A Hypertext-Like Multimedia Document Data Model

Rubén Caudillo[1]
Michel Mainguenaud

FRANCE TELECOM - INT
9 rue Charles Fourier
91011 Evry France
email : CAUDILLO@FRINT51.bitnet
MAINGUENAUD@FRINT51.bitnet

ABSTRACT :

We present a data model and interface specifications for hypertext-like documents. The main goal is to offer browsing and authoring large general hypermedia documents. This proposition tries to take full advantage of electronic media. Hypertext links with system-defined semantics are considered as a way to represent both : the organization of the documents and structured database coupling. The data model defines two organization levels for the documents. The first level defines the construction of labelled graphs acting as a representation medium for small cognitive document units. The second level allows these graphs to act like elementary objects, which are organized in more complex structures using object-oriented abstraction techniques.
At the interface level, the data model components correspond to the real pages of a document. They are composed of nested blocks of multimedia contents. The model of the documents and the Document Definition and Manipulation Languages are proposed to be represented at the interface level using the same paradigms. They are based on the representation of structures, content and queries by nested blocks (e.g. rectangles) connected by arrows. The interface considers in an unified way, all the processing steps for document conception in order to offer a general tool dedicated for both : the readers and the authors of electronic documents.

[1] Permanent address : TELMEX - CENTRO DE INVESTIGACION Y DESARROLLO - Ejercito Nacional 579 piso 8 Col. Granada C.P. 11520 Mexico D.F.

A.C.M. Int. Conf on Multimedia Information Syst. 16.18 Janvier 91 Singapour

1. INTRODUCTION

Several fields have been traditionally studying electronic documents. Early, only single-media document modelling was possible and the data models were designed to emulate the style of the paper documentation. Technology evolved allowing electronic multimedia documents, high storage capacities and the management of large-sized document applications.

Some approaches from word processors, have continued to propose paper-like ways of structuring and formatting data. The document processing capabilities are based on multimedia, high storage capacities and fast searching abilities. However, these approaches have several drawbacks such as the difficulties to read the electronic documents on line. For instance, structuring is not adapted to this way of reading because of the small size of available screens. Typical applications are oriented to document presentation—like document authoring and publishing. Queries are limited to regular expression searching and several semantic aspects of documents are not treated in depth. Some recent standardization efforts seem promising for high quality printing and document transmission [9,12].

Other approaches from the database community propose a particular kind of documents oriented to business applications. For instance, the typical form-oriented data models are only adapted to strongly structured documents of small size. A document is a particular display representation of structured data possibly mixed with unformatted data (e.g., image and sound). The semantics of structured data, are efficiently treated on these approaches. Unformatted data are not merely treated because : they are considered like atomic data, and there are no primitives to manipulate them efficiently. Office automation is strongly based on these approaches. They are oriented to small documents reading and writing on line. The problem is that these models allow to consider only a small range of the rich variety of existing documents. The interesting aspects of some of the proposed data models are their capabilities of mixing the structuring and presentation aspects in a unified way. From database area, powerful abstraction mechanisms and languages have been proposed for structured data [4,15][3]. However, unstructured data coupling remains an open problem [14,16,19].

Some recent approaches from hypertext systems propose to relax the strongly typing and structuring modalities for documents—like a way to allow reading and writing documents on line [8]. The goal is to be able to take full advantage of the electronic supports [22]. That makes electronic documents more attractive than their paper versions and better adapted to all kind of readers (e.g., casual readers). Several interesting formal data models have recently been proposed [20,21]. The proposed features help to alleviate the device limitations and allow to properly read the documents on line (e.g., document customizing, free structuring, multimedia spatial linking and concurrent browsing). Some semantic aspects of unformatted data are implicitly treated by means of spatial-oriented annotations or links. Most data models perform well for small applications, but they have problems for representing both large-sized documents, and the existing relationships for a high number of small documents [11].

We present in this paper a data model, which is dedicated to large-sized documents, and which permits to mix the authoring and reading orientations. In order to take full advantage from output device technologies, we adopt a relatively free structuring style for documents. At the lower level of definition, document units are represented by a labelled graph with nodes representing pages and labels representing hypertext-like links. However to diminish user's bewilderment, the complexity of these graphs is limited. The labelled graphs should have a size small enough to be represented in a readable way at the screen. In order to provide more complex organizations, these graphs can be further organized using data abstraction mechanisms like aggregation or generalization.

For the document page representation, we adopt at the interface level a block metaphor where a page is considered like a set of nested rectangles or *blocks,* which may overlap each other. The blocks have several graphical properties (e.g., transparency, priority) and content types (e.g., graphics and sound). *Links* are blocks with specific semantics and with user-defined representation. Several semantic links are provided in order to permit both : linking unformatted data (e.g., usual document links) and the coupling of document contents to structured databases (e.g., by database query-like links).

In all blocks, dynamical aspects are also considered by the means of the *scripts*. These scripts are used to program the behavior of the blocks when they are manipulated at the interface level (e.g., opened and closed). Animation aspects, generic scripts and exception conditions may be homogeneously treated with this approach.

The interface proposes an environment where all the existing steps for document processing are defined using the same representation metaphor—a block and link world. Therefore editing is a process of filling up the document blocks and querying consists of filling conditions on empty blocks.

Lot of applications need the proposed features. For instance, geographical information systems, library automation, CAD/CAM, project management and videotex applications are characterized by large-sized documents, multiple media, graphical capabilities. They also need structured database coupling. An additional requirement is to eliminate the border between the readers and the author(s) of a document—in order to expand the bandwidth of their communication channel.

The rest of this paper is organized as follows. In Section 2 we present the main factors to consider in order to define a data model and an interface for electronic documents. We describe in section 3 the data model for the hypertext-like document. The aspects of the user interface are developed in section 4. There, we show the visual representation and manipulation aspects of the documents. Section 5 deals with the conclusions and further works.

2. FRAMEWORK FOR HYPERTEXT DOCUMENT MODELLING

In order to define the data model principles, the first part of this section presents the main issues for modelling hypertext-like documents. The second part presents the interface aspects for hypertext-like electronic documents processing.

2.1 Node and Link Taxonomy for Hypertext Documents

Here, a taxonomy of links and nodes is presented. The goal is to determine what kind of links could be proposed for hypertext-like documents.

Links can be classified into two groups : structured links and weak-structured links. Three main structured links exist :

—The *set-oriented* links take into account that the data space could be considered like a set of entities. Typical examples are Is-A, Is-Instance-Of or Is-Member-Of links, which are used in object-oriented databases or artificial intelligence.

—The *composite* links characterize a specific organization of a collection of node corresponding to its structure (e.g., Is-Part-Of links in aggregation hierarchies).

—The *version* links are used in some data models (e.g., Last-version, Historical-version or Alternative-version links) in order to represent the evolution of a document.

Six main weak-structured links exist :

—The *user-defined* links are required when extensibility is mandatory. The data model must supply a way to include the definition of the semantics of specific user's links.

—The *indexing* links allow multiple access points to the data space. They are extensively used in documents (e.g., From-Glossary, To-Glossary, To-Bibliography, From-Bibliography or Keyword Reference).

—The *annotation* links allow the authors of a document to define some remarks. The typical notes found on paper books (e.g., footnotes) are also possible, as well as more complex ones, such as guided tours on documents, which correspond to structural annotations [23]. Annotations are not fully considered in paper-like documents. They allow groups of users (readers or authors) to comment and communicate ideas to each other.

—The *execution* links allow the execution of programs. For instance, this facility provides dynamic changes on data, customized data display, data input and data output.

—The *weighted* links are frequently used in areas like collaborative work or idea processing (e.g., ideas to be submitted to public discussion). The definition of weights is a way to formalize the discussion in a group. They often represent the public or private relevance of a link.

—The *virtual* links may be defined using database query-like or information retrieval-like queries (boolean query or regular expression searching). Therefore, extensional linking is not mandatory, and consequently, intensive linking could be economically represented (e.g. a single node to a set of nodes by a query).

2.2 Interface Issues

In a hypertext document system, the data model *interface* defines the characteristics of the mapping between the formal nodes and edges, to their page and link representations on a screen. Links may be indicated with icons, highlighting, underlining, font change, string or special symbols. Some characteristics of the links may be defined in term of the referencing style :

—They differ in *scope*. Source and destination parts for a link may be a single node, special node elements (e.g., string, line, pixel, graphical element and special symbol) or multiple nodes [1].

—They can be *directed* or *non-directed*. These links characterize the symmetry of the relationships between nodes. Browsing semantics in an electronic document system are deeply influenced by this feature.

—They may be *typed*. Some complex type mechanisms may be defined allowing links to have several properties [6].

Therefore, formal *nodes* correspond to document parts which can be represented in several ways:

—*Single* or *multiple page* orientation is possible (e.g., one or several logical pages are associated to a node). In a multiple page orientation, one can associate either a single window (e.g., normal or scrolled) or multiple windows (e.g. overlapped or non-overlapped) [2].

—*Spatial* or *non-spatial* pages are characterized by link indicators which can be freely positioned and sized.

—*Composite* or *simple* pages are possible. Simple page means that a node directly corresponds to an atomic page (at the screen level). For composite pages, a collection of page components could be implicitly linked by nested composite links. These pages behave like a unity represented in a single associated window.

—*Typed* or *non-typed* nodes depend on the way used to define node contents and structure. If nodes are typed, node contents may be defined for unformatted data (e.g. sound, animation, bit-map image, video, text, graphic and program) and formatted data (e.g. integer and float).

The *processing model* defines the different global states taken by the interface during an interaction with the end-user. Several processes can be distinguished in processing electronic hypertext-like documents such as: structuring, editing, communicating, browsing and querying.

Structuring corresponds to the way of defining the type (e.g. the schema) of a document. This type describes intentionally the structure and contents of the instances. Types for hypertext documents should be defined as a way to specify the possible structures and contents that the documents may take. Defining document types permits to consider structural and content predicates at the query language level. So it is possible to determine, for instance, the equality or similarity between two documents. The structuring process defines a linguistic interface equivalent to the Data Definition Language used in databases.

Editing is a well known process. In hypertext, editing must be defined for the structure and for the content of documents. The two kinds of editing should be made using a WYSIWYG philosophy. The dynamical *feedback* provided for the user's actions executed under different input devices, has to be considered in this level. For instance, typical pointing devices are tablet, touch screen and mousse. Cursor treatment may be different such as continuous, jump-arrow or directed by a grid. For linking document parts, a way of link indication and creation is needed. Guidance for link creation could be indicated by rubber-banding, browsers, or other ways. Feedback from the conditions of exception must be also defined (e.g. sound).

Communicating process has to be oriented to both distributed Hypertext documents and transmitted documents among different processing tools and devices (interoperatibility). Several important aspects in network or multi-user document systems need special attention (e.g., multi-authoring and reader's location changes). Interoperatibility is necessary to freely transfer all or part of a document among different tools. For multimedia applications, device independence is also required (e.g., for several device types like visual or audible). Hypertext documents are composed of non-linear elements which must be adapted to the representation device. For instance, visual devices can be : linear (e.g., printers), single window or multi-window (e.g. prefixed, partitioned space or overlapping windows). Changing the output device may require to change the document structure and the style of presentation. For these linear devices some structures need linearization (e.g. a graph document data model for its printed representation) [18].

Browsing electronic documents has a different meaning than in traditional applications. Because of the different ways of structuring documents, browsing semantics can be indicated by authors as a link crossing guidance. Browsing can be either sequential or concurrent. Concurrent browsing has several advantages. For instance, in multi-windows environment different parallel histories could be concurrently represented. This possibility enforces understanding, exploits the presentation medium and takes advantage of human parallel processing capabilities. Browsing can be seen on three different plans. First, *structural* browsing relies on the document structure. It may need several windows simultaneously opened . Scrolling in a high level window could need synchronized scrolling of dependent windows. Otherwise, in composite link traversals, multiple windows opening for all linked nodes could be necessary. Primitives for connecting, disconnecting, overlapping windows or changes of the browsing plan are necessary. In addition, global or local structure views are needed for user guidance [10]. Next, *reference* browsing is needed when links could be freely traversed—specially when the referential network is too complex. Referential or space browsers which display the reference structure are also needed for navigation. The third class of browsing is *temporal*. Arbitrary sequences of link crossing may take place—the user may be bewildered when navigating. Temporal browsers are useful to remember already visualized nodes. They often offer an history of visited nodes and they are also used as a general tool for unlimited undo/redo on traversed links.

Querying process in hypertext is a content and structure filtering process (against querying in database, which is content oriented [7]). In document systems, browsing is as important as querying because of the low intentional nature on document reading. Readers do not know in advance neither content nor structure, so, usually they cannot specify queries [13]. However when documents are known, or in "a-priori" querying, searching relies on both content and structure specifications. Four main elements must be specified in a query : document structure description, content description, the scope of the query and the structure of the result.

At interface level, all these five processes should be represented using the same visual and operative paradigm (avoiding multiple user's mental models). So we focus on the structuring, browsing and querying process. In this paper special attention will be shown for the structuring process because it is directly related to the data model definition.

3. DATA MODEL

In order to capture node and link semantics a special purpose semantic data model is proposed. Such a model acts as a logical data model providing a second level of logical data independence. An additional goal is to represent all kinds of referencing ways found in document applications and formalize their semantics.

A tension exists between the specificity and the productivity of modelling when using a specific data model. As a high level productivity tool, this data model tries to supply in an *inherent* [5] way the typical kind of constraints and useful patterns needed for electronic documents. The formal model components directly correspond to the document elements seen at the user interface level. The representation style and its correspondence with the data model enforce convergence on multiuser modelling and document manipulation tasks.

3.1 Informal Discussion

Against conventional existing models of documents, this model is designed to allow writing and reading documents on line. Hierarchical document structuring is encouraged, however, several kinds of linking semantics allow flexible construction of virtual or intentional structures. These structures are built by means of conventionally structured database coupling. Standard hypertext-like link traversal is the way to implement standard document cross-references or annotations. Concurrent browsing is an important point to privilege in electronic document structuring. These characteristics are provided in the model by dividing contents in small subject units (freely structured and browsed) which are further aggregated and generalized to provide more complex structures.

Two organization levels are distinguished. The first one, at lower level, permits the construction of small labelled graphs acting as a representation medium for cognitive document units. In that way free idea structuring in documents is possible to be represented in these chunks. At the second level of abstraction, these graphs (which eventually could be a single node representing a piece of document) act like elementary objects. These objects are organized in more complex aggregation hierarchies built recursively using tuple, set, choice and list (group of ordered components) constructors. Low level objects are called *blocks* and high level objects are called *graphs*. Blocks are linked together to constitute graphs and these graphs are further aggregated and generalized. All blocks are differentiated by a unique system defined object-identifier *OID*.

3.2 Formal Definition

At the block level, two modelling components are considered : *static* and *dynamic* components. First, static contents are constructed from system defined elemental data types such as: number (**NUM**), text (**TXT**), boolean (**BOL**), video (**VID**), image (**IMA**), program (**PGM**), sound (SND), query (**QRY**) and OID (**OID**). An inherent *graphical frame* (as a window) is also defined surrounding the block content. Next, dynamic components are represented in a *block script* part, composed of a pre-script with input parameters and actions, a post-script with actions and output parameters, and an exception-script with exception handler. The pre-script is invoked when blocks are opened, the post-script when the blocks are closed and the exception-script when an exception condition takes place. Consequently, helped by the scripts, link traversal in the block graphs could be made for instance in a concurrent browsing way with animation. Typical actions are : close the block in 'n' seconds, change priority, visibility, show dependent blocks, execute a generic script and destruction control.

Content Blocks (atomic blocks, multi-valued blocks and multi-blocks)

Blocks are recursively defined. Let B be a string defined over an alphabet Σ^* used to name *blocks*. The most elemental block is the *atomic block* (b_a). The notation B : () is used to define a type B with its structure.

$b_a = B : (i, S, I_g, d)$ is a block

i : Block identifier.

S: Script program.

I_g: Values for the graphical information attributes : Frame and content visibilities, transparency, priority, state (selected, normal), the dimensions (X and Y) and the origin (taking its values from **NUM** x **NUM**), and the type of the graphical frame.

d : Data for the content information part taken from one of the predefined types : **NUM, TXT, BOL, VID, IMA, PGM, SND.**

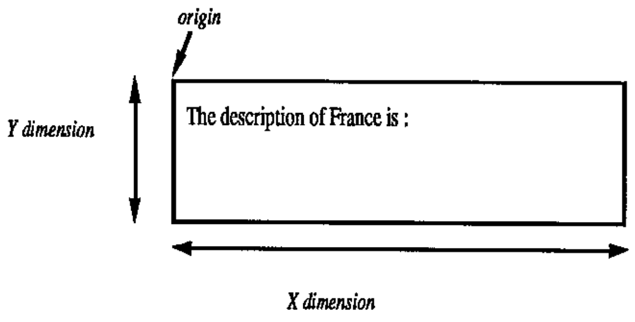


Figure 1 : Graphical representation of an atomic block instance.

Explicit OID-reference or querying are forbidden on atomic blocks because special semantic link blocks are in charge of centralizing reference control.

Blocks are recursively constructed from other blocks. So, they may content other blocks called *dependent blocks. Multi-valued blocks* (b_v) store several homogeneous blocks in a block :

$b_v = B : (i, S, I_g, d, \{b_1, b_2, \ldots, b_n\})$ is a block

i, S, I_g and d	:	as previously defined.
$\{b_1, b_2, \ldots, b_n\}$	:	is a set of dependent blocks.

Multi-blocks (b_m) could also be built from other blocks :

$b_m = B : (i, S, I_g, d, <b_1, b_2, \ldots, b_n>)$ is a block where

i, S, I_g and d	:	as previously defined.
$<b_1, b_2, \ldots, b_n>$	:	is a list of dependent blocks.

Multi-valued blocks are used to allow a block to have lists of dependent blocks visible one by one. On the other hand, multi-blocks allow a list of dependent blocks grouped for presentation purposes. All blocks form a hierarchy where the highest level block is called *root-block* (b_r) which represents the granularity unit for version control.

$b_r = B : (i, S, I_g, d, D_b, V, A)$ is a block where

i, S, I_g and d	:	as previously defined.
D_b	:	represents the set of dependent blocks.
V : <dc,vn,cu>	:	version part composed of the date of creation, version number and creator's user id.
A	:	is the name of the root-block and sharing possibilities.

Sharing blocks is an important issue. This functionality is user dependent. A shared block can not be deleted while there still exists blocks referencing it (by OID). Existential dependency relies only on dependent blocks definition (as descendants of a root-block). Therefore to be shared a block must have a specific owner which defines its existence and all other linked blocks do not determine their existence.

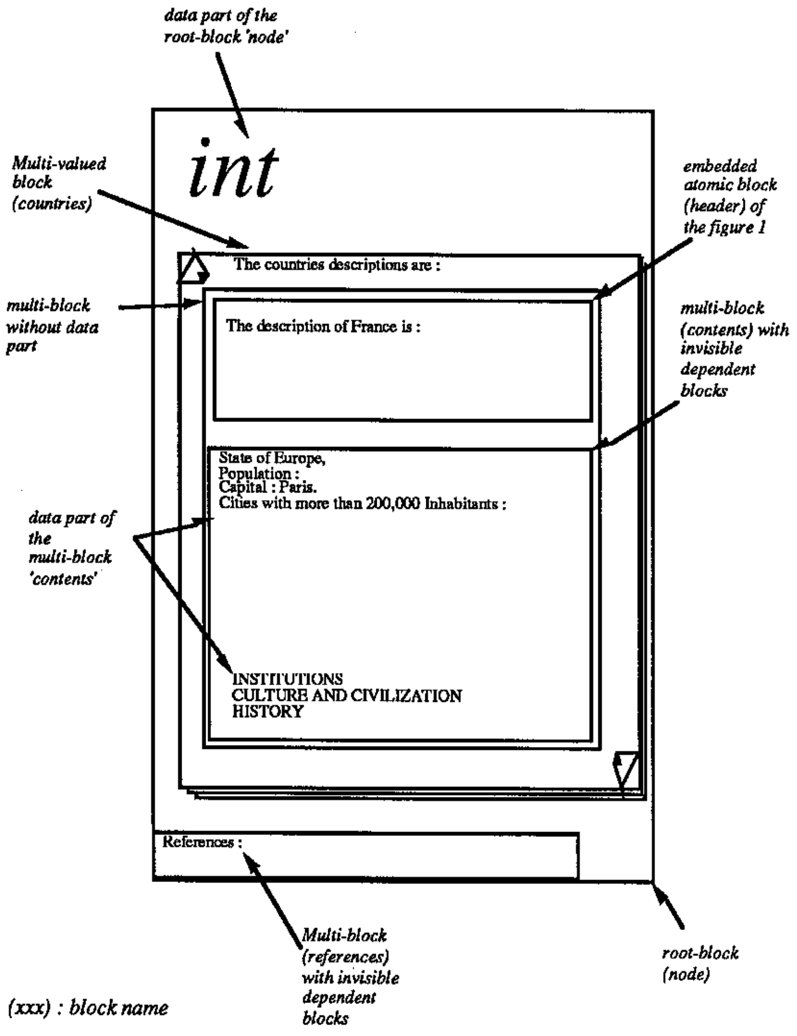

data part of the root-block 'node'
int
Multi-valued block (countries)
embedded atomic block (header) of the figure 1
The countries descriptions are :
multi-block without data part
The description of France is :
multi-block (contents) with invisible dependent blocks
State of Europe,
Population :
Capital : Paris.
Cities with more than 200,000 Inhabitants :
data part of the multi-block 'contents'
INSTITUTIONS
CULTURE AND CIVILIZATION
HISTORY
References :
Multi-block (references) with invisible dependent blocks
root-block (node)
(xxx) : block name

figure 2 : Graphical representation of a 'node' root-block instance.

Link-blocks.

Links are blocks with special semantics called *link-blocks* (b_l). The *system defined link* types are considered at the data model level as a matter of semantic constraint constructors. They represent the typical kinds of referencing found in document applications.

In system defined links the destination block is not always a block identifier. Therefore, linking (referencing) could be defined in either an *explicit* way by its **OID**, or in a *virtual* or intentional way by query expressions (**QRY**). The two forms of referencing accept two modalities : *snapshots* and *views*. The block references can be defined in several ways with its specific semantics.

For *explicit referencing* five main categories are available.

—First, *block-static-version snapshot* or *view* (labels **bsvs** and **bsvv**). These link types are used to allow a local customized display of the target block referenced by **OID.** In snapshots, the system creates a mirror block of the target block contents and its dependent blocks The target block is referenced by a special data part of the link-block (called Information : Id). As in Database environments, updates in source data are not propagated. Views are always executed when the originator block is opened. Both references use the same frame from the source block therefore preserving attributes, dimensions and origin (allowing the "customizing" of contents).

—The second one is *block-last-version snapshot* or *view* (labels **blvs** and **blvv**). These links have the same semantics that the previous links but they (may) reference the last version block. Versional blocks are specified by the user.

—Third, *block-static-version reference* (label **bsvr**) link, represents the typical referential integrity used in databases. In document applications, this constraint is checked when a block is updated. The value of block origin must exist in the referenced block.

—Fourth, *block-last-version reference* (label **blvr**). This link has the same semantics that the previous one but it always defines a reference to the last block.

For *virtual referencing*, two main modalities are available :

—First are the *query-based snapshot* or view (labels **qbs** and **qbv**). They are virtual links represented by storing a specific text of a query in the information part (I_d) part of link-blocks. The aim is to copy the blocks satisfying the query into the originator block. Therefore, originator block is multi-valued because the evaluation of the query returns a set of blocks. These blocks are stored in a mirror block (with or without their descendent blocks).

—Second is the *query-block reference* (label **qbr**). Here, updating the source block constraints its content to be equivalent to the blocks referenced by the query.

Two sets are distinguished from the system labels : **L(OID)**={ **bsvs, bsvv, blvs, blvv, bsvr, blvr** } representing the OID-based references, and **L(QRY)**={ **qbs, qbv, qbr** } representing the query-based references. So let Ls be a set : Ls = **L(OID)** $\cup$ **L(QRY)**.

To supply extensibility, the *user defined links* allow to construct links with specific semantics by means of script programming and link labelling. User defined links have no inherent semantics. They always made explicit references. Therefore, there is a single type of link called *user-tagged reference* (user's defined label). Link semantics are similar to hypertext links but specific actions (like animations), concurrent browsing (similar to token passing), user's specific programmed constraints and multiple representations of data are permitted using nodes and link scripts. Consequently, the user's defined labels constitute a set called **Lu** = { *utr1, utr2,...,utrn* }.

User-tagged reference links are used to build *atomic graphs*. These graphs are useful for two purposes. First, to represent document units browsed as a unitary knowledge body. They are called *internal links* (**ILT**). Next these graphs are used to reference other documents or graphs in the same document. They are called *external links* (**ELT**). We made this distinction because at user interface level, internal-links are seen by default (by an internal graph browser) against external links which are selectively seen depending on browsing scope (see figure 4). Therefore, a typing mechanism is defined for the intentional definition of the document units graphs. The system makes the difference between an internal reference and an external one by means of a graph identifier part stored with the block identifier. All user tagged links are represented by a user defined symbol stored in its data part (*d*).

Links blocks are defined by :

$b_l = B : (i, S, I_g, d, l, I_d)$ is a block where

i, S, I_g and d	:	as previously defined.
l	:	is a label of **Ls** $\cup$ **Lu** where **Ls.**
I_d	:	is a block destination information part of type **QRY** or **OID.**

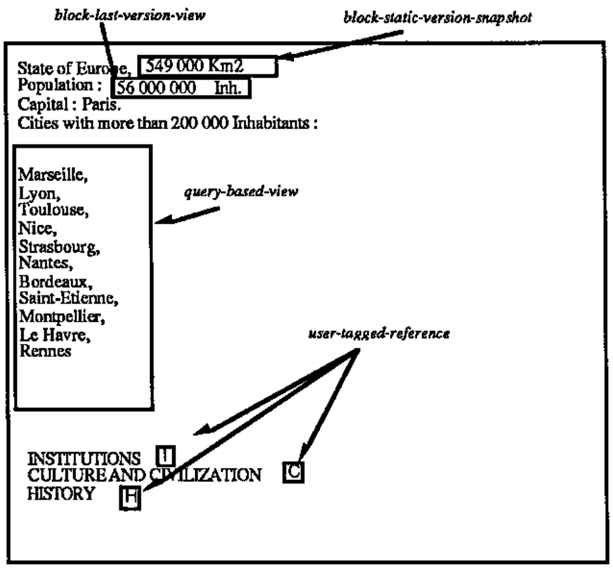

block-last-version-view
block-static-version-snapshot
State of Europe, 549 000 Km2
Population : 56 000 000 Inh.
Capital : Paris.
Cities with more than 200 000 Inhabitants :
Marseille, Lyon, Toulouse, Nice, Strasbourg, Nantes, Bordeaux, Saint-Etienne, Montpellier, Le Havre, Rennes
query-based-view
user-tagged-reference
INSTITUTIONS I
CULTURE AND CIVILIZATION C
HISTORY H

Figure 3 . Representation of the multi-block 'contents' : A referencing example.

Using the blocks attributes and playing with their visibility the final look of the example root-block ('node') is shown in the figure 5.

Recursive construction of atomic graphs allows to construct an *aggregation hierarchy* where the graphs are intrinsically linked by an is-part-of relationship. Therefore, they describe a structured body of knowledge for a whole document. They are represented like complex objects used in others Database fields (CAD/CAM, OIS,...)[4]. The difference here is that the low level components are allowed to be small graphs. The highest level component of an aggregation hierarchy represents complex graph objects which may be grouped in classes. These classes could be further generalized in other classes called superclasses.

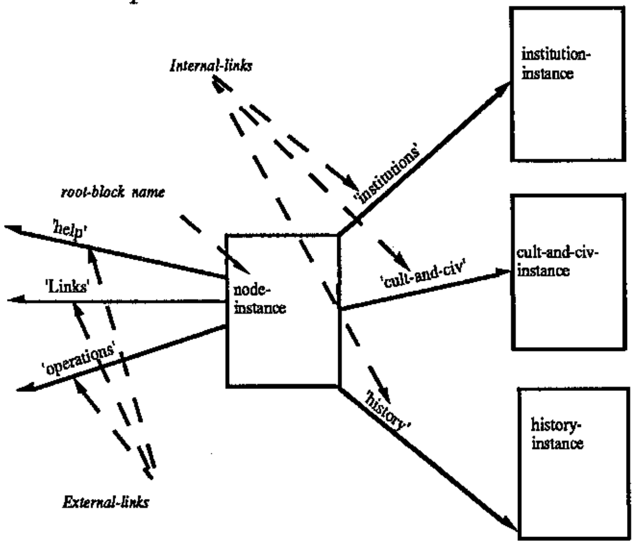


figure 4 : Internal -links and External-links.

3.3 Types and Schemata

A single mechanism is necessary to allow the identification of a block along the data space. So, the identifier of a block has to be divided in four components allowing its identification at a block level, a root-block level, a graph level, and a class level (*i : (block-part-identifier, root-block-part-identifier, graph-part-identifier,class-part-identifier) . To simply i is written* (b_i, r_i, g_i, c_i)).

Let I(t) be a function which returns a block (instantiation) of type t. Let TYPE (b) be a function which returns the type of a block b. Let DATA_TYPE (d) be a function which returns the type of the data part of a block. Let OID (b) be a function which returns the OID of a block b. Let DATA (b) be a function which returns the content part of a block b. Let QUERY (Id) returns the set of the relevant blocks for the query Id. Let T_c = {NUM, TXT, BOL, VID, IMA, PGM, SND}, T_q = {QRY, OID} and T_s = {NUM, TXT, BOL, IMA}. The notation B : () is used to define the structure of a bloc instance bc.

DEFINITION

a) an *atomic-type* $B : T$ is a type and

$bc = I\,(B : T)$

$bc = B : (i, S, I_g, d) \Rightarrow \quad DATA\_TYPE\,(d) = T \wedge T \in Tc$

b) a *multi-valued type* $B : T : \{T_1\}$ is a type and

$bc = I\,(B : T :\{T_1\})$

$bc = B : (i, S, I_g, d, \{b_1,b_2,\ldots,b_n\}) \Rightarrow \forall\, i\; TYPE\,(b_i) = T_1 \wedge DATA\_TYPE\,(d) = T \wedge T \in Tc$

c) a *multi-block type* $B : T : <T_1,T_2,\ldots,T_n>$ is a type and

$bc = I\,(B : T : <T_1,T_2,\ldots,T_n>)$

$bc = B : (i, S, I_g, d, <b_1,b_2,\ldots,b_n>) \Rightarrow \forall\, i\; TYPE\,(b_i) = T_i \wedge DATA\_TYPE\,(d) = T \wedge T \in Tc$

d) an *internal-link type* $B : T : \boldsymbol{ILT}$ is a type and

$bc = I\,(B : T : ILT)$

$bc = B : (i, S, I_g, d, l, I_d) \wedge l \in \mathbf{Lu} \Rightarrow DATA\_TYPE\,(Id) = OID \wedge DATA\_TYPE\,(d) = T \wedge$
$T \in Ts \wedge i=(b_1,r_1,g_1,c_1) \wedge I_d = (b_2,r_2,g_1,c_1) \wedge$
$\exists\, b\; / OID(b) = I_d \wedge b_1 \neq b_2 \wedge r1 \neq r2 \wedge$
$DATA\_TYPE\,(DATA\,(b)) \notin Tq$

e) an *external-link type* $B : T : \boldsymbol{ELT}$ is a type and

$bc = I\,(B : T : ELT)$

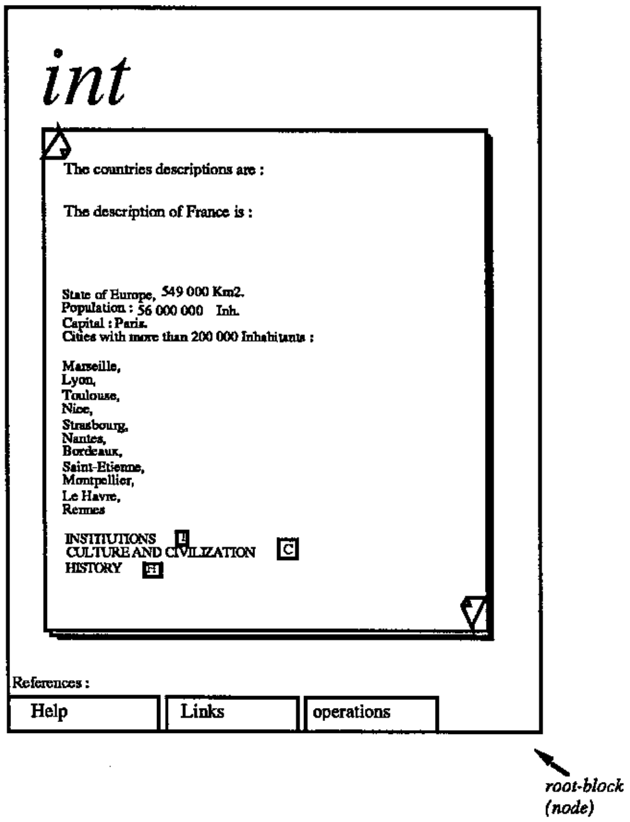
$bc = B : (i, S, I_g, d, l, I_d) \wedge l \in \mathbf{Lu} \Rightarrow DATA\_TYPE\,(Id) = OID \wedge DATA\_TYPE\,(d) = T \wedge$
$T \in Ts \wedge i=(b_1,r_1,g_1,c_1) \wedge I_d = (b_2,r_2,g_2,c_2) \wedge$
$\exists\, b\; / OID(b) = I_d \wedge b_1 \neq b_2 \wedge r1 \neq r2 \wedge g_1 \neq g_2 \wedge c1 \neq c2 \wedge$
$DATA\_TYPE\,(DATA\,(b)) \notin Tq$

f) a *system link type* $B : T : \boldsymbol{SLT}$ is a type and

$bc = I\,(B : T : SLT)$

$bc = B : (i, S, I_g, d, l, I_d) \wedge l \in L\,(OID) \Rightarrow DATA\_TYPE\,(Id) = OID \wedge DATA\_TYPE\,(d) = T \wedge$
$T \in T_s \wedge \exists\, b\; / OID\,(b) = I_d \wedge$
$DATA\_TYPE\,(DATA\,(b)) \notin T_q$

$bc = B : (i, S, I_g, d, l, I_d) \wedge l \in L\,(QRY) \Rightarrow DATA\_TYPE\,(Id) = QRY \wedge DATA\_TYPE\,(d) = T \wedge$
$T \in Ts \wedge \;\forall\, b\, \in QUERY\,(I_d) \wedge$
$DATA\_TYPE\,(DATA\,(b)) \notin T_q$



The countries descriptions are :

The description of France is :

State of Europe, 549 000 Km2.
Population : 56 000 000 Inh.
Capital : Paris.
Cities with more than 200 000 Inhabitants :

Marseille,
Lyon,
Toulouse,
Nice,
Strasbourg,
Nantes,
Bordeaux,
Saint-Etienne,
Montpellier,
Le Havre,
Rennes

INSTITUTIONS I
CULTURE AND CIVILIZATION C
HISTORY H

figure 5 : Final representation of the root-block 'node'.

Root-blocks and internal links modelled by a graph-schema, represent an oriented local graph for a local browser. Root-blocks and external links modelled by a graph-schema, represent an external graph for an external browser. These graphs can be further aggregated to form more complex documents.

Recursive use of these constructors forms a type hierarchy. The highest element of the type hierarchy is called *root-block type* and the *root-block schema* is defined as the regular expression (following the type definition of section 3.2) representing all its component blocks. For example, the regular expression defining of the root-block type of figure 2 (without any detail for ELT and ILT) is :

Node : (i_0, S_0, Ig_0, Logo : IMA,
{Country : (i_1, S_1, Ig_1,
Intro : TXT, { (i_3, S_3, Ig_3, d_3,
<Header : TXT,
Contents : (i_4, S_4, Ig_4,d_4, <Km : bsvs, Hbt : blvv, Intro : TXT, City : qbv,
Inst : ILT, Civ : ILT, Hist : ILT >)}),
References (i_2, S_2, Ig_2, intro : TXT, <help : ELT, links : ELT, operations : ELT>)
}, V_0, A_0) with $I(d_3) = \emptyset$.

The root-block schemata are grouped in order to constitute elementary graphs representing an *atomic graph type*. These graphs represent a document part browsed as a unit.

DEFINITION :

Let Sr1,Sr2,...,Srn be root-block-schemata :
An *atomic graph type B : / Sr1,Sr2,...,Srn /* is a type.

Let GT1,GT2,...,GTn be atomic-graph-types :

a) a *set-graph type = B : {GT1}* (set of)
b) a *choice-graph type = B : $ GT1, GT2,...,GTn $* (one of these types is instanced)
c) a *tuple-graph type = B : [GT1,GT2,...,GTn]* (aggregation of components)
d) a *list-graph type = B : %GT1,GT2,...,GTn%* (group of ordered components)

are types describing domains in the same way that the complex object models do [3].

4. INTERFACE CONSIDERATIONS

Interface design should consider the way of displaying both the formal document objects and the processing model. Formal document objects are atomic blocks, multi-valued blocks and multi-blocks. Each kind of these blocks have its own representational characteristics (Figure 2). So, each block can easily be identified by its frame shape and contents. Atomic and multi-blocks are represented using the same single frame having a simple shape. They can be easily identified because multi-blocks have embedded blocks as contents. Multi-valued blocks have a special frame which suggests that their contents are a set of blocks. Link-blocks are represented by a single frame with a user-defined content symbol from the system defined available types (see Figure 3).

Considering a document like a graph, content blocks and system-defined link block correspond to graph nodes. User-defined links correspond to usual directed edges (e.g. hypertext links). So, nodes can, be single or multi-page (e.g. atomic and multi-valued blocks). They are also spatial in the sense that they may be positioned anywhere in the space limited by the frame of their parent block. Root-blocks (having no parent block) may be considered as embedded in a system defined universal frame. Nodes are also single or composite (e.g. atomic blocks and multi-blocks), and they are always typed (see section 3.2).

Links blocks are directed, typed and are treated like a special kind of document objects (always embedded in content blocks). Inside these blocks, they may be freely positioned. System-defined links behave at representational level like content blocks. They permit a personalized display representation of the target block content (see Figure 3). Therefore, they are able to represent blocks with virtual contents, oriented to data sharing and structured database coupling.
User-defined links are aimed to cross-referencing. Internal links are used to represent a weak-structured body of information behaving like a whole unit represented by a graph (Figure 4). External links permit weaker forms of referencing and are oriented to represent non-restricted document links.

4.1 Structuring

Structuring process should be related to a description language aimed to intentionally define the contents and the structure of documents. A natural way is to define a visual interface allowing the construction of the types and schemata described in section 3.3. For flexibility reasons, this type system is enhanced to allow (1) to store generic structures, generic scripts, generic contents shared by all instances of a type, (2) variant definitions for types, and (3) some data constraints. Normally, types of electronic documents share generic data. This facility does not increase the power of the type mechanism but defines an "interface sugaring" to the underlying language making easier to use the document system. Some directions for instantiations and constraints have also to be included in the type language. Variant type definition is necessary because electronic document cannot be easily characterized by attributes of a simple type. Variant types force to define instances as dynamic instances (e.g. instances which carry out their type description). In general, using dynamic values allows a more flexible typing system. It permits for example, to specify the way of matching instances to types at query language level (e.g. structural, name or mixed matching criteria).

Link types may also enhanced to specify the type of the destination link : for both internal and external link types. This mechanism is equivalent to the types pointers implemented in some programming languages. Internal, external, user-defined and system-defined links make clear the semantics of referencing in documents. The data model offers a clearer semantics of referencing than in object-oriented databases (where object identity references may be used for composition with dependent objects, for sharing or for other kinds of referencing in an indistinct way).

Types are associated to document classes. Classes may be seen like folders containing atomic graphs. Therefore, atomic graphs are like traditional files, but they may be freely structured and cross-referenced. A class behaves like an Abstract Data Type hiding its implementation and showing an interface by means of commands. It defines the semantics of types : directing the instantiation and incrementally checking the types of its instances. Classes also know some system defined operations allowing the manipulation of its instances.

Structuring process may operate under existing types by derivation or it may create any new type. Derivation uses the fact that an existing type can be used like a prototype or a generic type in order to define other new types. So, to define a type similar to an existing one, the actions are (1) derive from the existing type, (2) rename it, (3) introduce the transformations, (4) save it. In consequence, all defined types behave like parametric types where parameter changes are aimed to new type definition. The available transformation commands are for a block : to change the name of a block type, to cut the selected blocks, to paste these blocks, to add a dependent block, to select a block and its descendent blocks, to change the system type of a block, and to change a block from multi-valued to multi-block or vice-versa.
The available transformation commands are for a link : to rename the link, to define link aliasing, to delete a link, to add a link, to change the type of a link, to define or to change a target type for a link, and to change the attributes of a link (e.g. mandatory link or optional).

This process also offers the integration of a view definition style operating by graph transformation. Each user has a private view of the underlying document. The view concept is aimed to implement a security control, to allow personal annotations and to define personal transformations of the underlying document. Views may be also used like document abstractions controlling the level of detail for a document (e.g. in a hierarchical way). Each view is considered like a data structure operating over the document or over other views. Each view stores also a local state, which permits to make persistent the last display representation of a document (preserving certain document aspects among the different sessions). Several commands are required to manage a view : to define, to create, to derive, to save, to cancel, to hide a block, to show a hidden block, to rename a block or a link, to define link aliasing.

Structuring process may be also used to define some document constraints : (1) browsing constraints used to show an icon when opened or when closed a given block (for concurrent browsing), (2) spatial constraints to define a position range for a block or to forbid the overlapping of some blocks, (3) block cardinality constraints (e.g. the number of allowed instances in a multi-valued block), (4) structure constraints like cardinality constraint to limit the permitted maximal and minimal number of some link type (e.g. define optional or mandatory user links), and to limit the maximal number of links arriving to a given block and (5) domain constraints to specify, for instance, the range of an atomic block of number type.

4.2. Editing, browsing and querying

These processes are not fully considered here for space limitations. Communication process is not of interest in our project. However, in order to show the general philosophy of these processes, this section presents what should be some of their main characteristics.

Editing process has to rely on a WYSIWYG philosophy. It must consider a kind of dynamic type checking executed when an edited instance is saved. Type checking must consider the types of all the blocks of the instance and the type of the structure (e.g. check optional or mandatory links). Editing by instance derivation has to be possible.

Browsing process must indifferently allow browsing in the documents or in the interface. So, an interface is like document or vice-versa. The problem presented for temporal versus spatial browsing must also be considered. A certain kind of abstraction of blocks and graphs must be taken into account to diminish user's bewilderment and cognitive overhead [8]. Browsing should implement a form of navigational language intended for low intentional aspects on reading electronic documents.

Querying process has to be implemented as another hyperdocument. Querying must be designed for the structure and for the contents of electronic documents. In order to define a structure, the user specifies the blocks and their types in a similar way that he derives new types in the structuring process. This derivation provides a general structure (like a form) with empty blocks. In order to specify content conditions, he specifies them in a fill in the form philosophy. Both, the navigational language of the browsing process and the declarative language of the querying process, compose an ad-hoc query facility to the low intentional nature of electronic document reading.

5. CONCLUSIONS

The first part of this paper defines a framework for hypertext (hypermedia) document modelling. We show the main link semantics required for hypertext-like documents and describe the interface characteristics in order to provide a general framework, which could be used to design other document data models.

Next, we have proposed a document data model that considers content, structural and presentation aspects. This model allows general abstraction techniques and extensible semantics for information structuring with: (1) A stratified method to represent multimedia document data. (2) Several linking modalities, which offer a flexible way to represent the multiple relationships in data—oriented to the "structure-intensive" and data-intensive nature of multimedia document management.

Links allow information sharing specifications and multiple styles for elaborating and structuring documents. Some system defined links (e.g., query-based) propose a *view* mechanism, which permits to consider a view, not only as an information filtering process, but as an information adding, transformation and multiple representation process. This links also allow structured database coupling.

The user interface offers a page model based on a nested block philosophy, which is the basis of all the interface process. This interface proposes a unified environment that could be used like (1) An *authorship* tool with a simple way of editing the structure and content, managing versions, defining types and structures. (2) A *reader's* tool with a unified query language for structure and content queries, as well as, a flexible browsing style useful for the weak intentional nature of reading unstructured data. (3) A multiple indexing method allowing to increase the precision and recall [17] by means of free cross-referencing and structural queries.

Electronic documents should exploit hypertext-like characteristics. Realistic electronic document processing is related to richer ways of structuring, presenting (hypermedia) and indexing documents (multiple access). These characteristics try to alleviate output device limitations (e.g., small screens). However, the existing flat documentation, (e.g., paper documents or electronic mail), should be adapted to Hypertext structures. Logical structure and content analysis research are necessary in order to determine how the existing documents may be adapted to the new models or restructured.

The future research works will be dedicated to the further definition of the visual language for document manipulation and browsing. Therefore, document definition, editing, browsing and querying will be defined with the same principles avoiding multiple representational models.

REFERENCES

1. AKSCYN, R. M., MCCRAKEN, D. L., AND YODER, E. A. KMS: A distributed hypermedia system for managing knowledge in organizations. *Commun. ACM 31*, 7 (July 1988), 820-835.
2. AMBLER, A. L., AND BURNETT, M. M. Influence of visual technology on the evolution of language environments. *IEEE Computer* 22, 10 (Oct. 1985), 9-22.
3. BANCHILON, F., AND KHOSHAFIAN S. A calculus for complex objects. In *Proceedings of the ACM SIGACT/SIGMOD/SIGART Symposium on Principles Of Database Systems* (Cambridge, MA, USA, Mar. 1986) 53-59.
4. BANERJEE, J., CHOU, H., GARZA, J., KIM, W., WOELK, B., BALLOU, N., AND KIM, H. Data models issues for object oriented applications. *ACM Trans. Off. Inf. Sys. 5*, 1 (Jan. 1987), 3-26.
5. BRODIE, M. L. On the development of data models. In *On Conceptual Modelling*, Brodie, M. L., Mylopoulos, J., and Schmidt, J. W., Eds. Springer-Verlag, New York, 1984, 19-47.
6. CAMPBELL, B., AND WOODMAN, J. M. HAM : A general purpose hypertext abstract machine. *Comm. of the ACM 31*, 7 (July 1988), 856-861.
7. CHANDRA, A. K. Theory of database queries. In *Proceedings of the ACM SIGACT/SIGMOD/SIGART Symposium on Principles of Database Systems*, (Austin, Texas, Mar. 1988), 1-9.
8. CONKLIN, J. Hypertext : An introduction and survey. *IEEE Computer 20*, 9 (Sept. 1987), 17-41.
9. ELLIS, C. A., AND NAFFAH, N. *Design of Office Information Systems*. Surveys in Computer Science, Springer-Verlag, New York, 1987.
10. FEINER, S. Seeing the forest for the trees : Hierarchical display of hypertext structures. *ACM SIGOIS bulletin 9*, 2 & 3 (Apr.-June 1988), 205-212.
11. HALASZ, F. G. Reflections on Notecards: Seven issues for the next generation of hypermedia systems. *Comm. of the ACM 31*, 7 (July 1988), 836-852.
12. HORAK, W. Office Document Architecture and Office Document Interchange Formats : Current status of international standardization. *IEEE Computer 18*, 10 (Oct. 1985), 50-60.
13. MARCHIONINI, G., AND SCHNEIDERMAN, B. Finding facts vs browsing knowledge in hypertext systems. *IEEE Computer 21*, 1 (Jan. 1988), 70-80.
14. ORENSTEIN, J. A. Can we meaningfully integrate drawings, text, images and voice with structured data?. In *IEEE Proceedings of the International Conference on Data Engineering* (Los Angeles, CA, Feb. 1988). IEEE Computer Society Press, Washington, D.C., 1988, 603.
15. PECKMAN, J., AND MAYANSKI, F. Semantic data models. *ACM Computing Surveys 20*, 3 (Sept. 1988), 153-189.

16. PHILLIPS, B. Multimedia systems and text. In *IEEE Proceedings of the International Conference on Data Engineering* (Los Angeles, CA, Feb. 1988). IEEE Computer Society Press, Washington, D.C., 1988, 601.
17. SALTON, G., AND MCGILL, M. J. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., New York, 1983.
18. SHASHA, D. When does non-linear text help?. In *Proceedings of the International Conference on Expert Database Systems*, (Charleston, South Carolina, Apr. 1986), 163-174.
19. SHETH, A. Managing and integrating unstructured and structured data : Problems of representation, features, and abstraction. In *IEEE Proceedings of the International Conference on Data Engineering* (Los Angeles, CA, Feb. 1988). IEEE Computer Society Press, Washington, D.C., 1988, 598-599.
20. STOTTS, P. D., AND FURUTA, R. Petri-net-based hypertext : Document structure with browsing semantics. *ACM Trans. Inf. Sys. 7*, 1 (Jan. 1989), 3-29.
21. TOMPA, F. W. A data model for flexible hypertext database systems. *ACM Trans. Inf. Syst 7*, 1 (Jan. 1989), 85-100.
22. YANKELOVITCH, N., MEYROWITZ, N., AND VAN DAM, A. Reading and writing the electronic book. *IEEE Computer 18*, 10 (Oct. 1985), 15-30.
23. ZELLWEGER, P. T. Active paths through multimedia documents. In *Document Manipulation and Typography*, J. C. van Vliet, Ed. Cambridge University Press, Apr. 1988, 19-34. *Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography* (Nice, France, Apr. 1988).