ports serve exactly to give these access points, allowing external references to touch nodes contained inside a composite node, without losing the model compositionality property (i.e. composite node entry and exit points shall be explicitly defined).

An action on a composite node must specify the interface where it applies. If this interface is not specified, the action must be considered to be applied on every composite node's ports.

Composite node subclasses will define semantics for specific collections of nodes. Five important composite node subclasses defined by NCM are the context node, switch node, trail, public hyperbase and private base.

---

[9] Obviously, *ip* is defined in the port list of *N* only if *N* is a composition.

2.5. Context Nodes

A *context node* is an NCM composite node that contains a set of content, context or switch nodes[10]. Context nodes have, as additional basic properties, a set of links and a presentation collection.

Each *link l* contained in the *set of links* of a context node *C* defines a relationship among nodes recursively contained in C[11], or the context node *C* itself. We say that a link *l* is a *component of a context node C* or simply that *l is contained in C*. We also say that a link *l* is *recursively contained in C* if and only if *l* is contained in *C* or *l* is contained in a context node recursively contained in *C*. Links will be discussed in Section 2.8.

The context node *presentation collection* specifies a group of generic descriptors[12] for each node contained in the context node. As aforementioned, descriptors are used to join presentation information to the node, as will be defined in Sections 2.9 and 2.10. The goal of the presentation collection property is to allow defining a group of descriptors for each node contained in a context node. Actually, the group should form a set of descriptors (the same descriptor cannot appear more than once within a group)[13]. If the contained node is also a context node, the descriptor group should be composed of at most one descriptor.

Context nodes will be useful, among other things, to define a logical structure, hierarchical or not, for hypermedia documents. These entities will also be suitable for defining different views for the same document and for improving the user orientation when traversing a document.

A context node *C* shall implement the abstract method deferred from the composite node class:

- addNode: insert a content, switch or context node into the set of nodes of *C*.

Besides the getter and setter methods, the methods for manipulating the lists of anchors and ports, and the methods for manipulating the set of nodes, context nodes shall also offer methods for manipulating their link sets and presentation collections (e.g. add, remove, get, iterate, etc.).

2.6. Switch Nodes (Node Alternatives)

NCM has several features to support document adaptation. One important facility is the group of node alternatives, selected based on document *rules*. Figure 3 depicts the class diagram for NCM rules.

---

[10] Switch nodes will be presented in Section 2.6. They are another specialization of composition that allows defining node alternatives.

[11] As discussed in Section 2.8, relationships may have their participants defined through mappings towards nodes recursively contained in a composition *C*.

[12] The group may be empty for any context node constituent.

[13] The semantics behind the descriptor group definition for each node *N* is to allow the simultaneous presentation of the same node with different exhibition characteristics, as it will be clarified in Section 2.9. Moreover, as will be explained in Section 2.10, a descriptor in the group may be the result of a selection among descriptor alternatives (descriptor switch), whose choice depends on user and platform parameters.
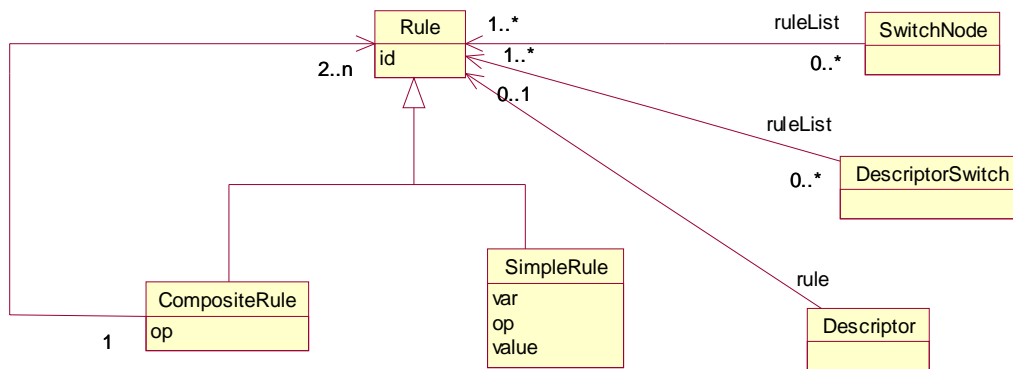
Figure 3 – Class diagram for rules

Based on contextual information (e.g. user preferences, platform characteristics, etc.)[14], an NCM document formatter should evaluate each rule to find whether a document entity associated with the rule should be considered in the document presentation. The way entities are associated with rules will be clarified further on.

A rule may be simple or compound. A *simple rule* is analogous to the connector statement expression that compares an assessment to a value (Section 2.8.1) and has three attributes: an identification (*var*) of the variable to be tested, a comparison operator (=, ≠, <, ≤, >, ≥) and a value. The *compound rule* is a logical expression involving two or more rules (simple or compound) related through the logical operators *AND* and *OR*.

In order to allow an author to specify node alternatives depending on contextual information (context attributes), NCM defines an entity called switch node. The *switch node* is a specialization of composite nodes. The content of a switch node is a set of content or context nodes. The switch node has an additional attribute that defines, for each node contained in its node set, an associated rule. The rules are defined in an ordered list. A document formatter should iterate the rule list in its order, evaluating each rule. The first node with the rule evaluated as true should be elected the *selected alternative*.

The switch node may contain a *default node*. During a document presentation, this node should be elected the switch selected alternative if none of the rules has been evaluated as true. In the absence of a default node and of a rule evaluated as true, none of the nodes contained in the switch should be played during the document presentation.

In addition to the ordered port list property common to every compositions (Section 2.4), switch nodes have an *ordered switch port list* property. Each *switch port* is an interface (Figure 2) that defines a set of mappings to interfaces of nodes contained in the switch node.

The definition of switch ports allows modeling links (relationships discussed in Section 2.8) that anchor on switch nodes, independent of the node that will be selected.

---

[14] As previously mentioned, this contextual information may be represented as attributes (properties) of the *settings* node.

13

The traditional composite node ports allow a link to touch a specific alternative. If this node alternative is not selected, the link will simply be ignored during the document presentation.

Like context nodes (Section 2.5), a switch node has a *presentation collection*, to allow defining a group of descriptors for each node the switch node contains[15]. Actually, the group should form a set of descriptors (the same descriptor cannot appear more than once within a group). If the contained node is a context node, the descriptor group should be composed of at most one descriptor. When containing a default node, the switch may also define a group of descriptors for this node, following all the rules mentioned in this paragraph.

The document formatter should decide when to evaluate rules and analyze switch nodes.

2.7. Events

As defined by Pérez-Luque and Little [PéLi96], an event is an occurrence in time that may be instantaneous or may extend over a time interval. NCM defines some basic classes of events, which may be further extended: presentation event, composition event, selection event, hovering event, drag event, focus event and attribution event.

A *presentation event* represents the exhibition of a node content-anchor (media segment), given a perspective and a specific descriptor. Thus, different perspectives or different descriptors for the same content node lead to different NCM presentation events. Presentation events may also be defined on context and switch nodes, representing the presentation of the information units of any node inside these composite nodes.

A *composition event* is defined by the presentation of the structure of a composite node. Composition events are used to present the composite map (composite organization).

A *selection event* represents the user selection of a node content-anchor, given a perspective and a specific descriptor. The common way for content selection is through the mouse or keyboard input devices, however other interaction input devices can be used, like TV remote controls. The *hovering*, *drag* and *focus* events also represent the corresponding interactive actions over a node content-anchor, given a node perspective and a specific descriptor.

An *attribution event* refers to a node attribute anchor, given a perspective and a specific descriptor.[16]

In NCM, each event defines a state machine that shall be maintained by the document formatter [Rodr03]. Figure 4 shows the NCM generic event state machine.

An NCM event may be in one of the following states: *sleeping, occurring*, or *paused*. Moreover, every event has an associate attribute, named *occurrences*, which counts how many times the event transits from *occurring* to *sleeping* state during a

---

[15] The group may be empty for any switch node constituent.

[16] It is important to remember that, as defined in Section 2.2, a node attribute anchor may refer either to a node attribute or to an attribute of the descriptor that was associated to present this node, as explained in Section 2.9.

document presentation. Events like presentation and attribution have also an attribute named *repetitions*. This attribute counts how many times an event should be automatically restarted by the formatter, and may contain the *indefinite* value, leading to an endless loop of the event until some external interruption.
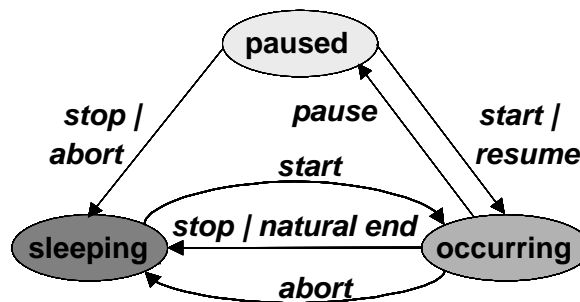

Figure 4 – NCM event state machine.

Intuitively, taking a presentation event as an example (see Figure 4), it starts 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. At the end of the exhibition, the event comes back to the sleeping state, when 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 goes to the *sleeping* state, without incrementing the *occurrences* attribute and setting the *repetitions* attribute value to zero. Selection events stay in the *occurring* state while the corresponding anchor is being selected. Similarly, *focus*, *drag* and *hovering* events stay in the *occurring* state while the respective operation over the anchor is being applied. Attribution events stay in the *occurring* state while the corresponding attribute values are being modified. Obviously, instantaneous events, like attribution events for simple value assignments, may stay in the occurring state only during an infinitesimal time.

A presentation event shall change from occurring to sleeping in two situations: as a consequence of the natural end of the presentation duration, or due to an action that stops the event.

The duration of an event is the time it remains in the occurring state. In the case of a presentation event, this duration may be intrinsic to the media object or specified by the event descriptor. The duration of a presentation event will be chosen by the document formatter taking into account content intrinsic parameters, descriptor parameters, document relationships (mainly links) and other external information, like platform characteristics.

A presentation event associated with a composite node 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.

A presentation event associated with a switch node 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 composition event stays in the occurring state while the composition map is being presented.

Links defined in context nodes indeed specify relationships among events defined on node anchors, more precisely, among event state machines, as will be discussed in the next section. In order to facilitate NCM link explanation, Table 1 defines state transition names and the action name to produce the corresponding state transition in NCM event state machines.

Table 1 - Transition and action names for NCM event state machines

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

2.8. Links

A *link* is an NCM entity that has two additional properties: a connector and a set of binds to this connector. Figure 5 presents the NCM link class hierarchy.

The connector is an entity introduced in NCM 3.0, which aims at defining hypermedia relation semantics, independently of the participants that will actually be interacting [MuRS02].

Connectors receive first-class status in the model [MuSo01], i.e., connectors may be defined independently of other model entities.

Links representing the same type of relation, but interconnecting different participants (nodes), may reuse the same connector.

A connector specifies a set of interface access points, called roles. An NCM link refers to a connector and defines a set of binds, which associate each link endpoint (node interface) to a role at the referred connector.
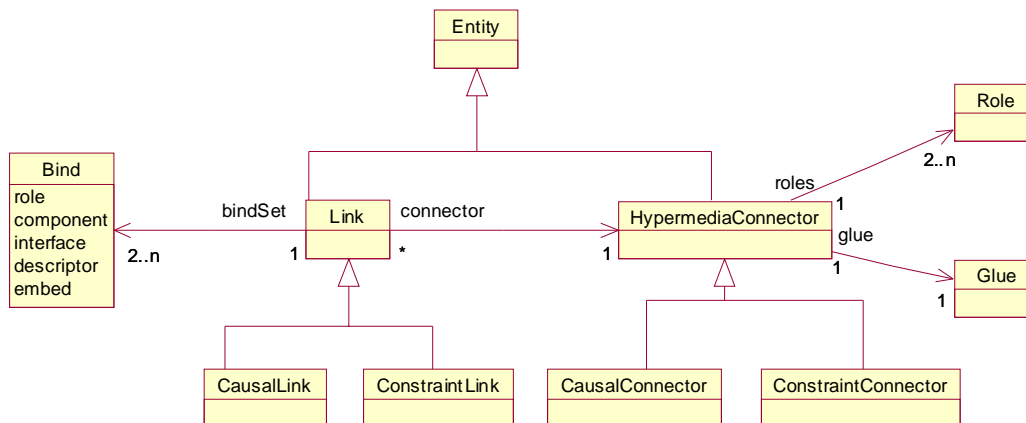
Figure 5 - NCM link class hierarchy.

2.8.1. Connectors

Figure 6 shows a connector named *R*, representing a relation with three different roles, which means three different types of participants. The figure also illustrates two different links, l_1 and l_2, reusing *R*. While a connector defines the relation type, the link set of binds specifies the interacting nodes. Link l_1 defines three binds interconnecting nodes *A*, *B* and *C* to the roles of connector *R*. Link l_2 also defines three binds, but it connects a different set of nodes (*B*, *C* and *D*). Links l_1 and l_2 define different relationships, as they relate different sets of nodes, but represent the same type of relation, as they use the same connector. In an NCM document specification, a link shall refer to a connector instance.
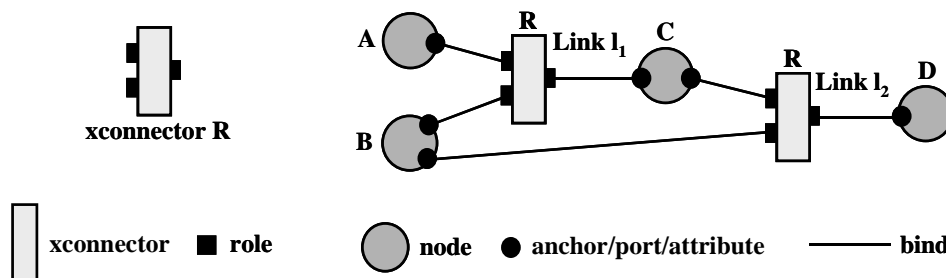
Figure 6 - Example of links using the same connector R

Conceptually, connectors may represent any type of relation, such as reference relations, synchronization relations, semantic relations, derivation relations, etc. This NCM version concentrates efforts in the specification of spatio-temporal synchronization relations and also reference relations[17], providing enough support for the creation of hypermedia documents.

The NCM *connector* allows the definition of multipoint relations with causal or constraint semantics. In a causal relation, a condition shall be satisfied in order to trigger an action. An example of causal relation is the traditional hypermedia reference relation, which causes the navigation to a target node when a source node anchor is selected. Another example of causal relation is one that starts a node presentation when another node presentation finishes. Besides causal relations, NCM connectors may also

---

[17] In fact, a reference relation is modeled as a special case of spatio-temporal synchronization relation.

specify constraint relations, with no causality involved. Consider, for example, a constraint specifying that one participant shall finish its presentation at the same time another participant begins its exhibition. The occurrence of the presentation of one participant without the occurrence of the presentation of the other also satisfies the constraint, which specifies that, if and only if these two participants are presented, their end and beginning times have to coincide.

In order to capture causal and constraint relations, connectors are specialized in causal and constraint types. In both types, the definition of a connector is done by a set of *roles* and a *glue*. The definition of roles is based on the concept of event (Section 2.7). Each role describes an event to be associated to a relation participant, while the glue describes how the events should interact, according to causal or constraint semantics.

Each connector role defines an *id*, which has to be unique in the connector role set, an event type and its cardinality. The *event type* specifies the name of one of the event class specializations. Table 2 describes the names for NCM event types. The role *cardinality* specifies the minimal and maximal number of participants that may play this role (number of binds) when this connector is used for creating a link, as defined later.

Table 2 - Definition of the names for specifying event types in NCM connectors

Event specialization	Name for the Event Type
Presentation event	*presentation*
Selection event	*selection*
Point over event	*pointOver*
Drag event	*drag*
Attribution event	*attribution*
Focus event	*focus*

Roles are specialized in action roles, condition roles and assessment roles. Different types of roles are used depending on the connector type. In constraint connectors, only the use of assessment roles is allowed. In causal connectors, any type of role may be used. Figure 7 illustrates the class hierarchy for NCM roles.
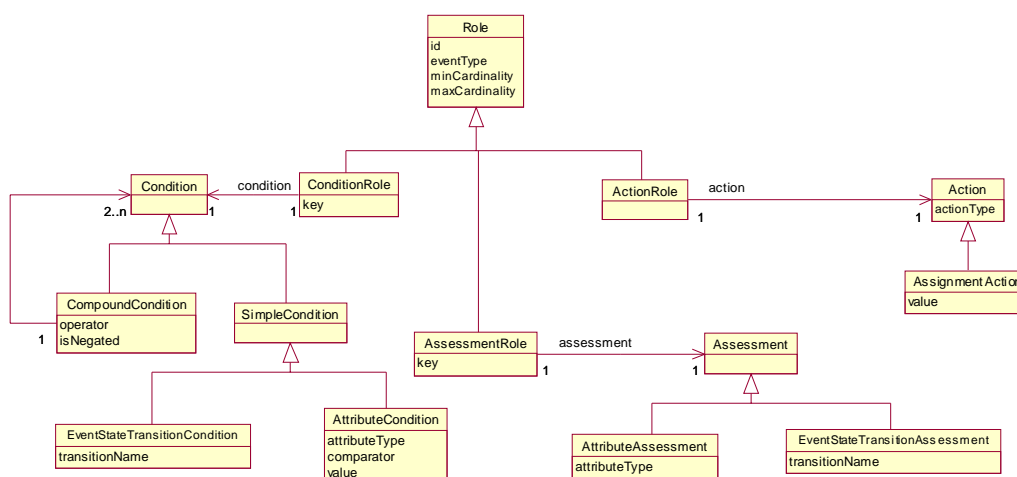
Figure 7 - Class hierarchy of NCM roles in NCM connectors.

Action roles capture actions that are triggered in causal relations. *Types* of actions are illustrated in Figure 4 by labeled arcs causing transitions in the event state

18

machine. Besides the action type attribute, an action may define a *value* to be assigned to a participant attribute (if the role event type is attribution). As an example, an action role can be: "pause the presentation of a content node".

NCM actions may be further extended. For instance, animation actions could be specified, defining an initial value, a final value and a duration for performing a node attribute assignment (e.g. the object x position in the screen, in a 2D environment).

In causal connectors, conditions shall be satisfied in order to trigger actions. Conditions are captured by the *condition role* type, which defines a logical expression evaluating event state transitions, event attribute values or node attribute values. When evaluated, a condition returns a boolean value.

Conditions may be simple or compound. A simple condition may test an event state transition, an event attribute value or a node attribute value. In the case of *event state-transition condition* the test result is true only at the moment the specified transition (*transitionName* attribute, defined as in Table 1) occurs. The *attribute condition* shall specify the attribute type (*attributeType*) to be tested: an event state or an event attribute (*occurrences* or *repetitions*) associated by a link to a node anchor; or the attribute type *nodeAttribute*, that shall be associated by a link to a node attribute. The referred attribute will be compared with the *value* attribute specified in the condition using one of the following comparators: $=$, $\neq$, $<$, $\leq$, $>$, $\geq$. The *attribute condition* obliges the role event type to be attribution, as discussed in Section 2.7. As for selection events, the condition role may additionally specify to which selection apparatus (for example, keyboard or remote control keys) it refers, through its key attribute.

A *compound condition* consists of a logical expression involving two or more other conditions over the same event and based on operators "and" or "or". A compound condition role example is "the presentation of a participant finished for the second time", which would be specified as "[(eventType = "presentation"), ((transition = "stops") AND (occurrences = "2"))]"[18]. Note that any compound condition may be negated using its *isNegated* attribute.

While a condition always returns a boolean value when evaluated, an assessment role contains an assessment that returns a value, depending on its type. An *attribute assessment* returns the value of an event attribute (*attributeType* equals to one of the event attributes: *occurrences* or *repetitions*), or the value of an event state (*attributeType* equals to *state*), when associated by a link to a node anchor; or returns a node attribute value (*attributeType* equals to *nodeAttribute*), when associated by a link to a node attribute. An *event-state transition assessment* returns the time instant an event state transition occurs (specified in the *transitionName* attribute). When referring to a selection event, the assessment role may additionally specify to which selection apparatus it refers, trough its key attribute.

As previously mentioned, a connector is defined by a set of roles and a glue, where the glue specifies how the roles interact. Every connector role shall be used in the glue. A constraint connector has a *constraint glue*, which defines a *statement expression* relating assessment roles. A causal connector has a *causal glue*, which defines a *trigger expression*, relating condition and assessment roles, and an *action expression*, composed

---

[18] Compound condition operators may be extended with other types, like temporal logic operators. Of course, these operators will need to be correctly interpreted by document formatters.

by action roles. When the trigger expression is satisfied, the action expression shall be executed. Figure 8 depicts the class hierarchy of NCM expressions defined for NCM connectors.

Figure 8 - Class hierarchy of NCM expressions in NCM connectors.

A statement expression may be simple or compound. A *simple statement* may compare either assessment roles of the same type (*assessment statement*) or an assessment role to a value of the same type of the assessment result (*assessment value statement*). An *offset* value may be added to an assessment role before the comparison. For example, an offset may be added to an assessment role to specify: "five seconds after the time instant a presentation event stops" or "the screen vertical position plus 50 pixels". The comparison may use the same comparators defined for simple conditions. For example, suppose that an event state-transition assessment role P specifies a presentation-event type and the "starts" transition, and that another event state-transition assessment role Q specifies a presentation-event type and the "stops" transition. If an assessment statement S_1 defines that "$P = Q$", S_1 will be evaluated as true if a presentation event bound to P starts at the same time another presentation event bound to Q stops[19]. As another example, suppose the assessment role H containing a node attribute assessment for the screen horizontal position attribute value. If a simple assessment value statement S_2 specifies that "$H \geq 100$", S_2 will be evaluated as true if the horizontal position of a participant playing the H role is greater than "100". When the maximal cardinality value of a role is greater than one, several participants may play the same role. In this case, a *qualifier* shall be defined each time this role is used in glue expressions, as will be discussed in Table 4 explanation.

A *compound statement* consists of a logical expression, based on the operators "and" or "or", involving two or more statement expressions. Compound statements may be negated.

Although statement expressions may be used in causal connectors, their main utility is in the specification of constraint connectors. The semantics of a constraint connector is that the statement expression shall be kept true during a presentation. Table 3 illustrates a constraint connector example expressing a spatial synchronization relation

---

[19] As aforementioned, if the first event presentation does not start, or the second event presentation does not stop the expression is also evaluated as true.

specifying "two nodes shall be horizontally aligned by their tops (their top attribute values defined in their descriptors shall be identical)".

Table 3 - Example of constraint connector

Role Type and id	Event type	Cardinality (min,max)	Attribute Name
Assessment P_1	*attribution*	*(1,1)*	*descriptor.top*
Assessment P_2	*attribution*	*(1,1)*	*descriptor.top*

Glue Type	Statement Expression
Constraint	$P_1 = P_2$

A trigger expression may be simple or compound. A *simple trigger expression* refers to a condition role. A *compound trigger expression* consists of a logical expression, based on the operators "and" or "or", involving trigger or statement expressions, moreover a *compound trigger expression* may be negated. Any trigger expression may specify *minimal* and *maximal delays* to its evaluation. For example, given that a trigger expression C is true at instant t, C' defined as C with minDelay="*t1*" and maxDelay="*t2*" is true in the interval $[t+t1, t+t2]$[20].

Compound trigger expressions may relate any number of condition and assessment roles (through condition and statement expressions). However, one restriction is necessary to guarantee causal relation consistency. Every trigger expression associated to a causal connector shall be satisfied only at an infinitesimal time instant; requiring at least one event state-transition condition role.

An action expression may also be simple or compound. A simple action expression refers to an action role. If the action role of a simple action expression is played by an event of presentation or attribution type, a *repeat* attribute may be assigned to the repetitions attribute of the event, and also a *repeat delay* to be waited before repeating the action. A compound action expression consists of an expression, based on the operators "*par*", "*seq*" or "*excl*", involving two or more action expressions. Parallel (*par*) and sequential (*seq*) compound actions specify that the execution of actions shall be performed in any order or in a specific order[21], respectively. Exclusive (excl) compound action specifies that only one of the actions shall be fired. In the latter case, the document formatter should decide by itself which one is to be performed or ask the user to decide.

Any action expression may also specify a *delay* to be waited before executing its actions.[22]

---

[20] Note that the temporal behavior of NCM relations may also be obtained using NCM time content nodes (Section 2.3), instead of exploring connector delay parameters.

[21] 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.

[22] An application based on NCM can allow the parameterization of this value and other attributes present in glue expressions and roles. In NCL, for instance, a same connector specification can be reused with different parameter values, deriving different NCM connectors. Indeed, the NCL parameterization is used not only for action attributes, but also for assessment and condition attributes, specified either in the connector roles or in the connector glue expressions. Parameterization, however, is an implementation issue and not a model issue, so it is left to application definitions, as discussed in Part 5: NCL (Nested Context Language).

As previously mentioned, when the maximal cardinality value of a role is greater than one, several participants may play the same role. In this case, a *qualifier* attribute specifies the role behavior. Table 4 presents possible qualifier values.

Table 4 - Role qualifier values

Role type	Qualifier	Semantics
Condition	*all*	All conditions shall be true
Condition	*any*	At least one condition shall be true
Assessment	*all*	All assessments shall be considered
Assessment	*any*	At least one assessment shall be considered
Action	*par*	All actions shall be executed in parallel
Action	*seq*	All actions shall be executed in parallel, but respecting the order the participants have been associated to the role.
Action	*excl*	Only one of the actions shall be executed

Table 5 illustrates a causal connector example expressing a temporal synchronization relation. The connector specification is interpreted as "if a group of participants is being presented (C_1) and another participant is selected (C_2), stop the presentation of a group of participants (A_1) and start the presentation of another participant". In order to stop the presentation of the same group of participants that played role C_1, a link using this connector shall create two binds for each participant in the group, one to role C_1 and another to role A_1.

Table 5 - Example of causal connector

Role Type and id	Event Type	Cardinality (min,max)	Condition	Action
Condition C_1	*presentation*	*(1, unbounded)*	*state=occurring*	
Condition C_2	*selection*	*(1, 1)*	*transition=stops*	
Action A_1	*presentation*	*(1, unbounded)*		*stop*
Action A_2	*presentation*	*(1, 1)*		*start*

Glue Type	Trigger Expression	Action Expression
Causal	*all(C_1) AND C_2*	*seq(par(A_1), A_2)*

Since the definition of connectors is not simple to be done by non-expert users, because they would need to have the notion of event states and event state-transitions, the idea is to have expert users defining connectors, storing them in libraries (*connector bases*) and making them available to others for creating links[23].

2.8.2. Link Binds

As aforementioned, links are defined inside a composition (indeed, until now, inside a context node). A *link* refers to a connector and defines a set of *binds*, which associate each link endpoint (node interface) to a role of the referred connector. Binds are constrained to directly associate interfaces of nodes inside the composition *C* where the

---

[23] As an example of connector base, consider the thirteen well-known synchronization relations defined by Allen [Alle83]. Although Allen specified a complete set of all possible relationships that can exist between two intervals, they do not precisely express the causal or constraint semantics that should exist between the time intervals [DuKe95]. For instance, the meets relation only specifies that the end of interval *x* should coincide with the beginning of *y*, but several interpretations could be given. The meets relation could be considered either a simple constraint, or a starting causality (the end of *x* shall cause the start of *y*), or even a stopping causality (the beginning of *y* shall cause the stop of *x*). When using NCM connector, the author can choose and define the exact semantics (s)he wishes to express through the relation, allowing an unambiguous specification.

link was defined, or anchors of the composition *C*. However, since a composite-node port may be mapped to another inner composite-node port, and so on until a node anchor is reached, links defined in a composition *C* may indirectly associate events defined in any node recursively contained in *C* to its connector roles. This brings the notion of visible links and contextual links, defined as follows.

Recalling that different NCM composite nodes can contain the same node, and that links can be defined in any composite node in the node perspective, it is necessary to identify which links effectively anchor on a node, or "cross" a node (in the case of composite nodes), in a given perspective. The set of links that anchor on the node in a given perspective *P* is called *contextual links of P*. The set of links that anchor on or cross a composite node in a given perspective *P* is called *visible links of P*.

More precisely, given a node N_1 and a perspective $P = (N_m,....,N_1)$, with $m>0$:

1. A link *l* is *visible in P* if and only if there is a composite node N_i, for $i \in [1,m]$, such that N_i occurs in *P*, N_i contains *l* and:

   i) if *i>1* thus $(N_{i-1}, p,...,N_1, p_1)$ defines a *port mapping sequence* (Section 2.4) of a port *p* of composition N_{i-1}, and *p* is bound to a role of the connector used by *l*; or

   ii) N_1 is a node that has an interface directly bound to a role of the connector used by *l*.

2. A link *l* is *contextual in P* if and only if there is a composite node N_i, for $i \in [1,m]$, such that N_i occurs in *P*, N_i contains *l* and:

   i) if *i>1* thus $(N_{i-1}, p...,N_1, p_1)$ defines a *port mapping sequence* of a port *p* of composition N_{i-1}; N_1 contains an anchor p_1 that is in the port mapping sequence of *p*, and *p* is bound to a role of the connector used by *l*; or

   ii) N_1 is a node that has an anchor directly bound to a role of the connector used by *l*.

For example, suppose that nodes *A* and *Z* contain node *B*, which contains nodes *C*, *D*, *E* and *F* with the links illustrated in Figure 9. Thus, the presentation of node *C* through the perspective (*A*, *B*, C) will show a link from anchor *i* of *C* to anchor *j* of *E*, and a link from anchor *m* of *C* to anchor *n* of *F*, defined in *A* and *B*, respectively. Both are the visible and contextual links in (*A*, *B*, *C*). The presentation of node *B*, through the perspective (*A, B*) will show a link from *B* to *B*, which is defined in *A*. This is the only visible link in (*A*, *B*); there is no contextual link in (*A*, *B*).
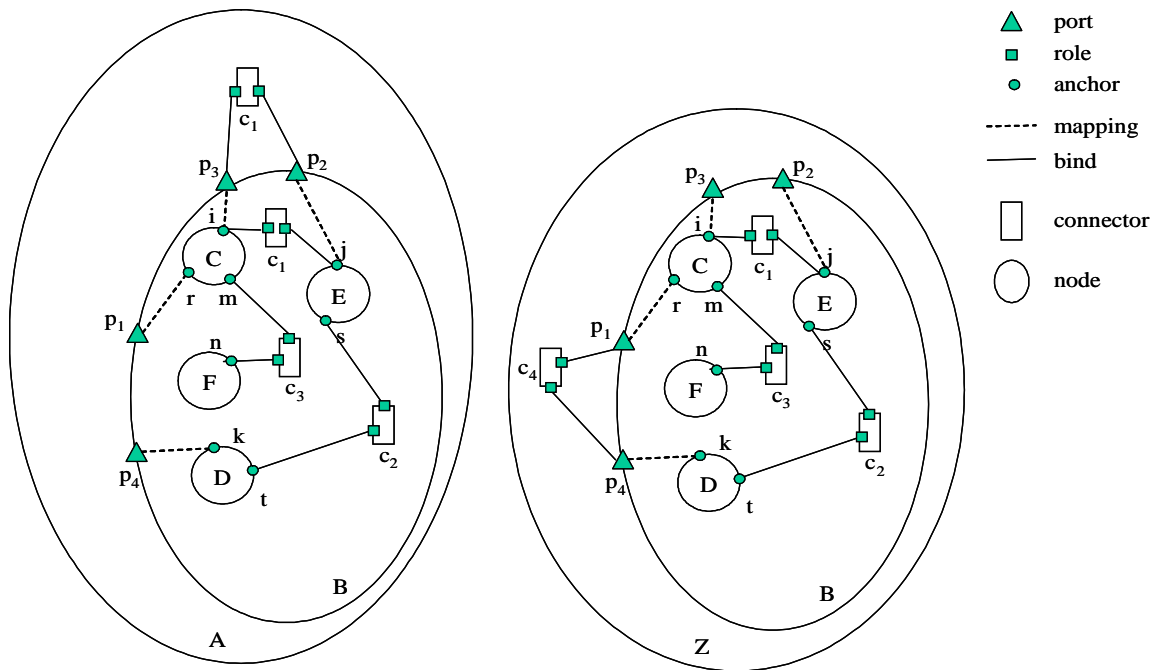
Figure 9 – Examples of visible and contextual links in NCM.

On the other hand, the exhibition of node C through the perspective (Z, B, C) will show a link from anchor r of C to anchor k of D and a link from anchor m of C to anchor n of F, defined in Z and B, respectively. Both are the visible and contextual links in (Z, B, C). The presentation of node B, through the perspective (Z, B) will show a link from B to B, defined in node Z. This is the only visible link in (Z, B); there is no contextual link in (Z, B).

Link binds have other attributes besides those used to associate an interface to a role: *descriptor* and *embed*. A *descriptor* attribute is optional and specifies a generic descriptor for the node associated with the connector role. In the case where the associated node N is a composite node, the descriptor is null. Note that several binds may be made to the same content node with different generic descriptors, leading to the simultaneous presentation of the same node with different exhibition characteristics, similar to the depth navigation discussed in Section 2.4.

The *embed* attribute of a bind is a boolean attribute and is used only when associating an action role (only for *start* or *prepare* actions) with a presentation event E. If the bind descriptor refers to a player (Section 2.10) that is already being used to control the presentation of another event F, then the player is requested to also control E, without stopping F, if the embed attribute is true, otherwise, if *embed* is *false*, the player is requested to replace F by E. When not specified this attribute should be considered as false.

The definition of switch ports, presented in Section 2.6, allows modeling links (relationships) that anchor on switch nodes, independent of the node that will be selected, as illustrated in Figure 10. The traditional composite node ports allow a link to touch a specific alternative. If this node alternative is not selected, the link will simply be ignored during the document presentation.
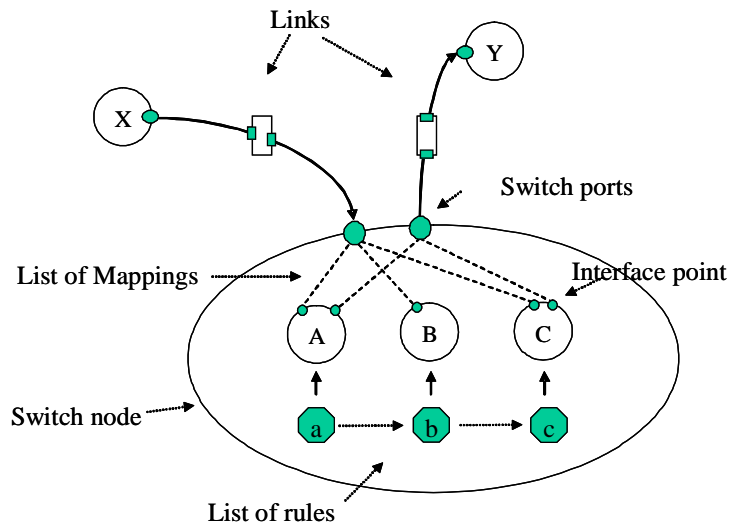
Figure 10– Switch nodes, switch ports and links.

Based on NCM switch nodes, switch ports and link binds, an author can also specify link adaptations. For example, one could insert the same node more than once into a switch node with different rules and links associated to each occurrence. Figure 11 depicts an example of switch node usage to adapt document relationships. In the example, the link that starts the presentation of node Z will be enabled only if rule a is evaluated as false and rule b is evaluated as true.
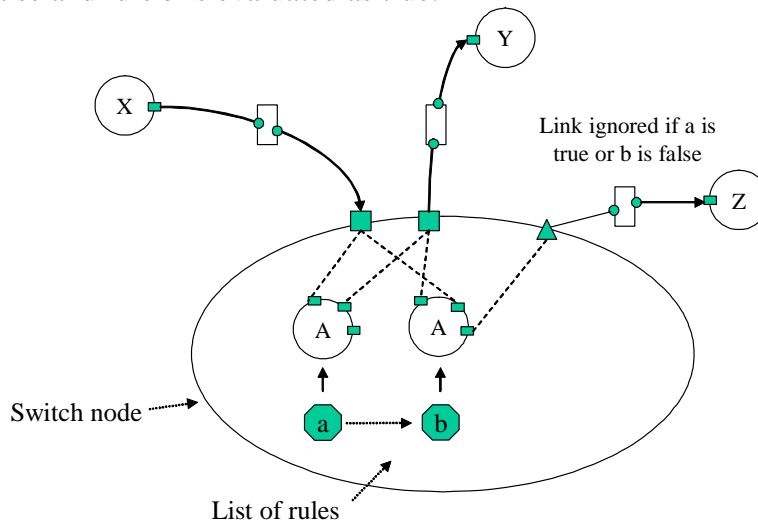
Figure 11 – Support to link adaptation in NCM.

2.9.   Data Objects versus Presentation Objects

NCM defines a *data object* as an entity that comprises an NCM node with all operations for manipulating this node, except operations related with the presentation of node content (content information units defined in Section 2.2). The functionality for presenting the node content is given by a descriptor instance that shall be associated to the node (data object).

The aggregation of a data object and an NCM descriptor is called a *presentation object*. The association between data objects and descriptors is illustrated in Figure 12 through dashed lines connecting objects in the lower plane with the ones in the upper

plane. In the figure, nodes are represented by circles, links are illustrated by arcs and compositions are represented by bigger circles containing other circles and arcs.
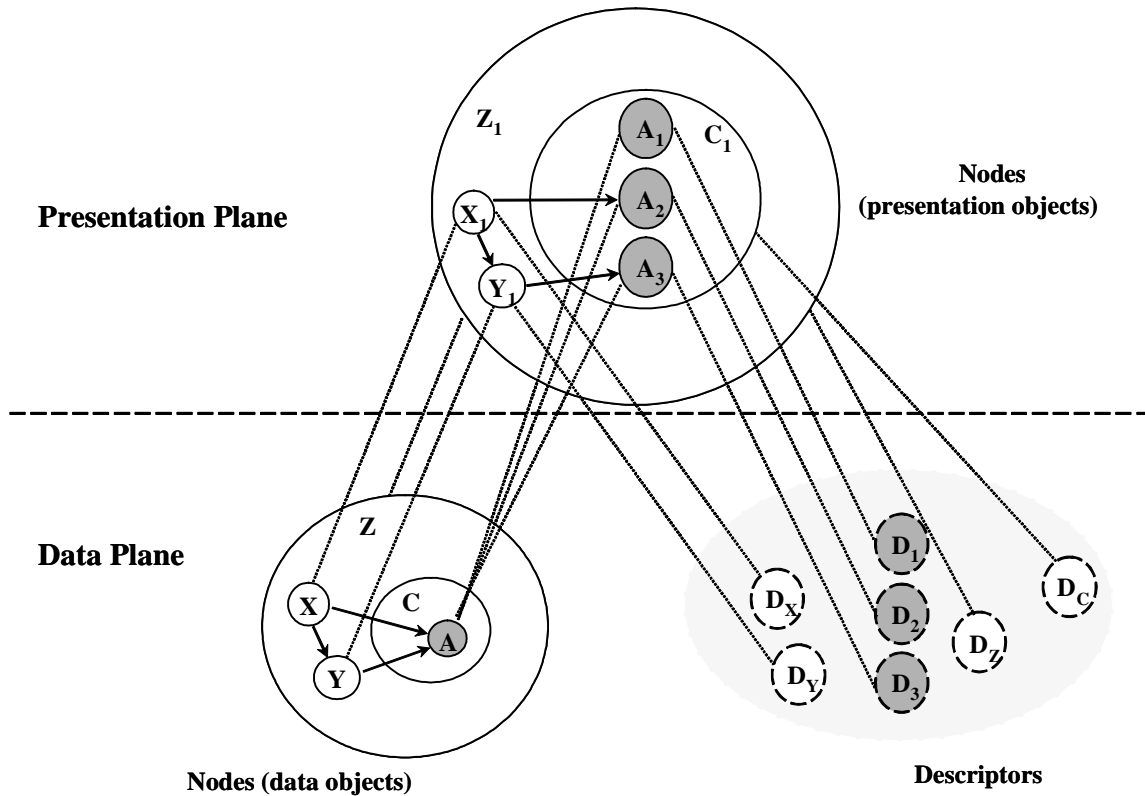
Figure 12 - Association between data objects and descriptors generating presentation objects

Note that one node can be combined to different descriptors, originating different presentation objects of the same node. Figure 12 shows this feature with the association of descriptors D_1, D_2 and D_3 to the data object A, originating presentation objects A_1, A_2 and A_3. Node A has three different presentations because there are, for example, three different ways to access it: one can reach A by link navigation (through one of the two links that touch the node) or by depth navigation (through composite node C). Therefore, due to the possibility of one data object generating several presentation objects, the presentation object of a context node may contain a different number of elements from the corresponding data object context node (e.g. context C_1 versus context C in the figure).

As previously defined, a descriptor may be specified as a node property, as part of a descriptor group in the composition containing the node or as an attribute of a link bind. Besides that, default descriptors may be specified for node classes (text, image, etc.) or explicitly suggested by the document reader (user). If a node has more than one of these descriptors specified, the presentation system (NCM document formatter) should build the resulting descriptor based on the following cascading rule.

Suppose a node N of class C with the current perspective (C_k, …, C_1, N) reached through a link navigation (link l). Let D_1 be a descriptor defined for the node class C, D_2 be the descriptor defined by the descriptor property of N, D_3 be a member of a group of descriptors specified for N in C_1, and D_4 be a descriptor specified by in the bind used to associate N link l connector. The resulting descriptor will be formed by the sum of all attributes/properties of these descriptors (D_1, D_2, D_3, D_4). If two or more descriptors define the same attribute/property with different values, D_4 information will have

priority over D_3, which will have priority over D_2, which in turn will have priority over D_1. Finally, if the user specifies a fifth descriptor when navigating to N, this descriptor would be included in the cascading list with precedence over D_4.

2.10. Generic Descriptors, Descriptors and Descriptor Switches

As aforementioned, descriptors are NCM entities in charge of grouping presentation characteristics, aiming at separating this information from the document content and structure.

NCM defines a *generic descriptor* class that is further specialized in *descriptor* and *descriptor switch* classes. The NCM descriptor has as additional information the following optional properties: a *start presentation specification*, an *end presentation specification* and a *collection of event descriptions*.

The *start presentation specification* property should contain an identifier to a formatter presentation tool (a *player*). This tool will be responsible for rendering the node content during the presentation. When the player is not specified or points to a presentation tool that is not available in the formatter, the NCM document formatter should choose one based on the node content type.

The start presentation specification may also contain an attribute specifying if a new player shall be instantiated or if an already instantiated player shall be used. Moreover, the start presentation specification may contain a *set of particular attributes* or parameters that will be interpreted by the player selected for controlling the node presentation. Some examples of these parameters are:
−   for nodes with visible presentations, like text node, video node and image node, a *device* parameter specifies the device where the presentation will take place; *spatial region* parameter specifies a location to present the node content, identifying a position in a screen of the specified device; etc. When these parameters are not specified, the document formatter should choose a default spatial device and area for presenting the node.
−   for audio nodes, a *device* parameter specifies the audio device where the sound will be played; a *volume* parameter specifies the initial playing volume; etc[24]. When these parameters are not specified, the document formatter should choose a default audio device and volume for presenting the node.
−   for a text node to be played in a TtS (*text to speech*) player, parameters may specify the desirable voice language, accent and gender.

The *end presentation specification* property specifies what actions shall be done at the end of the presentation. It should contain an attribute specifying what will happen to the presentation tool at the end of the presentation, that is, if the *player* will be closed or will remain opened, ready for the next presentation. In this last case, it shall also specify if the player will remain hidden or not.

An *event description* in a descriptor consists of the tuple *<AnchorId, ExplicitDuration, DurationCostFunction, Rep>*. *AnchorId* identifies an anchor of the data object to which the descriptor will be associated (this anchor can be generically identified by a label or its position in order to allow the descriptor to be reused by more than one node). *ExplicitDuration* is optional and overrides the node presentation ideal

---

[24] The author can choose to present a visual information associated to an audio, for instance, a temporal progress bar. In this case, the parameters described in the first bullet should also be specified.

duration. *DurationCostFunction* is also optional provides metrics to guide the formatter on how to adjust the corresponding event duration. *Rep* specifies a value to initialize the repetitions attribute of the event (as discussed in Section 2.7). In particular, every descriptor contains at least the description event *<all content anchor, ExplicitDuration, DurationCostFunction, Rep>* associated to the exhibition of the entire node content.

Similar to the switch node entity (Section 2.6), the descriptor switch contains an ordered list of rules, where each rule is associated to a descriptor. The document formatter should traverse the list evaluating each rule and selecting the first descriptor whose associated rule is satisfied. If none of the rules is satisfied, the descriptor switch may specify a *default descriptor* to be selected.

The descriptor switch entity allows document formatters to adapt node presentation characteristics independent of structure and content information. Obviously both kinds of flexibility (presentation and content adaptation) may be combined. Together with duration adjustments, these model characteristics lead to a flexible support for authors specifying adaptive hypermedia documents and adaptive hypermedia presentations.

2.11. Trails

Issues such as large number of nodes, large number of links, several changes in the document, bad response time for user actions, insufficient visual differences between nodes and links, and visually disoriented users combine to hinder document navigation mechanisms. Disoriented users need context information to reestablish the sense of orientation. In particular, temporal context information is needed to answer questions like: "How did I get here?" These questions are answered by introducing the concept of trail.

Given a context, a private base or a public hyperbase[25] composite node *C*, a *trail T* for *C* is a composite node whose content is an *ordered* list of content, context and trail nodes, such that: all nodes that are not trails are recursively contained in *C*, and all trails are trails for *C*. Moreover, *T* has an additional basic property named *current node*, whose value is the position of a node in the ordered list of *T*. This pointed node is called *current entity of T*. The trail has another additional property named *view*, whose value associates each node *N* (in the ordered list) that is not a trail with a node nesting $(N_m,....,N_1)$, $m \geq 1$, and a descriptor *D*, such that: $N_1 = N$, $N_m = C$, N_{i+1} is a composite node, N_i is contained in N_{i+1}, for $i \in [1,m)$. We say that trail *T is associated with C*.

Every trail shall implement the deferred method of the composite node class:

- addNode: inserts a node in the trail node list in the specified position with an associated view.

Additionally, every trail shall implement the following methods:

- next: if the current node attribute does not point to the last node in the trail list, then it increments the current node attribute value;

- previous: if the current node attribute does not point to the first node in the trail list, then it decrements the current node attribute value;

- home: sets the current node attribute pointing to the first entity in the trail;

---

[25] Private bases and public hyperbase are defined in the next section.

- last: sets the current node attribute pointing to the last entity in the trail;

- enable: enables the *forward*, *back*, *home* and *last* methods and disables the *addNode* and *removeNode* methods;

- disable: disables the *forward*, *back*, *home* and *last* methods and enables the *addNode* and *removeNode* methods;

The reason for defining the *enable* and *disable* methods is to prevent a trail from being simultaneously used for navigation (through the trail) and for maintaining the navigation history. The same trail may be used for one task or the other.

Note that a node may appear more than once in the ordered list of *T*. Moreover, each node occurrence is associated with a node nesting from the point of view of *C*. Trails are useful for lining up hypermedia documents and for implementing *guided tours*. A special *system trail* may keep track of all navigation made during a work session, so that a user can move at random from node to node and go back step by step. Finally, it must also be remarked that, if a node *N* contained in a switch node *S* pertains to a trail, this node shall be the selected node of *S* when navigating through the trail, independent from the evaluation result, at the trail navigation moment, of the switch rule associated to *N*.

The definition of trails is important for hypermedia systems, since they represent ordered navigation. Through the trails, authors can provide an ordering for reading, which helps readers that are not familiarized with the document. Authors can also provide an appropriate presentation order to a certain audience. Users tend to feel less disoriented when they follow an already defined trail, since they have a limited number of options to choose from when navigating.

According to the NCM model, a possible implementation for trails to maintain navigation history is to create a *main trail* (or *system trail*). Every time a user navigates to a node, this node, its perspective and the resultant descriptor (cascading descriptor) are inserted in the main trail by the system. If the user decides to navigate through the trail, a copy of the main trail is created and the enable method is called for this copy. Even when navigating through this trail copy, the main trail can be updated in order to maintain the navigation history. If the user tries to go to a node without respecting the trail sequence, the trail copy should be destroyed.

2.12. Public Hyperbase and Private Bases

The public hyperbase is an NCM concept that represents the global public repository of available entities in a hypermedia system. The *public hyperbase* is a unique composite node such that if it contains a composite node *C*, then it also directly contains all nodes recursively contained in *C*. The public hyperbase composition has, as additional basic property, a set of descriptors. The descriptors in the *set of descriptors* are those used to create presentation objects (see Section 2.9) from the set of nodes contained in the public hyperbase.

A *private base* is a special type of composite node, such that:

i)   it may contain content, context, switch, trail and private base nodes;

ii) a private base may be contained in at most one private base;

iii) if a composite node is contained in a private base *PB*, its components are either contained in *PB*; or in the public hyperbase; or in any private base recursively contained in *PB*.

Private base compositions have, as additional basic properties, a set of links and a set of descriptors. Each *link l* contained in the *set of links* of a private base node *PB* defines a relationship among nodes recursively contained in *PB*[26]. The descriptors in the *set of descriptors* of a private base are those used to create presentation objects (see Section 2.9) from the set of nodes recursively contained in the private base.

Intuitively, a private base collects all entities used during a work session by a user.

---

[26] As discussed in Section 2.8, relationships may have their participants defined through mappings towards nodes recursively contained in a composition *C*.

3. 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. This technical report deals with basic model entities that comprise the NCM core.

As a last note, it should be mentioned that it is also possible to have NCM implementations that ignore some of the basic entities, but this is not so relevant to deserve a minimum-core definition.

References

[Alle83] Allen J.F. "Maintaining Knowlegde about Temporal Intervals". Communications of the ACM, 26(11), November 1983, pp. 832-843.

[BMRS04] Bachelet B., Mahey P., Rodrigues R.F., Soares L.F.G. "Elastic Time Computation in QoS-Driven Hypermedia Presentations", Technical Report of TeleMídia Lab., Departamento de Informática, PUC-Rio, Brazil, May 2004.

[DuKe95] Duda A., Keramane C. "Structured Temporal Composition of Multimedia Data". Proceedings of the IEEE International Workshop on Multimedia Database Management Systems, Blue Mountain Lake, USA, August 1995.

[HaSc90] Halasz F.G., Schwartz M. "The Dexter Hypertext Reference Model". NIST Hypertext Standardization Workshop. Gaithersburg. January 1990.

[MuSo01] Muchaluat-Saade D.C., Soares L.F.G. Hypermedia Spatio-Temporal Synchronization Relations Also Deserve First Class Status, VIII Multimedia Modeling Conference - MMM'2001, Amsterdam, Netherlands, November 2001.

[MuRS02] Muchaluat-Saade D.C., Rodrigues R.F., Soares L.F.G. "XConnector: Extending XLink to Provide Multimedia Synchronization". ACM Symposium on Document Engineering - DocEng'02, Virginia, USA, November 2002.

[PéLi96] Pérez-Luque M.J., Little T.D.C. "A Temporal Reference Framework for Multimedia Synchronization". IEEE Journal on Selected Areas in Communications (Special Issue: Synchronization Issues in Multimedia Communication), 14(1), January 1996, pp. 36-51.

[Rodr03] Rodrigues R.F. "Formatação e Controle de Apresentações Hipermídia com Mecanismos de Adaptação Temporal". PhD Thesis, Departamento de Informática, PUC-Rio, March 2003.

[Soar00] Soares L.F.G. et al. "Modelo de Contextos Aninhados versão 2.2", Relatório Técnico do Laboratório TeleMídia, PUC-Rio, Rio de Janeiro, Brazil, 2000. (*in Portuguese*)

[SoCR95] Soares L.F.G., Casanova M.A., Rodriguez N.R. "Nested Composite Nodes and Version Control in an Open Hypermedia System", International Journal on Information Systems; Special issue on Multimedia Information Systems, 20(6):501-520, Elsevier Science Ltd. England, September 1995.

[SoCR91] Soares L.F.G., Casanova M.A., Rodriguez N.R. Modelo de Contextos Aninhados. Relatório Técnico PUC-Rio - Departamento de Informática. Rio de Janeiro. May de 1991. (*in Portuguese*)

[SSRM99] Soares L.F.G., Souza G.L., Rodrigues R.F., Muchaluat-Saade D.C. "Versioning Support in the HyperProp System". Multimedia Tools & Applications, 8(8), May 1999.

[XLin01] "XML Linking Language (XLink) Version 1.0", W3C Recommendation, June 2001.

Appendix A : Examples of Link Usage

Let us consider a working document of a Drama Research Team about English Poetry of the XVI Century. Assume that the document is modeled as a user context node E containing a user context S, grouping plays by Shakespeare, and another user context node M, grouping sonnets by Christopher Marlowe. Assume also that S contains the text nodes H and L, representing the plays "Hamlet" and "King Lear", and that M contains a text node F representing "Dr. Faustus". Suppose the group wants to register a connection between "Hamlet" and "Dr. Faustus" (such a link could be used, for example, to register a connection between plays where the main theme is conflict). The idea is to create a link l_1 with the following behavior: when the user selects (clicks over) an anchor i (for example, a sentence) of H, the sentence where the same concept appears for the first time (anchor j) in F is presented, replacing the exhibition of H. This link may be defined in E, but requires the creation of a hyperlink connector and the specification of two ports in compositions S and M. Table 6 shows the connector cH specification, while Figure 13 illustrates the context organization and port mappings.

Table 6. Example of hyperlink causal connector

Role Type and id	Event Type	Cardinality (min,max)	Condition	Action
Condition C	*selection*	*(1, 1)*	*transition=stops*	
Action A_1	*presentation*	*(1,1)*		*stop*
Action A_2	*presentation*	*(1, 1)*		*start*

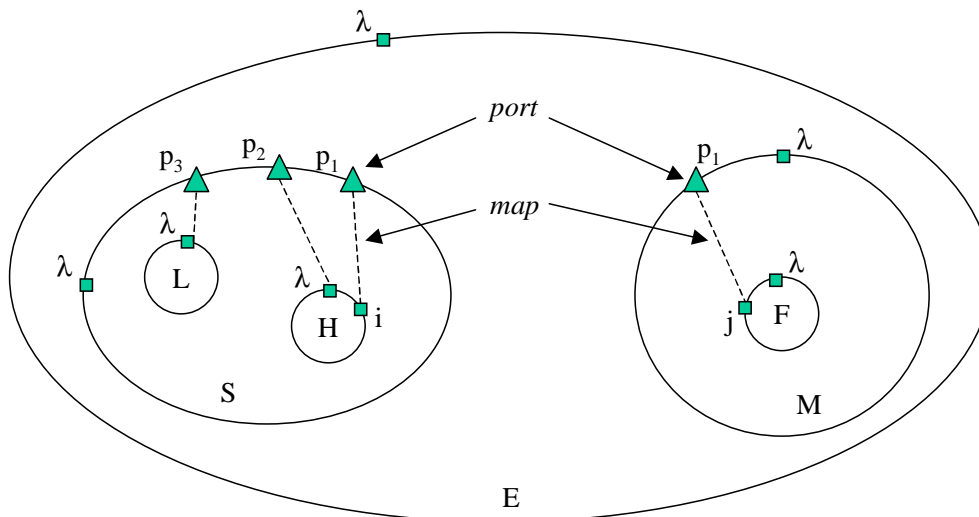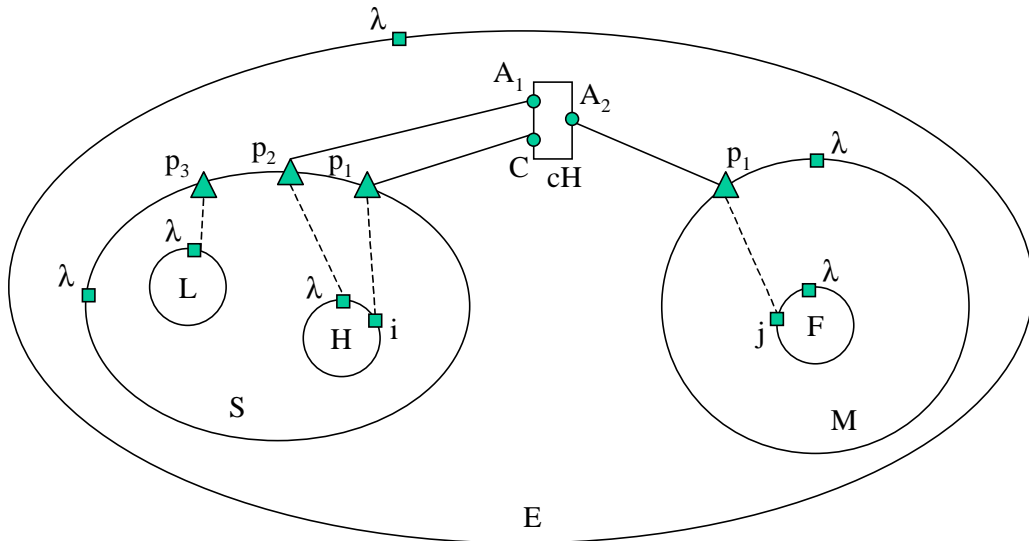| Glue Type	Trigger Expression	Action Expression
Causal	*C*	*seq(A_1, A_2)*

Figure 13 - Example of Drama Research Team document

The link l_1 would then be created in E using connector cH and establishing three binds: $<(S, p_1), C>$, $<(S, p_2), A_1>$ and $<(M, p_1), A_2>$, as illustrated in Figure 14.

Figure 14 - Link creation in the example of Drama Research Team document.

As another example, illustrated in Figure 15, since S contains H and L, a link connecting these nodes may, in principle, be defined in S with a connector c_1 and the following binds to this connector: $<(H, \lambda), C>$ and $<(L, \lambda), A>$, where C and A are role identifiers defined in c_1. If one wants to create a link in E connecting H and L with the same connector, one may define the link with binds $<(S, p_2), C>$ and $<(S, p_3), A>$. Note the difference between defining the link in S and in E. A link defined in S will be seen by every document which includes S (the user context node grouping plays by Shakespeare will probably be shared by several documents), while a link defined in E will be seen in S only by the readers of document E.
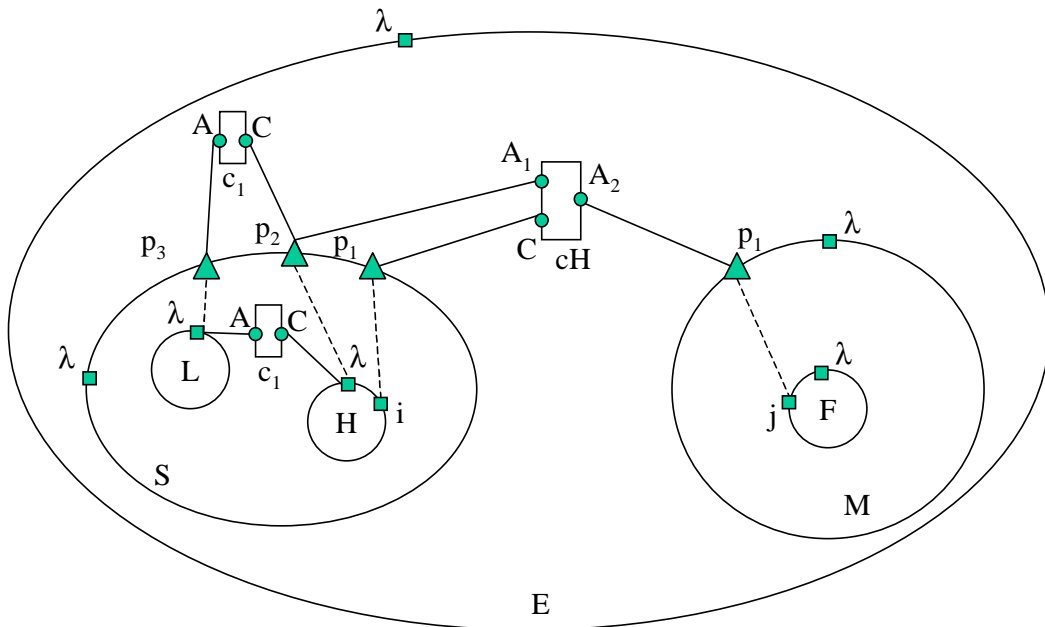
Figure 15 - Defining links with the same behavior in different contexts.

Figure 16 offers a less polluted view of document presented in Figure 15. In this drawing, connectors and port mappings are implicit defined in the link arrows. Since we

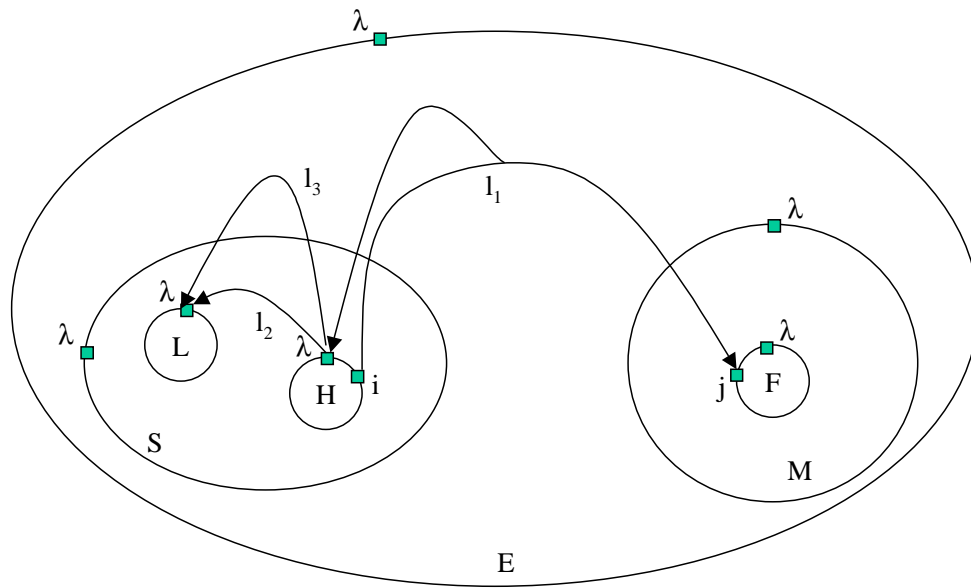used only causal connectors, we can specify a direction from conditions to actions (link arrows).

Figure 16 - NCM causal links abstracting the connector and port concepts.