The Nested Context Model for Hyperdocuments

Marco A. Casanova, Luiz Tucherman

Centro Cientifico Rio
IBM Brasil
P.O. Box 4624
20.001, Rio de Janeiro, RJ - Brasil
casanova@riovmsc.vnet.ibm.com

Maria Julia D. Lima, Jose L. Rangel Netto, Noemi Rodriguez, Luiz F.G. Soares

Departamento de Informatica
Pontificia Universidade Catolica do RJ
R. Marques de S. Vicente, 225
22.453, Rio de Janeiro, RJ - Brasil

ABSTRACT

This paper describes the nested context model, a conceptual framework for the definition, presentation and browsing of documents. The model carefully combines hypertext links with the concept of context nodes, used to group together sets of nodes. Context nodes can be nested to any depth and, thus, generalize the classical hierarchical organization of documents. The nested context model also defines an abstract and flexible application program interface that captures the idea that different applications may observe the same node in different ways. Finally, the model offers a rich set of operations to explore the double structure of a hyperdocument - that defined by the links and that induced by the nesting of context nodes.

1. INTRODUCTION

We describe in this paper a conceptual framework, called the *nested context model*, for the definition, presentation and browsing of documents. The *definition submodel* introduces the basic concepts used to structure documents, which are that of *node* and *link*, as in the familiar hypertext systems [4,8]. The model carefully combines hypertext links with a special type of node, called *context nodes*, used to group together sets of nodes, including context nodes. Context nodes can be nested to any depth and, thus, generalize the classical hierarchical organization of documents [4].

The *presentation* and *navigation submodels* define an abstract and flexible application program interface. The presentation submodel introduces the concept of the *presentation* of a document node, that in some sense captures the idea that different applications may observe the same node in different ways. It also defines the concept of *state*, that roughly corresponds to a model for a set of partially retrieved documents.

Finally, the navigation submodel, as the name implies, defines the navigation primitives and the higher-order navigation operations. It offers a rich set of operations to explore the

double structure of a hyperdocument - that defined by the links and that induced by the nesting of context nodes.

The concept of context node generalizes the homonym concept introduced in the Neptune system [5], which was in turn based on some ideas from PIE [7]. However, contexts in Neptune cannot be nested. It also generalizes Intermedia's webs [11] and Notecard's fileboxes [9]. Indeed, it is quite easy to implement these concepts using the context nodes we propose here. Moreover, context nodes have their own semantics defined within the model (in the same line as [6]).

This paper is organized as follows. Sections 2 to 4 respectively introduce the definition, presentation and navigation submodels. Finally, section 5 contains the conclusions.

2. THE DEFINITION SUBMODEL

The definition of hyperdocuments in the nested concept model is based on two familiar concepts, node and links. *Nodes* are fragments of information and *links* interconnect nodes into networks of related nodes.

The model goes further and distinguishes two basic classes of nodes, called terminal and context nodes, the latter being the central concept of the model. Intuitively, a *terminal node* contains data whose internal structure, if any, is application dependent and will not be part of the model. The class of terminal nodes may be specialized into other classes (*text*, *voice*, *image*, etc.) as required by the applications. The class of the node determines its *attributes*[1], that contain user-defined information or implementation dependent information. Independently of the class, every node N must have an attribute *contents*, that describes the data associated with the node, and an attribute Id that assigns a unique *node identifier* to the node. The values of these attributes will be referred to simply as the *contents* and the *id* of the node.

A *context node* groups together sets of terminal or context nodes, recursively. The concept of context node therefore permits organizing, hierarchically or not, sets of nodes and offers a mechanism to define different views of the same document, tuned to different applications or classes of users.

The reader is referred to section 4.1 for examples of the concepts introduced by the following definitions.

More precisely, if N is a context node, then its contents must define a pair (**N**,**L**), where **N** is a set of nodes and **L** is a set of links (see below for this concept) whose end nodes belong to **N**. We say that N contains a node M iff M is in **N** and that N contains a link l iff l is in **L**. It should be clear that a node M may be contained by more than one context node, but every change to any of the attribute values of M will become visible through all context nodes that contain M. However, a link is internal to a context node, that is, links are not shared by context nodes. We also say that N recursively contains a node M iff N contains M or N contains a context node that recursively contains M.

A context node *defines a hyperdocument* in our model and a *hyperbase* is a set of nodes **H** such that, for any context node N in **H**, if N contains a node M, then M is in **H**. The hyperbase is *consistent* iff no two nodes have the same *id*. The process of adding a hyperdocument to a hyperbase **H** then consists of adding a context node H to **H** and, recursively, adding to **H** all nodes recursively contained in H that are not already present in **H**.

---

[1] Attribute and operation names will be overloaded, that is, two classes may have the same attribute name or operation name.

A link basically connects two nodes. But, since the contents of a node has an internal structure, that may be quite complex, a link also indicates, for each end node, a region where the link is "anchored". For example, for a text node, the region can simply be a string of characters within the text, for a 2-D image node, it can be a rectangle determined by two pairs of coordinates.

More precisely, an *anchor* is a pair A=(N,s) such that N is a node and s is either:

- the *null offset Null*; or

- a *displacement* within the contents of N, if N is a terminal node, where the exact notion of displacement depends on the class of N; or

- an anchor (M,r) such that N contains M, if N is a context node.

We call s the *offset* and N the *base* of A. We also say that a node M *is present* in an anchor (N,s) iff M=N or N is a context node and M is present in s.

A *link* is a pair (S,D) of anchors, where S is the *source anchor* and D is the *destination anchor* of the link. The *end nodes* of a link are the bases of its anchors. Thus, by defining anchors recursively, an application may possibly traverse a link in various stages, each one covering the next node in the offset anchor, until reaching a terminal node or a null offset.

As previously mentioned, a link belongs to the context node that contains its end nodes. For example, let A be a context node that contains another context node B that in turn contains two nodes, C and D. A link connecting C and D may, in principle, be defined in B as ((C,*Null*),(D,*Null*)) since B contains its end nodes, C and D. This means that any other context node A' that also contains B will also implicitly contain the link. If one wants this situation, he must redefine the link in A as ((B,(C,*Null*)),(B,(D,*Null*))).

The nested context model allows different context nodes to contain the same node and context nodes to be nested to any depth. Thus, we need a way of identifying through which sequence of nested context nodes a given node is being observed and which links actually touch the node from that nesting. This is captured by the notion of perspective of a node and the notion of links visible by a node from a perspective.

A *perspective* for a node N is a sequence P=(N_1,...,N_m), with m>=1, such that N_1=N and N_i is contained in N_{i+1}, for i in [1,m]. Since N is implicitly given by P, we will refer to P simply as a perspective. We say that a node M *is present* in P=(N_1,...,N_m) iff M=N_i, for some i in [1,m].

Finally, we say that a link l *is visible* by N from a perspective P for N iff there is a context node M such that M is present in P, M contains l and N is present in one of the anchors of l. We denote the set of links visible by N from P by V(P).

3. THE PRESENTATION SUBMODEL

We discuss in this section how to present a hyperdocument, which is based on the notion of representation of a node and the notion of state, covered in two separated subsections.

3.1 THE NOTION OF REPRESENTATION

The *representation* of a node N reflects the values of the attributes N according to the needs of a given application. For example, a voice node may have its contents edited using a waveform representation or may have its contents output through an audio

interface using a second representation. The act of *navigation* is then the way the application traverses the nodes of a hyperdocument, creating new representations that permit exhibiting or manipulating in any other way the attribute values of these nodes.

A *representation class* **RC** for a node class **NC** in general defines the attributes of the representations in **RC** and the operations allowed on these representations. The minimal requirements the core model imposes on **RC** are the following. First, **RC** must have an operation *Conv* that maps **NC** into **RC** and an attribute **Id**, such that for any node N in **NC**, if $R=Conv(N)$ then $Id(R)=Id(N)$. Second, R must have a second attribute, *RId*, that acts as the internal identifier of the representations. Finally, if **NC** is a class of terminal nodes, **RC** must have an operation, *ConvDispl*, that converts displacements that may occur in links. This is necessary because the value of displacements depend on **NC**. The inverses of *Conv* and *ConvDispl* are not required in this paper since we will not consider updates on representations.

The core model does, however, include a *canonical representation* class **CC** for context nodes that defines two attributes, *NodeSet* and *LinkSet*, in addition to *Id* and *RId*. For class **CC**, the mapping *Conv* is such that, for any context node N, $Conv(N)=R$ iff

- $Id(R) = Id(N)$

- $NodeSet(R) = \{Id(M) / N \text{ contains } M\}$

- $LinkSet(R) = \{((i_1,(...(i_p,d)...)), (j_1,(...(j_q,e)...))) / N$ contains a link $((M_1,(...(M_p,d)...)),(N_1,(...(N_q,e)...))$ such that $i_k=Id(M_k)$, for each k in [1,p], and $j_k=Id(N_k)$, for each k in [1,q]$\}$

Thus, $NodeSet(R)$ and $LinkSet(R)$ represent the set of nodes and links contained in N simply by replacing nodes by their *ids*. The definition of *LinkSet* must be revised when more than one representation for each perspective of a node is allowed (see [3]).

The model permits defining default representation classes for the nodes in various ways:

- a node may directly indicate an associated default representation class;

- a link may have a default representation class for each node present in its anchors;

- a context node may define a default representation class for the nodes it contains;

- finally, a node class must be associated with a default representation class.

When the application selects a node, it may specify which representation class to use. If the application does not specify any representation class, the navigation operations first try to use that specified with the node, if any, and then the default representation class associated with the node class. The other forms of indicating default representation classes exist just to offer a basic selection from which the application may choose from. This is further discussed in section 4.2.

3.2 THE NOTION OF STATE

The navigation primitives produce a set of representations with a certain structure that we capture with the help of the notions of state and consistent state. The definitions that follow are relative to a given hyperbase **H**.

A state is a pair s=(G,P) where G is an acyclic digraph whose nodes are representations and P is a path from a node to a sink in G. We say that P is the *current perspective* of s and that the first element of P is the *current representation* of s.

If (R,S) is an edge of G, we say that S is a *parent* of R and that R is a *child* of S. A node S *dominates* a node R in G iff there is a path from R to S in G. By carefully indicating when the context is that of a graph or that of a hyperdocument, the double use of the term "node" should cause no confusion.

A state s=(G,P) is *consistent* iff

C1. no two representations which are nodes of G have the same internal *id*;

C2. for every edge (R,S) in G, S must be a representation of a context node that contains the node that R represents;

C3. for every pair of edges (R,S) and (R',S') in G, if $S=S'$ then R and R' must not be representations of the same node.

Let s=(G,P) be a consistent state in what follows. Condition C2 implies that each path from a representation R_1 in G of a node N_1 to a representation R_m in G of a node N_m induces a perspective of N_1 ending on N_m. Indeed, let $(R_1,...,R_m)$ be a path from R_1 to R_m in G and define $P_1=(N_1,...,N_m)$, where N_j is the node that R_j represents, for j in $[1,m]$. By C2, N_j is contained in N_{j+1}, for j in $[1,m-1]$. Hence, P_1 is a perspective of N_1.

Condition C3, together with the fact that the collection of nodes contained in a context node is in fact a set, in turn implies that, once a representation R in G of a context node N_m is chosen, any perspective $P_1=(N_1,...,N_m)$ of a node N_1 ending on N_m is associated with at most one representation in G. This is easily proved by induction on m using C2 and C3. This property is justified since it simplifies the handling of links as the navigation operations construct new representations.

Finally, let s=(G,P) be a consistent state, and R and S be representations in G. Suppose that R represents a node N. We say that l is in *LinkSet*(S) is visible by R in s iff there is a path from R to S in G and *Id*(N) occurs in one of the anchors of l. Suppose that the current perspective is $P=(R_1,...,R_m)$. The set of links *currently visible* in s (or the set of links *visible* by the current representation R_1) is the set of links l in *LinkSet*(R_i) visible by R_1 in s, with i is in $(1,m]$. This set corresponds to the links visible by N_1, the node R_1 represents, from the perspective induced by P.

4. THE NAVIGATION SUBMODEL

The non-linear structure of hyperdocuments requires facilities to traverse the links, descent through the context nodes, or explore the structure in some other form. We then address in this section the navigation submodel, beginning with an informal example in section 4.1. Then, we define in section 4.2 the navigation primitives and discuss in section 4.3 more complex navigation operations.

4.1 AN EXAMPLE OF NAVIGATION

Consider a hyperbase H={A,B,C,D,E}, where A and B are context nodes and C, D and E are terminal nodes. Node A contains B and C and a link f=((B,(D,d)),(C,c)). Node B

contains D and E and a link g=((*D*,*Null*),(*E*,*e*)). Hence, d, c and e are displacements within D, C and E, respectively. The following sequence of operations is a valid navigation over **H**:

1. selection of A from the hyperbase **H** and creation of a representation R_a for A;

2. selection of B from the set of nodes contained in A and creation of a representation R_b for B;

3. selection of D from the set of nodes contained in B and creation of a representation R_d for D;

4. traversal of the link f from D to C, since f is visible from R_d, and creation of a representation R_c for C;

5. reversed traversal of the link f to B, using the representation R_b already created;

6. traversal of the offset of the link f to D, using the representation R_d already created;

7. traversal of the link g from D to E, since g is visible from R_d and creation of a representation R_e for E.

4.2 NAVIGATION PRIMITIVES

All navigation primitives implicitly receive a state as input and, in some cases, produce a state as output, also implicitly; when they fail, they return FALSE, otherwise they return TRUE. A brief description of the primitives follows:

AddRepr(NODE,REPRCLASS)->BOOL

This requires indicating the node (NODE) to be represented, and, optionally, the representation class (REPRCLASS) of R. If the class is not specified, it is taken as the default class associated with the node or with the node class, with this priority.

The current representation P is the parent of the new representation R. The new current perspective is the old one with R concatenated to the front. If P is not a representation of a context node that contains NODE, or if P already contains a child which is a representation of NODE, the operation fails because it will create an inconsistent state.

CreateRepr(NODE,REPRCLASS)->BOOL

Creates a new state exactly as *AddRepr*, except that the new representation R is not connected to any other representation and the new current perspective becomes the sequence (R). The operation always succeeds.

This operation exists to create a new representation for the first node of a hyperdocument brought into memory.

ConnectRepr(REPR)->BOOL

Creates a new state by adding the arc (REPR,C) to the current graph, where C is the current perspective. The new current perspective is the old one with REPR concatenated to the front. If C is not a representation of a context node that contains the node N that

REPR represents, or if *C* already contains a child which is a representation of *N*, the operation fails.

DeleteRepr(REPR)->BOOL

Creates a new state by deleting the representation (REPR). If REPR is connected to any other representation or occurs in the current perspective, the operation fails.

DisconnectRepr(REPR)->BOOL

Creates a new state by deleting the arc (REPR,*C*), where *C* is the current perspective. If no such arc exists, the operation fails.

MoveUp(REPR)->BOOL

Creates a new state that maintains the same representations as the current state, but whose current perspective is the old one with the prefix up to, but excluding REPR, dropped. That is, REPR becomes the current representation. If REPR does not occur in the current perspective, the operation fails.

MoveUpOne->BOOL

Creates a new state that maintains the same representations as the current state, but whose current perspective is the old one with the first element deleted. If the current representation has just one element, the operation fails.

MoveToChild(NODE)->BOOL

Creates a new state that maintains the same representations as the current state, but whose current perspective is the old one with *R* concatenated to the front, where *R* is the (only one) representation of NODE that is a child of the current representation. If *R* does not exist, the operation fails.

ObtainVisibleLinks->MARKEDVISIBLELINKSSET

Returns all pairs (*S*,*l*) such that *l* is in *LinkSet*(*S*) is currently visible, that is, visible by the current representation.

SavePerspective->RIDLIST

Returns the sequence RIDLIST of internal ids of the representations that form the current perspective.

ChangePerspective(RIDLIST)->BOOL

Creates a new state that maintains the same representations as the current state, but whose current perspective is the sequence of representations whose internal ids form the sequence RIDLIST.

4.3 HIGHER-LEVEL NAVIGATION MECHANISMS

The primitives describes in section 4.1 permit the definition of several higher-level navigation mechanisms. We exemplify in this section two mechanisms that are intrinsic to our model.

Ascending and Descending Navigation

The nested structure of context nodes permits two forms of navigation, that we call *descending* and *ascending*.

Ascending navigation consists of moving from a representation C of a node to a representation of the node that contains it, within the current perspective. Assuming that C is always the current representation, this can be achieved simply of executing *MoveOneUp*.

Descending navigation is to move from a representation C of a context node M to a representation of a node N that M contains. The operation described below assumes that C is always the current representation.

The first step of descending navigation is to obtain the set S of nodes contained in the context node represented by C. This is performed by calling an operation of the class to which C belongs, as discussed in section 3. After selecting a node N from S, there are three alternatives. The first alternative is to create new representation R for N, whose parent is C, by calling *AddRepr*(N,**f**), where **f** specifies a representation class for N, or is a null value. Then R becomes the current representation. The class **f** may be chosen among the default representation associated with N, in particular that defined in the context node that C represents. The second alternative is to set the current representation of the state to be the representation for N that is a child of C, if one exists. This requires executing the primitive *MoveToChild*(N). The third alternative is to set one of the existing representations R of N as a child of C. This is achieved by executing *ConnectRepr*(R), after selecting R, via some query mechanism, from the representations in the current state.

Link Traversal

Link traversal is the most typical navigation in hypertext systems and requires no preliminary comments. Using the navigation primitives, link traversal is performed as follows.

The first step is to obtain the set of links visible by the current representation C by invoking the primitive *ObtainVisibleLinks*. Recall that this operation actually returns pairs (F,f) such that f is in *LinkSet*(F), F is a context node in the current perspective and the *id* of the node C represents occurs in one of the anchors of f. The next step is to select one such pair (M,l) and obtain the destination anchor, (N,s) of l. By definition of link, N must be contained in the node M represents. Once this is completed, M is set as the current representation by calling *MoveUp*(M). The final step, to define a representation for N, is exactly as that described for descending navigation. If N is a terminal node, the displacement s is adjusted to the representation chosen by calling *ConvDispl* at this point.

We conclude by observing that these two operations just illustrate how to use the navigation primitives and by no means exhaust the list of navigation mechanisms.

5. CONCLUSIONS

We described in this paper the nested context model, which introduces a framework for the definition, presentation and browsing of hyperdocuments. The model features context nodes, that help overcome the disorientation problem in hypertext systems by introducing a structuring facility for hyperdocuments which is orthogonal to links. In another direction, the notions of representation and state offered an abstraction for the presentation of hyperdocuments to an application. Finally, the navigation primitives capture the essential actions for the gradual reading and browsing of hyperdocuments.

We are currently working on an extension of the model to cover node versions and other necessary facilities [3], and on a prototype implementation of a hypermedia system that supports the nested context model.

REFERENCES

[1] Ahuja, S.R., Ensor, J.R. and Horn, D.N., "The Rapport Multimedia Conferencing System", *ACM SIGOIS Bulletin* Vol.9, N.2/3 (April/June 1988), 1-8.

[2] Campbell, B. and Goodman, J.M., "HAM: A General Purpose Hypertext Abstract Machine", *Communications of the ACM*, Vol.31, No.7 (July 1988), 856-861.

[3] Casanova, M.A. et alli, "Version Control in the Nested Context Model for Hyperdocuments", Rio Scientific Center, IBM Brazil (technical report in preparation).

[4] Conklin, J., "Hypertext: An Introduction and Survey", *IEEE Computer*, Vol.20, N.9 (Sept. 1987), 17-41.

[5] Delisle, N. and Schwartz, M., "Neptune: A Hypertext System for CAD Applications", *Proc. ACM SIGMOD '86*, Washigton, D.C. (May 1986), 132-142.

[6] Feiner, S., "Seeing the Forest for the Trees: Hierarchical Display of Hypertext Structure", *Proc. of the Conf. on Information Systems, ACM*, New York (1988), 205-212.

[7] Goldstein, I. and Bobrow, D., "A Layered Approach to Software Design", in *Interactive Programming Environments*, D. Barstow, H. Shrobe and E. Sandewall (Eds.), McGraw-Hill, New York (1987), 387-413.

[8] Halasz, F.G., "Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems", *Communications of the ACM*, Vol.31, No.7 (July 1988), 836-852.

[9] Halasz, F.G., Moran, T.P. and Trigg, R.H., "Notecards in a Nutshell", *Proc. of the 1987 ACM Conf. of Human Factors in Computing*, Toronto, Ontario (April 1987), 45-52.

[10] Halasz, F.G. and Schwartz, M., "The Dexter Hypertext Reference Model", in *Proc. of the NIST Hypertext Standardization Workshop*, J. Moline, D. Benigni and J. Baronas (Eds.), (Feb. 1990), 95-133.

[11] Meyrowitz, N., "Intermedia: The architecture and construction of an object-oriented hypermedia system and applications framework", *Proc. of the Conf. on Object-Oriented Programming Systems, Languages and Applications*, Portland, Oregon (Sept. 1986), 186-201.

[12] Poggio, A. et allii "CCWS: A Computer-Based Multimedia Information System", *IEEE Computer*, Vol.18, N.10 (Oct. 1985), 92-103.

[13] Tompa, F.WM. "A Data Model for Flexible Hypertext Database Systems", *ACM Transactions on Information Systems*, Vol.7, N.1 (Jan. 1989), 85-100.