ANCHORS AND LINKS FOR NESTED COMPOSITE NODES

LUIZ FERNANDO GOMES SOARES
Depto. de Informática, PUC-Rio. R. Marquês de São Vicente 225
22453-900 - Rio de Janeiro, Brasil
lfgs@inf.puc-rio.br

MARCO ANTÔNIO CASANOVA
Centro Científico Rio, IBM Brasil. Av. Pres. Vargas 824
20071-001 - Rio de Janeiro, Brasil
casanova@vnet.ibm.com

GUIDO LEMOS DE SOUZA FILHO
DIMAp, UFRN. Campus Universitário, Lagoa Nova
59072-970- Natal-RN, Brasil
guido@dimap.ufrn.br

This paper focuses on the definition of anchors and links for nested composite nodes, that is, composite nodes containing other composite nodes. It considers questions such as link inheritance and node reference, which become fairly complex in the presence of nested compositions. Then, the paper extends the discussion to include virtual anchors and links, defined by expressions in an appropriate language. Finally, it addresses some issues related to anchors and links in the presence of object versioning, such as how to maintain automatic reference to the last version of a node.

1 - Introduction

The definition of anchors and links for composite nodes poses serious problems to the formal definition of a conceptual model for hypermedia systems. The complexity is further increased if the model: (1) allows nested composite nodes, that is, composite nodes containing other composite nodes; (2) permits the same node to be contained in different composite nodes; (3) includes virtual objects, that is, objects defined by expressions in an appropriate language; and (4) provides for object versioning.

Indeed, nested composite nodes alone raise interesting questions, such as: How to refer to nodes inside nested composite nodes? How to define links between nested composition nodes? Can a link be inherited by the nodes nested inside a composition node? How to define link inheritance? Virtual objects and object versioning bring up other questions, such as: How to define virtual links and

anchors? How to refer to a particular version of a node? How to maintain automatic reference to the last version of a node?

This paper focuses on the definition of anchors and links for nested composite nodes, answering questions such as those posed in the previous paragraph. It also considers the problem of defining virtual anchors and links, and addresses issues related to anchors and links in the presence of object versioning.

Our definition of composite nodes follows [1] and is adopted in the HyperProp system.[2] It generalizes the context nodes introduced in the Neptune system,[3,4] which were in turn based on some ideas from PIE,[5] Intermedia webs,[6] NoteCard fileboxes and browsers,[7] and the Tree Items of KMS.[8] Also, HyperPro contexts[9] and HyperBase composite objects[10] are very similar to HyperProp composite nodes. Our definition of anchors provides the same facilities as Intermedia and Neptune (HAM), allowing anchors to point to regions inside both source and target nodes. However, it differs from that of KMS, NoteCard and HyperCard, where the target anchor must be a whole node. We model anchors as node attributes, unlike HAM, that treats anchors as link attributes. Therefore, changes to the content of a node do not require modifying links in our model, but they might do so in HAM. References[4,9,11,12] specifically address object versioning in hypermedia systems. In particular, HyperProp treats version groups as composition nodes, just as HyperPro and CoVer do.

The paper is organized as follows. Section 2 briefly presents a general conceptual model with composite nodes and defines anchors and links in this context. Section 3 deals with virtual links and anchors and addresses object versioning. Section 4 is reserved to the conclusions.

2 - A Conceptual Model with Composite Nodes

2.1 Nodes

The class hierarchy for a basic conceptual model for hypermedia documents that supports composition is illustrated in Figure 1.

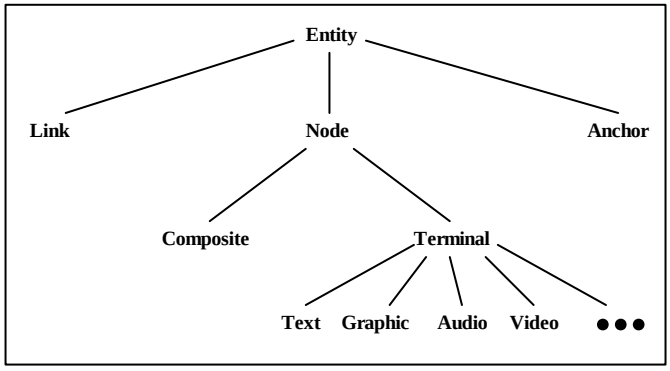

Figure 1: Class Hierarchy For a Basic Conceptual Model

An *entity* is an object that has as attributes[a] at least a *unique identifier*, an *access control list* and an *entity descriptor*. The unique identifier has the usual meaning. For each entity attribute, the access control list has an entry that associates a user[b] or user group to his access rights for the attribute. The entity descriptor contains information determining how the entity should be presented to the user, as in the presentation specification of the Dexter model.[13]

A *node* is an entity that has as additional attributes an *anchor list* and a *content*. The former concept is addressed in detail in Section 2.2 and the exact definition of node content depends on the class of the node.

A *terminal node* is a node whose content and anchor list are both application dependent. The model allows the class of terminal nodes to be specialized into other classes (*text*, *audio*, *image*, etc.).

A *composite node C* is a node whose content is a set *L* of nodes and links (sections 2.2 and 2.3 describe restrictions on composite nodes related to anchors and links). We say that an entity *E* in *L* is a *component of C* and that *E is contained in C*. We also say that an entity *A* is *recursively contained in B* iff *A* is contained in *B* or *A* is contained in an entity recursively contained in *B*. The class of composite

[a] We will frequently use the name of the attribute of an entity to refer to the attribute value. When the context does not allow this simplification, we will explicitly use the term *attribute value*.

[b] The word *user* in the context of this paper has multiple meanings: it means a user in the sense of a person, an application process or an application programmer. That is, anything or anybody that makes use of the services defined at the several interface layers of HyperProp.

nodes may be specialized into other classes.

The content of *C* could have been defined as an ordered list, as in [1], so that an entity might be included more than once in *L*. This alternative would be useful when defining sophisticated navigation mechanisms, as in trails,[1] but it requires more elaborated definitions for the concepts introduced in the following sections.

Composite nodes permit organizing, hierarchically or not, sets of entities, that is, sets of nodes and links. Therefore, they help structuring hyperdocuments, lessening the so-called "lost in hyperspace" problem.[7,14] For example, consider a working document of a Drama Research Team about English Poetry of the XVI Century. The document can be modeled as a composite node *E* containing a composite node *S*, grouping plays by Shakespeare, and another composition node *C*, grouping sonnets by Christopher Marlowe. Node *S* may contain, say, text nodes *H* and *M* representing the plays "Hamlet" and "Macbeth" and *C* may contain a text node *F* representing "Dr. Faustus". This example will be extended in Section 2.3 to include links.

The class of composite nodes may be specialized into other classes, for example, to organize a group of versions of the same node.[9,11,12] An application may then use these composite nodes to keep the version history of a hyperdocument, as well as to maintain automatic reference to the last version, two serious problems related to object versioning.[9,11,1] This topic will be expanded in Section 3.2.

Finally, composite nodes can be used to model the user's interaction with a hyperdocument, according to the work session paradigm proposed by the Dexter Model.[13] We refer the reader to [12,11] for an in depth treatment of the use of composition nodes to support cooperative work.

2.2 - Anchors

Recall that a node is an entity that has among its attributes a content and an anchor list. Each element of the anchor list is called an *anchor* of the node and is an entity that has as additional attribute a *region*. The anchors of a node are ordered in a list to facilitate the treatment of node versioning, as discussed in Section 3.2. Note that an anchor can be identified by: (1) its unique identifier (as any entity); or (2) by the unique identifier of the node where it occurs and its position within the list of anchors of that node. Note also that an anchor may belong to more than one anchor

list. The exact definition of anchor region depends on the class of the node, as discussed below.

We define a region of a terminal node as either the special symbol λ, representing the entire node content, or as a set of *marked information units*. The concept of marked information units is similar to the marked states of the Firefly system.[15] The exact notion of information unit and marked information unit is part of the definition of the terminal node. For example, an information unit of a video node could be a frame, while an information unit of a text node could be a character. Any subset of the information units of a terminal node may be marked.

The anchor list acts as the external interface of a terminal node *N*, in the sense that an entity can access regions of the content of *N* only through the anchor list of *N*. Hence, anchors shield other entities from changes to the content of *N*. As an example, consider a text node with just one anchor whose region points to the second paragraph of the text. Assume that a link indirectly refers to the second paragraph of the text by identifying the appropriate anchor. Then, any changes to the text must be reflected only in the anchor region and do not affect the link.

For a composite node *C*, an anchor region must be either the special symbol λ or a subset of *L* containing only nodes (λ again represents the entire node content). The anchor list of *C* has about the same role as that of a terminal node, that is, to act as an external interface to the node, except that it does not entirely shield entities from changes to the content of *C*. In particular, links are potentially sensible to changes to the content of *C* (this remark will be clarified immediately after the definition of links in Section 2.3).

Finally, we stress that, unlike the conceptual model of WWW, we defined the anchor list as an attribute of a node, outside the content of the node. Indeed, anchors can be defined: (1) as part of the content of a node; (2) as a separate attribute of a node; (3) as an attribute of another object, such as a link; (4) as an entirely independent object. In the last three cases, illustrated by HyperProp,[12] Microcosm,[16,9] the user can create anchors even for nodes that he cannot alter, such as a node stored in a CD ROM.

2.3 - Links

A *link* is an entity with three additional attributes, the *source end point set*, the *destination* or *target end point set* and the *meeting point*. The values of the two first

attributes are sets whose elements, called *end points* of the link, are pairs of the form $<(N_k,...,N_1),A>$ such that N_1 is a node, N_{i+1} is a composite node and N_i is contained in N_{i+1}, for all $i \in [1,k)$, with k>0, and A is an anchor of N_1. The node N_k is called a *base node* of the link.

We also require that, for every link l contained in a composite node C, for every end point $<(N_k,...,N_1),A>$ of l, node N_k must be either C or a node contained in C.

If the link has a single source end point $<(S_m,...,S_1),A>$ and a single target end point $<(T_n,...,T_1),B>$, we will denote the link as the pair $(<(S_m,...,S_1),A>,<(T_n,...,T_1),B>)$ in what follows.

The *meeting point* attribute defines *conditions* that must be satisfied at the source end points in order that *actions* be applied at the target end points. Examples of conditions are: the selection of an anchor by a user, the presentation of a region of an anchor, etc. Examples of actions are: run a presentation of a node, modify a presentation of a node, etc. Conditions and actions in a link are in conformance with MHEG[17] and are discussed in details in [18] and [19].

Links define relationships among nodes, such as *go to* relationships (for reference) and synchronization relationships (such as those defined in the MHEG proposal). Multiple source and destination end points allow the definition of many-to-many relationships, and are intended to support applications where, for example, the selection of a link can lead to the simultaneous exhibition of several nodes. Defining end points of a link l as pairs of the form $<(N_k,...,N_1),A>$ considerably simplifies modeling documents in some cases. However, if the content of N_{i+1} is changed by deleting N_i, link l will have to be changed. Therefore, the anchor list of a composite node does not entirely shield other entities from changes on the content of the node, as is the case above involving N_{i+1} and l. However, if the link has an end point of the form $<(N_k),A>$, then the anchor A in this case shields changes to N_k.

The list of nodes in each end point allows the definition of links connecting information not directly contained in the same composite node. As an example, consider again a composite node E containing a composite node S, grouping plays by Shakespeare, and another composition node C, grouping sonnets by Christopher Marlowe. Also recall that node S contains text nodes H and M representing the plays “Hamlet” and “Macbeth” and C contains a text node F representing “Dr. Faustus”. It may well be that the Drama Research Team 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). This link may be defined in *E* as *(<(S,H),r>,<(C,F),s)>),* where the anchors *r* and *s* allow the connection to be made more precise, for instance, by pointing to the sentences where the common concept first appears.

As another example, since *S* contains *H* and *M*, a link connecting these nodes may, in principle, be defined in *S* as l_S*=(<(H),i>,<(M),j>),* where *i* and *j* are valid anchors for *H* and *M*, as shown in Figure 2.a. If one wants to create a link in *E* connecting *H* and *M*, one may define the link as l_E*=(<(S,H),i>,<(S,M),j)>),* as shown in Figure 2.b. Note the difference between these two alternatives. Link l_S will be seen by every document that includes *S* (the composite node grouping plays by Shakespeare will probably be shared by several documents), while link l_E will be seen in *S* only by users that access *S* through *E* (see the discussion about perspective in Section 2.4).

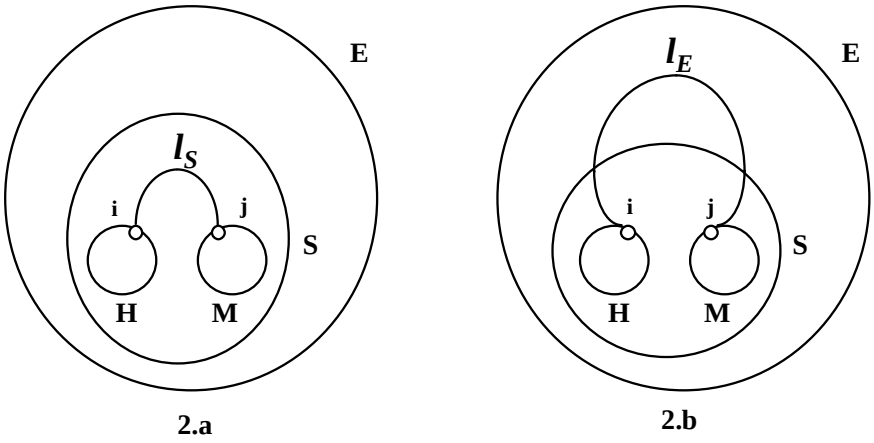

Figure 2: Links in composite nodes.

The relationship between *H* and *M* can also be captured by a set of three links, l_1 , l_2 and l_3 , as shown in Figure 3. In this case, link l_1 *=(<(S),m>,<(S),n>)* will be defined in *E* and links l_2 *=(<(H),i>,<(S),m>)* and l_3 *=(<(S),n>,<(M),j>)* will be defined in *S*. Note that, in this case, the anchors indeed shield changes to the composite nodes, as opposed to the example in figure 2.b.

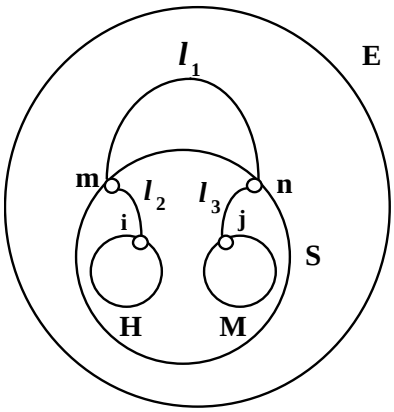

Figure 3: Other way of define links in composite nodes.

These alternatives for capturing the relationship between *H* and *M* open different possibilities to make the presentation of these nodes dependent on how the nodes are reached. For example, in both cases of Figure 2, assume that the relationship defined by the link is "go to the end point and show it". Then, after presenting node *H,* if the user navigates through the link, node *M* will be presented. The user will not known that *M* was contained in *S.* On the other hand, in Figure 3, the relationship defined by l_3 and l_2 could be "go to the end point and show it", whereas the relationship defined by l_1 could be "go to end point after one minute and close its exhibition". The user would know that *M* was contained in *S* when links l_2, l_1 and l_3 are followed.

2.4 - Perspectives and Visible Links

Recall that we allow different composite nodes to contain the same node and composite nodes to be nested to any depth. We thus introduce the concept of perspective to identify through which sequence of nested composite nodes a given node is being observed and the notion of visible link to determine which links actually touch the node in a given perspective.

A *perspective* of a node *N* is a sequence $\boldsymbol{P} = (N_m,...,N_1)$, with $m \geq 1$, such that $N_1=N$, N_m is a node not contained in any composite node, N_{i+1} is a composite node and N_i is contained in N_{i+1}, for $i \in [1,m)$. Since *N* is implicitly given by $\boldsymbol{P}$, we will refer to $\boldsymbol{P}$ simply as a perspective. Note that there can be several different

perspectives for the same node *N*, if this node is contained in more than one composite node. The *current perspective* of a node in a given user session is that traversed by the last navigation to that node in the session.

Given a node N_1 and a perspective $\boldsymbol{P} = (N_m,...,N_1)$, we say that a link *l* is *activated for* N_1 *by* $\boldsymbol{P}$ if and only if there is $r \in [1,m]$ such that *l* is contained in N_r and *l* has an end point either of the form $<(N_r,...,N_1,E_n,...,E_1),A>$ or of the form $<(N_{r-1},...,N_1,E_n,...,E_1),A>$, where in both cases the suffix $E_n,...,E_1$ may be omitted. Moreover, if the suffix is indeed omitted, then we say that the link is *fully activated for* N_1 with anchor *A visible in* N_1. Otherwise we say that the link is *partially activated for* N_1 with anchor λ *visible in* N_1.

Note that the requirement on the end point is consistent with the general definition of end point. Intuitively, link *l* must have an end point that contains N_1 and which follows the list of nested composite nodes down to N_1.

Note also that the list of nodes of an end point of a link is never a perspective, because a link always pertains to a composition. However, the concatenation of the sequence of nodes, that defines the perspective of the composite node *C* that contains the link *l*, and the list of nodes of the end point of *l* is always a perspective, except when *C* pertains to the list of nodes of the end point, when it must be computed only one time in the concatenation to form the perspective.

A *multi-perspective* of a list of nodes $N_1,...,N_k$ is a list $\boldsymbol{p}=(\boldsymbol{P}_1,...,\boldsymbol{P}_k)$ such that $\boldsymbol{P}_i$ is a perspective for N_i, for each $i \in [1,k]$.

Given a list of nodes $N_1,...,N_k$ and a multi-perspective $\boldsymbol{p}=(\boldsymbol{P}_1,...,\boldsymbol{P}_k)$ for the nodes, we say that a link *l* is *visible from* $\boldsymbol{p}$ *by* $N_1,...,N_k$ if and only, for some $r \in [1,k]$, link *l* is activated for N_r by $\boldsymbol{P}_r$.

For example, considering Figure 4 assume that: (i) node *A* contains nodes *B* and *Z*; (ii) node B contains nodes *C* and *D*; (iii) node *Z* contains *C* and *E;* and (iv) the composite nodes contain the following links:

	Node	***Nodes contained in***	***Links contained in***
(1)	*A*	*B* and *Z*	*(<(B,C),i>,<(Z,E),j>)*
(2)			*(<(B),m>,<(Z),n>)*
(3)	*B*	*C* and *D*	*(<(C),r>,<(D),k>)*
(4)	*Z*	*C* and *E*	*(<(E),s>,<(C),t>)*

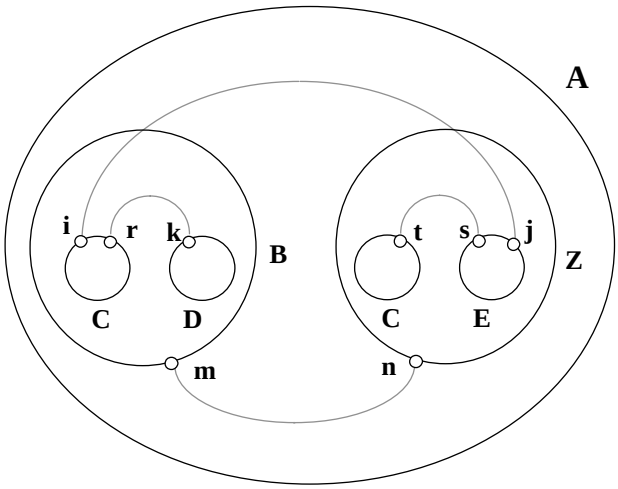

Figure 4: Examples of links in composite nodes.

Then, the following are examples of multi-perspectives and visible links:

	List of Nodes	***Multi-Perspective***	***Links Visible***	***Explanation***
(5)	*C*	*((A,B,C))*	*(<(B,C),i>,<(Z,E),j>)*	from (1)
(6)			*(<(C),r>,<(D),k>)*	from (3)
(7)	*C*	*((A,Z,C))*	*(<(E),s>,<(C),t>)*	from (4)
(8)	*B*	*((A,B))*	*(<(B,C),i>,<(Z,E),j>)*	from (1)
(9)			*(<(B),m>,<(Z),n>*	from (2)
(10)	*C,C*	*((A,B,C),(A,Z,C))*	*(<(B,C),i>,<(Z,E),j>)*	from (1)
			(<(C),r>,<(D),k>)	from (3)
			(<(E),s>,<(C),t>)	from (4)

Note that link *(<(B,C),i>,<(Z,E),j>)* is not visible by *C* from perspective *(A,Z,C)* since *(B,C)* is not a suffix of the perspective.

Intuitively speaking, the current perspective of a node determines which anchors the user will see when the node is presented and to which node the user will navigate to when, for instance, he selects one of the visible anchors. For example, if node *C* is presented through perspective *((A,B,C))* then, by lines (5) and (6), the anchors *i* and *r* will be visible in *C*. If he selects anchor *i* then he will

navigate to anchor *j* of *E*, by line (5), and if he selects anchor *r* then he will navigate to anchor *k* of *D*, by line (6). Likewise, by lines (8) and (9), if node *B* is exhibited through the perspective *((A,B))* then the anchors visible in *B* will be the special symbol λ, indicating the whole node *B*, and *m*, from which the user may navigate to anchor *j* of *E* and to anchor *n* of *Z*, respectively.

3 - Extending the Concepts

3.1 - Virtual Anchors and Links

In general, a *virtual entity E* is an entity such that the value of at least one attribute *A* is an expression ***e***, written in a formally defined hypermedia query language, whose evaluation results in an object of the appropriate type. The attribute *A* is also called *virtual*. When some operation requests the value of *A*, the object resulting from evaluating ***e*** is then returned. The importance of virtual entities to versioning is discussed in Section 3.2.

In particular, a *virtual anchor* in the anchor list of a node *N* is an anchor whose region is defined by an expression ***r***, which will be computed when traversing a link that has an end point touching the anchor. The expression ***r*** must in this case evaluate to a segment (marked region) inside the content of *N* or to the special symbol λ, if *N* is a terminal node, or to a subset of the nodes contained in *N* or to the special symbol λ, if *N* is a composite node.

A *virtual link* is in turn a link *l* that has at least one end point defined by an expression ***e***, which will be computed when selected. If *l* is contained in a composite node C, then ***e*** must return a base node that is either *C* or a node contained in *C*, as generally required in Section 2.3. Virtual links provide a powerful authoring tool, similar to that found in Microcosm[16] and may reduce the authoring effort.

We may go one step further and define a *generic link l* for nodes of classes $\boldsymbol{T}_1,...,\boldsymbol{T}_m$ as a link whose source end point set contains expressions $\boldsymbol{e}_1,...,\boldsymbol{e}_m$ such that $\boldsymbol{e}_i$ returns objects that qualify as segments of the content of nodes in class $\boldsymbol{T}_i$, for $i \in [1,m]$. We consider that *l* connects any tuple of nodes $M_1,...,M_m$ in classes $\boldsymbol{T}_1,...,\boldsymbol{T}_m$ that have anchors whose regions $r_1,...,r_m$ define segments that match expressions $\boldsymbol{e}_1,...,\boldsymbol{e}_m$, respectively.

Intuitively, a link is generic when it is not applicable to only one document, but it may be used for a whole class of documents. For example, one may define a link that has as target end point a video node showing an aerial sight of the Amazon rain forest and as source end point the expression ***Amazon rain forest*.** Then, any text node with an anchor whose region contains the sentence *Amazon rain forest* would automatically be linked to the video node.

Let *N* be a node representing a given document. It may not be reasonable to attempt to match every generic link with *N*, creating the appropriate anchors, if the number of links is very large. Also, it may be unfeasible, if *N* is large, to attempt to match every possible region in *N* against the collection of generic links. In the design of HyperProp, we opted for a solution to this problem that allows the user to mark an area of the document and to select from a menu an operation that tries to match all possible regions in the selected area against the collection of generic links and displays any matches found. Then, for each region found, if the user desires, he may create an anchor *A* for that region and a new link having the same definition as the matched generic link, except that it has *A* as the source end point anchor. This link would belong to a composite node containing both of its base nodes, as required by the model.

3.2 - Anchors and Links in Object Versioning

To address the problem of maintaining the history of a document, we extend the conceptual model introduced in Section 2 with the class of version groups.

A *version group V* is a composite node such that all nodes contained in *V* have anchor lists with the same length (the reason for this restriction is explained after we redefine link end points below). Intuitively, *V* groups nodes that represent versions of the same entity, at some level of abstraction, without necessarily implying that one version was derived from the other. The nodes in *V* are called *correlated versions*, and they need not belong to the same node class. Note that a version group may contain composite nodes, which provides us with an explicit versioning of the document structure.

The derivation relationship is explicitly captured by the links in *V*. We say that v_2 *was derived from* v_1, if there is a link of the form $(<(v_1),i_1>, <(v_2),i_2>)$ in *V*. The anchors in this case simply let one be more precise about which part of v_1 generates which part of v_2. There is no restriction on the derivation relationship,

except that it must be acyclic.

A user may either manually add nodes (to explicitly indicate that they are versions of the same object) and links (to explicitly indicate how the versions were derived) to a version group, or he may create a new node from another by invoking a versioning operation, which will then automatically update the appropriate version group.

An application has several options to define the node of a version group *V* the application considers to be the *current version* of *V*, according to a specific criteria. For example, the application may reserve an anchor *A* of *V* to maintain a reference to the current version (of the application). In fact, if virtual anchors are supported as defined in Section 3.1, the application may define the region of *A* as an expression ***e*** that always evaluates to the current version. The expression ***e*** may evaluate to several versions (for example, "all versions created by John"), which can be interpreted as alternatives and returns a composite node. As another example, the expression ***e*** may be defined in a link or even in a more general way, as is the case of some models,[12,9] where the current version is defined in an attribute of the composite node, for all its component nodes.

In Section 2, we defined an end point of a link contained in a composition node *C* as a pair of the form $<(N_k,...,N_1),A>$ such that N_k must be either *C* or a node contained in *C*, N_{i+1} is a composite node and N_i is contained in N_{i+1}, for all $i \in [1,k)$, with k>0, and *A* is an anchor of N_1. However, to incorporate the notion of version in a version group, we must redefine an end point of a link contained in a composite node *C* as a triple $<(N_k,...,N_1),A,q>$ such that:

- For all $i \in [1,k)$, N_{i+1} is a composite node, N_i is contained in N_{i+1} and N_k is either *C* or a node contained in *C*.
- If N_1 is a version group, then *A* is an anchor of N_1 and $q \in [1,n)$, where *n* is the common length of the anchor lists of nodes in N_1. That is, *q* identifies an anchor of each node in N_1, in special in the node(s) specified by *A*. We say, in this case, that N_2 does not contain N_1, but that it contains the node(s) specified by the anchor *A*.
- If N_1 is a terminal node or a composite node (which is not a version group), then *A* is an anchor of N_1 and *q* is the null value.

This definition is consistent because we required that all versions in a version group must have anchor lists with the same length and that the anchors of a node must be ordered. That is, one may refer to the ith anchor of a version in a version

group without actually specifying the version.

4 - Conclusions

We addressed in this paper primarily the problems one must face when defining anchors and links in the presence of nested composite nodes. We also touched the additional problems raised by the introduction of virtual anchors and links and by the presence of object versioning.

Among other points, we equated the problem of hiding changes to the content of a node by treating anchors as node interfaces. We explored the nesting of composite nodes by introducing a very general format for links, allowing end points to be sequences of nested composite nodes ending in an anchor. We then captured link inheritance through the notion of perspective. By resorting to virtual anchor and links, we pointed out how to solve the problem of maintaining automatic reference to the last version of a node.

The discussion can be extended further in many directions, such as specifying a query hypermedia language and by exploring conditions and actions added to anchors and links, topic addressed in [18] and [19].

Acknowledgments

The work reported in this paper was partially financed by a grant from the HyperProp-PROTEM project of CNPq-Brazil.

References

1. *Soares, L.F.G.; Casanova, M.A.; Rodriguez, N.L.R. "Nested Composite Nodes and Version Control in an Open Hypermedia System"*. International Journal on Information Systems; Special issue on Multimedia Information Systems. *September 1995.*
2. *Soares, L.F.G.; Casanova, M.A.; Colcher, S. "An Architecture for Hypermedia Systems Using MHEG Standard Objects Interchange"*. Information Services & Use*, vol.13, no.2. IOS Press. Amsterdam, The Netherlands. 1993; pp. 131-139.*

3. *Delisle, N.; Schwartz, M. "Neptune: A Hypertext System for CAD Applications". Proceedings of ACM SIGMOD '85. Washington, D.C. May 1985.*

4. *Delisle, N.; Schwartz, M. "Context - A Partitioning Concept for Hypertext". Proceedings of Computer Supporteed Cooperative Work. December 1987.*

5. *Goldstein, I.; Bobrow, D. "A Layered Approach to Software Design". Interactive Programming Environments. McGraw Hill, pag. 387-413. Nova York. 1987.*

6. *Meyrowitz, N. "Intermedia: The Architecture and Construction of an Object-Oriented Hypermedia System and Applications Framework". Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications. Portland, Oregon. September 1985.*

7. *Halasz, F.G. "Reflexions on Notecards: Seven Issues for the Next Generation of Hypermedia Systems". Communications of ACM, Vol.31, No. 7. July 1988.*

8. *Aksscyn, R.M.; McCracken, D.L.; Yoder, E.A. "KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations". Communications of ACM, Vol.31, No. 7. June 1988.*

9. *Osterbye, K. "Structural and Cognitive Problems in Providing Version Control for Hypertext". Proceedings of European Conference on Hypertext, ECHT'92. Milano. December 1992.*

10. *Schütt, H.A.; Streitz, N.A. "HyperBase: A Hypermedia Engine Based on a Relational Database Management System". Procceedings of European Conference on Hypertex, ECHT'90. 1990.*

11. *Haake, A. "Cover: A Contextual Version Server for Hypertext Applications". Proceedings of European Conference on Hypertext, ECHT'92. Milano. December 1992.*

12. *Soares, L.F.G.; Casanova, M.A.; Rodriguez, N.L.R. "Nested Composite Nodes and Version Control in Hypermedia Systems". Proceedings of the Worshop on Versioning in Hypertext Systems, in connection with ACM European Conference on Hypermedia Technology, Edinburgh. Setembro de 1994.*

13. *Halasz, F.G.; Schwartz, M. "The Dexter Hypertext Reference Model". NIST Hypertext Standardization Workshop. Gaithersburg. January 1990.*

14. *Halasz, F.G. "Seven Issues Revisited". Final Keynote Talk at the 3rd ACM Conference on Hypertext. San Antonio, Texas. December 1991.*

15. *Buchanan, M.C.; Zellweger, P.T. "Specifying Temporal Behavior in Hypermedia Documents". Proceedings of European Conference on Hypertext, ECHT'92. Milano. December 1992.*

16. *Fountain, A.; Hall, W.; Heath, I.; Davis, H. "Microcosm: An Open Model for Hypermedia wiyh Dynamic Linking". Proceedings of European Conference on Hypertext, ECHT'90. Cambridge University Press. 1990, pp. 298 -311.*

17. *MHEG. "Information Technology - Coding of Multimedia and Hypermedia Information - Part1: MHEG Object Representation / Base Notation (ASN.1)". ISO/IEC DIS 13522-1. September. 1995.*

18. *Souza, G.L.; Soares, L.F.G. "Synchronization Aspects of a Presentation Model for Hypermedia Documents with Composite Nodes". Research Report PUC-RioInf.MCC31/95. Rio de Janeiro, Brasil. Outubro 1995.*

19. *Souza; G. L.; Soares, L.F.G.; Casanova, M.A. "Synchronization Aspects of an Hypermedia Presentation Model with Composite Nodes". ACM Workshop on Effective Abstractions in Multimedia, in connection with the ACM Multimedia'95, San Francisco, EUA. November 1995.*