Design and Analysis of Active Hypertext Views on Databases

G. Falquet*, J. Guyot*, L. Nerima*, S. Park**
* Centre Universitaire d'Informatique, University of Geneva
24, Rue General-Dufour, CH-1211 Geneve 4, Switzerland
** Information Sciences Institute[1]
4676 Admiralty Way, Marina del Rey, CA 90292-6601, USA
{Gilles.Falquet, Jacques.Guyot, Luka.Nerima}@cui.unige.ch, separkht@isi.edu

1. Introduction : Hypertext Views and Database Publishing

The simplicity and ease of use of the hypertext paradigm are probably among the most important factors that lead to the rapid success of the World-wide web. In particular, the concept of hypertextual navigation has proven highly efficient for easily accessing information without prior knowledge of its structure and organization. Contrary to what happens with databases, the hypertext user does not need to learn a sophisticated query language, like SQL, to retrieve information. Nevertheless, storing, retrieving, and processing large amounts of data in an efficient and secure way requires the use of database technology. This is why most organizations store their information in databases, which are at the heart of their information systems. It must also be noted that highly structured data are necessary to ensure exhaustive processing, i.e. to be sure that all the relevant information has been retrieved or processed.

The idea of using the hypertext paradigm of the Web as a universal database interface emerged during the early days of the Web. It seems that the ability to access a database from any personal workstation, through a Web browser, without installing any specific client application, was another success factor of the Web. Recent studies claim that 80% of the information available on the Web is actually stored in databases and generated on the fly (Lawrence and Giles, 1998). This shows that database publishing on the Web is an important activity in the general field of web engineering.

In this perspective, we can define the notion of hypertext view of a database as a set of hypertext nodes and links that

- presents the contents of the database to the hypertext user
- replaces database querying by hypertext navigation
- enables the authorized user to act on the database in order to update its content.

[1] This work was done while working at the University of Geneva.

Thus, publishing a database on the Web amounts to designing and implementing a hypertext view of the database in the Web context. This means that the view must be composed of Web pages produced by a HTTP server somehow connected to the database. These pages can be either dynamically computed on demand or generated once and stored as HTML files. In this latter case a view maintenance strategy must be set up to propagate database updates to the view files. We will see in the next section that there are several approaches to program or specify the generation of Web views.

In addition to such implementation issues, publishing a database on the Web raises many design issues that are related to hypertext and database design. Since database and hypertext models are based on radically different information representation and processing paradigms (Conklin 1987)(Nanard and Nanard 93), obtaining an efficient hypertext view cannot be accomplished by a simple transformation of the database schema. However, if one focuses on information representation, not considering information retrieval and processing, one can use different strategies to map the contents of a database onto hypertext components. Designing efficient and effective hyperspaces is a hard task, probably because there is an extremely large number of paths that a user can follow. It is thus difficult to ensure that the user will be able to reach any information node, that he or she will not get lost or disoriented in the hyperspace, that any information can be reached within a reasonable amount of time/number of clicks, etc.

Database publishing can be combined with database design and implementation when creating new web sites. In that case, the database does not pre-exist, it is designed and set up with the aim to store the site›s information. In that case the publishing problem is slightly different from the general problem since the database has been specifically designed for managing a web site. We will see how the Araneus system deals with this problem. The database to be published can also come from the integration (wrapping) of multiple existing databases (and other data sources).

In this chapter we will first review existing approaches to databases publishing, then we will present the Lazy hypertext view specification language and system. Using the Lazy framework we will then study the hypertext view design problem and propose a design methodology. Finally we will see how to create active and adaptive hypertext views .

2. Backaground

2.1. Procedural approaches (CGI scripts, Servlets, database tools)

In the procedural approach, the hyperspace designer must write a set of server programs, for instance CGI programs in C or Perl, Java servlets, or PHP scripts that will be invoked by an

HTTP server to process specific HTTP requests. In response to a request such programs will typically prepare a database request, send it to the database server and then process the results to generate an HTML page. Although this approach is widely used, it has several important drawbacks. Firstly, server programs contain in fact three different languages: the programming language, SQL statements to query and update the database, and HTML to create the output page. From a software engineering point of view this leads to serious development problems. For instance, it is not clear how to type check or debug such a mix of code. The second problem is the hiding of the hypertext structure within the program's code. The hypertext structure is hidden for two reasons: 1) the internal (HTML or XML) structure of the produced page appears only in the "write" instructions of the program, so it is neither apparent to the program's reader nor checked by any compiler; 2) the same server program can produce several completely different pages. So there is no one to one correspondence between the programs and the page types. At one extreme a single program could generate all the pages of a site, while, at the other extreme, different programs could generate the same type of page. So there is clearly a software architecture problem due to the mismatch between the programs' structure and the site structure. For these reasons, such a code is demanding to maintain and update. Of course, a rigourous design practice should strive to maintain a good matching between the programs' and hypertext structures, but this is not enforced by the procedural approach.

An interesting enhancement to the procedural approach is the dynamic document approach. The dynamic document approach consists in extending some document markup language (such as HTML) with specific tags for database querying, result processing and formatting. See for instance Cold Fusion (www.allaire.com). These tags introduce procedural parts in the document description. A similar approach consists in including procedural code into page descriptions as in Java server pages (HTML + Java) and ASP (HTML + Visual Basic). Although this technique overcomes the most striking drawbacks of the "pure" procedural approach, it still mixes three languages in a single page. In particular, the application or business logic is intermingled with the presentation logic.

2.2. Declarative approaches:

The declarative approach rests on explicit declarative representations of both data structure and web site structure and a (query) language to transmit the data from the first structure to the second one.

Building a web site using a declarative language for describing the structure of the site has several advantages. Primarily, it allows to easily create multiple versions of the site over the

same data. It also allows to support easily the evolution of both database and web site structures (see (Florescu, Levy, and Mendelzon, 1998) for other considerations). Several systems have been developed using a declarative approach. Their aim is to provide Web views over (heterogeneous) data sources (see (Mendelzon and Atzeni, 1998) and (EDBT, 1998) where many of these systems are described).

The key point of these systems is that they provide a declarative representation to define a web site as a view over existing data. Although this view concept is very close that of the traditional database one, the main difference is that it expresses web pages as well as hypertext links.

More precisely, the systems using a declarative approach provide usually three kind of declarative languages:

- a data modeling language for expressing the source data
- a language for expressing the web site structure
- a language for expressing the graphical layout of the web site pages (e.g. markup tags, style sheets, etc.)

The source data come from a structured or semi-structured database or from web pages. The data are represented in term of the data model, possibly after a uniformization process by the mediator. The web site structure expression contains queries over the underlying data to define the transformation between the source data to their representation in the web structure. Eventually, the visual representation definition is applied to generate the effective web pages.

For example, in Araneus (Atzeni, Mecca, and Merialdo, 1998) the source data model is the relational data model and the web site structure is defined by a set of page-schemes expressed in the Penelope language. This language is based on a reference data model called ADM (Araneus Data Model). In ADM, pages are described by atomic attributes, complex attributes such as nested lists of tuples, and hypertext links. The Penelope language thus maps relational tables to ADM structures.

Several systems are able to deal with heterogeneous data. In addition to the data model, they provide wrappers and a mediator to translate the different kind of data in a uniform data structure. For instance, the Strudel system (Fernandez et al., 1998) is intended to exploit data from various sources: databases (relational or object-oriented), structured files or existing web-sites. It provide wrappers and mediator to give a uniform view of the source data. The others features of Strudel are:

- a unique model for representing both data structure and site structure: the labeled directed graph model

- a query language called StruQL for integrate the source data (mediator) and for transform the structure of data to the structure of the web site
- a HTML template language for specify the graphical presentation of the web site.

The Strudel methodology advocate the importance to separate the three main task of web site creation: (1) the management of data, (2) the management of site structure a and (3) the design of the graphical representation.

Other works such as (Anderson, Lavy, and Weld, 1999), (Ceri, Fraternali, and Paraboschi, 1998)., (Toyama, Nagafuji, 1998), (Cluet, Simeon, 1998), (Dar et al., 1998) are representative of the declarative approach.

The Lazy language and system that we will present and use in the following section also belongs to the declarative approach. It is based on the relational data model and on a hypertext model that is more sophisticated than the web model. In particular, this model supports inclusion links (to build complex hierarchical pages) and active links (to trigger database operations).

3. Lazy: A Declarative Language to Specify and Implement Hypertext Views

Specifying a hypertext view of a database amounts to define a mapping from database objects (relational tables, tuples, etc.) to hypertext components (nodes and links). We must first fix a database and a hypertext model to work with. In this chapter we will use the relational data model because it is well defined and by far the most widely used in practical applications. However, the concepts and even the language that we will present require very few changes to support an object-oriented data model (as described in (Falquet, Guyot, and Nerima, 1998)) or a semi-structured data model.

3.1. The hypertext model

The basic Web hypertext model, whose main concepts are the HTML page and the URI (uniform ressource identifier), is simple to implement but it lacks features found in other interesting models. In particuler, the Web has a single type of links that are uni-directional, with a single target node. There is also a single link behaviour, namely "jump to the refered node when the anchor is clicked". There is no notion of containment. In particular, a node cannot be a sub-node of another one. One way to circumvent these limitations consists in working with a more sophisticated hypertext model (as in Araneus) that will eventually be mapped onto the standard Web model. This is the approach taken in Lazy. The Lazy hypertext model has different types of

links (reference and inclusion) and link behaviours (jump and expand-in-place). The content of a node is a tree of XML elements.

We consider a simple hypertext model whose structural part is composed of nodes, anchors, and links. Each node has a unique identity and a content. The content of a node is a sequence of elements which may be character strings, images, etc. An anchor is an element or a sequence of elements within the content of a node, it serves as a starting or ending point of a link. A link is defined by its starting and ending anchors and by its category which is either ‘reference’ or ‘inclusion’. Reference links are intended to create a navigation structure within the nodes. Inclusion links are intended to create nested structures that represent complex contents (structured documents).

This model can easily be mapped on the Web model. The only problem comes from inclusion links because, in HTML, the <A HREF...> tag corresponds to reference links and there are no inclusion links (except for images, with the IMG tag). The approach we took when creating tools for the Web was to represent in a single HTML page the content of a node and of all its subnodes and to use embedded list tags to show the inclusion structure (Fig. 2 shows an example).

3.2. Node schemas and node instances

A hypertext view specification consists of a set of node schemas, which correspond to node types. Each node schema specifies the data collections (relational tables) from which the node's content is to be drawn; the selection and ordering criteria; the elements that form the node content; and links to other nodes. A simple node schema takes the following form:

node ***node-name*** **[** ***parameter_list*** **]**
element_list
from ***collection, ...***
selected by ***boolean_expression***
ordered by ***numeric_expression***

The content of a node is specified by a list of content elements. An element is either a compound element of the form **<** *element_type* **>** **(** *element*, ... **)** or a simple expression based on litteral constants, database attributes, functions, and operators. Element types are HTML or XML tag names (depending on the target language).

The actual hypertext view consists of instances of view schemas. A node instance is determined by its schema name and a list of actual parameter values. The content of an instance is obtained by querying the database according to the selection condition, and then evaluating the content elements on the selected tuples.

Example. This example and the subsequent ones are based on the same database schema that represents a part of a virtual (or real) museum database. It consists in the following relation schemas (key attributes are shown in bold)

work(**wno**, title, c_date, author, picture, support, height, width)

artist(**ano**, name, birthdate, deathdate)

art_cnty(**ano, country**, activity)

exhibition(**exno**, title, organizer, description)

ex_content(**exno, wno**, comment)

Consider the following node definition

```
node Artists_after[date]
       { <p>(name, "(born ", birthdate, ") ")  }
  from artist
    selected by birthdate >= date
    order by name
```

It is intended to present lists of artists born after a given date. The content of an instance *Artist_after*[*d*] of this node is obtained as follows:

- select all the tuples *t* of table *artist* that satisfy *t.birthdate* >= *d*
- for each tuple *t*, generate the character string made of the tag ; the value of *t.name*, the text "(born ", *t.birthdate*, the text ") ", and the closing tag .

The following figure shows an instance of this node computed on an actual database.

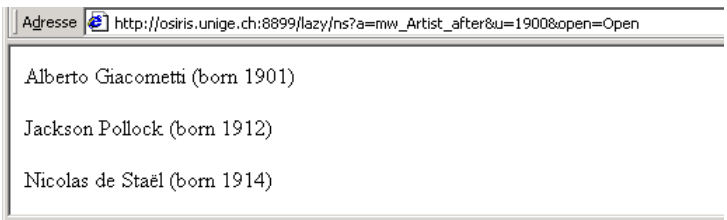


Figure 1. A node instance Artist_after[1900]

Elements enclosed within { and } are called tuple elements because they are repeated for each selected tuple. The tuple elements may be preceded or followed by non-tuple elements, they may also appear as sub-elements of non-tuple elements. Non-tuple elements are evaluated only once, they are intended to display constant or aggregate contents. They may not refer to attributes, except in arguments of aggregate functions like min(), max(), sum(), count(), etc.

3.3. Reference links

A reference link creates an active element whose action (when activated by a mouse click) consists in jumping to (opening) the referred link. A link specification refers to a node through its identity, which is composed of its schema name together with actual parameter values. The source anchor of a link can be any element or list of elements. A reference link is specified with the following syntax :

href *node_schema_name* [*actual_parameter_list*] (*element*, ...)

The following schema contains a reference link (works) to a node *Works_by[ano]* that is intended to display the list of works of this artist. It also shows a non-tuple element used to display a header and a footer.

```
node Artists_after_2[date]
  <h1>("Artists born after ", date) ,
    {
       <p>( href Works_by[ano] (name), "(born ", birthdate, ") " )
    } ,
 <p>(count(ano), " artist(s) selected")
from artist
     selected by birthdate >= date
     order by name
```

The Works_by schema is in turn defined as follows:

```
node Works_by[artist]
  {
     <p>(<img src=picture>()),
     <p>(<b>(title), " ", c_date, <br>(),
          support," ", height," x ",width
        )
  }
  from work
  selected by author = artist
```

Each selected work will be displayed as two paragraphs (<p>), the first one showing an image of the work and the second one giving textual information (title, creation date, etc.). Note that the *picture* attribute contains the URL of a picture and that it is used as an HTML attribute value. Figure 2 shows the content of an instance Artist_after[1875] and an instance Works_by[…] that can be reached by following the link labeled *Kasimir Malevich*.

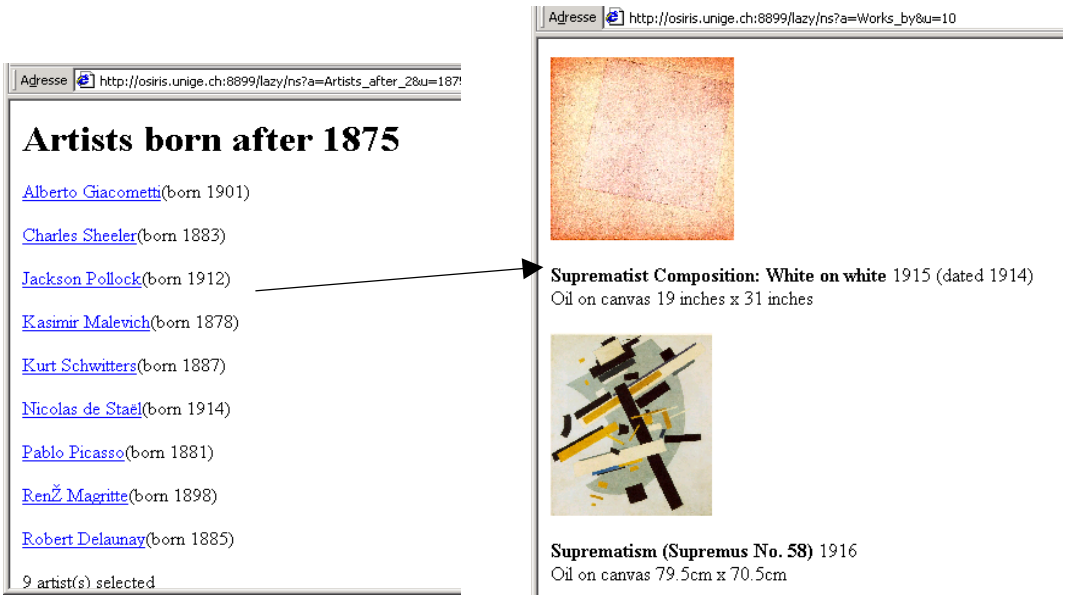


Figure 2. Two node instances with reference links.

3.4. Inclusion (translusion) links

An inclusion link creates a compound-component relationship between two nodes. The content of the included node is a part of the content of the parent node. With inclusion links one can construct arbitrarily complex nodes, for instance to represent complex structured documents. Figure 3 shows an instance of the following node schema, that includes three other node instances (Countries, Work_list, and Contemporary (defined in section 4.2):

```
node Artist_ext[id]
   {    <p>("id = " , id) ,
        <h2>( name, " (", birthdate, "-", deathdate, ") "),
        include Countries[id],
        <h4>("Some works"),
        <blockquote>(include Work_list[id]),
        <h4>("Contemporary with: "),
        <blockquote>(include Contemporary[birthdate, deathdate])
   }
   from artist
   selected by ano=id
```

Figure 3. A node instance that includes other node instances .

Since included nodes can themselves include other nodes, it is possible to create complex hierarchical documents through inclusions. Indeed, the inclusion mechanism is very powerful and can serve many purposes such as: computing outer-joins; constructing complex hierarchical contents; reusing node schemas (for instance standard headers, footers, side bars, etc.); computing paths in a graph ; or computing transitive closures.

The inclusion mechanism corresponds to what Nelson calls “transclusion” (Nelson 1995).

3.5. Expand in place links

An expand-in-place link is an inclusion link that defers the inclusion until the user activates the link. The content of a node with expand-in-place links will thus depend on user actions taken so far. For instance, the link labeled [open image] in node schema Work_list is an expand-in-place link of the form

```
expand href Work_image[wno] ("[open image]")
```

Clicking on it will display the content of the Work_image instance corresponding to that work number. The following figure shows a node with one such link open.

Expand-in-place links can of course be nested in “expanded” nodes to create box opening and closing effects.

3.6. The development process

The Lazy system is composed of

- a node compiler that checks the syntax of node schema definitions; translates them into database queries (SQL statements with parameter placeholders); and stores the compiled definitions in a *nodes* table of the data dictionary.
- a node server (a Java servlet) that receives node requests from clients (browsers); loads the appropriate node definitions; executes database queries to build the node contents; and sends the resulting Web pages to the clients.

The node server also processes inclusion links. When a node contains an inclusion link (signaled by a special tag), the content of the node to include is computed and inserted at the marked place. This process is carried out recursively until all inclusions are resolved. The node server treats expand-in-place requests by recalling the previously generated node and processing the requested inclusion.

The Web site development cycle consists in writing or editing source files that contain node schemas; compiling the definitions; viewing (testing) the newly defined nodes in a Web browser. Since the system is dynamic, once a node definition has been modified and recompiled, all the subsequent instances are generated according to this new version of the schema. In addition, node schemas can be compiled independently (i.e. there is no global site generation phase). This makes the system well suited to incremental development and prototyping. Every design choice can be immediately implemented and tested. Since every generated Web page is an instance of a node schema it is easy to locate the origin of a problem and fix it. This is not the case with a

single large server program which generates several or all the page types, and where fixing a problem necessitates to dig in the program to find the faulty piece of code.

4. The Design of Hypertext Views

Although declarative approaches to hypertext view generation make the hypertextual structure of the site more explicit, they do not automatically entail a good hypertext design. It is therefore necessary to apply hypertext design principles to obtain efficient hypertext views.

4.1. Hypertext design

Hypertext design is a complex problem that has generated a lot of research (see [ACM95] for instance) that lead, among others, to the identification of design issues, the study of the design process, the definition of design methodologies and principles, the definition of design patterns, and the development of hypertext design environment.

One of the recurring issues in hypertext design is the (dis)orientation problem. It has been described by Conklin as the problem of knowing “where you are in the network and how to get to a place that you know (or think) exists in the network” [Conklin87]. In the field of literary hypertexts Landow [Landow97] indicates that hypertext design techniques must answer four main questions:

- How to orient the readers and help them read efficiently and with pleasure?
- How can one help the readers retrace the steps in their reading path?
- How can one inform those reading a document where the links in that document lead?
- How can one assist readers who have just entered a new document to feel at home here?

One can see that orientation and access structures such as indexes, folder hierarchies, navigation histories, site maps, home pages, etc. are indeed intended to address these questions.

Another important issue is the typing of nodes and links (Nanard, Nanard, 1995): “typed nodes and links enable the user to recognize familiar landscapes within the diversity”. When creating large hyperspaces it becomes important to classify the nodes according to their contents. The same is true for links, the type of which indicate the meaning of the relationship they materialize.

Nanard and Nanard's study of the design process shows that designers act incrementally and have an opportunistic behaviour, reasoning at the abstract level and at the instance level. Thus the hypertext designers need prototyping tools to have the necessary experimental feedback.

4.2. Structured hypertext design methodologies

Structured hypertext design methodologies generally consist of successive steps such as conceptual design, navigational design, abstract interface design, and interface implementation. The conceptual design step is similar to database conceptual design, its main concern is to capture the domain semantics. This step produces a conceptual model which can be a class diagram in OOHDM (Schwabe, Rossi and Barbosa, 1996) or an entity-relatiohship diagram in RMM (Isakowitz, 1995). Apart from usual diagramming tools, this phase can be supported by a hypertext system, such as MacWeb, that allow the designer to represent concepts and instances and provide concept elicitation tools such as structure cloning, instantiation, generalization.
The navigational design is more specific to hypertext design, it determines what information will be presented to users in the hypertext's nodes, and how these nodes will be interrelated through hypertext links. Starting from the conceptual model, the aim is to define all the necessary types of hypertext nodes and an efficient linking structure on these node types. A node type is specified by the type of entities it will present, some selection predicate, and a list of attributes to include in the node content. In fact, these nodes can be considered as views on the conceptual schema (OOHDM) intended to select the information that are relevant for the hypertext user. The linking structure defines what access primitives interconnect the nodes. The access primitives may be unidirectional or bidirectional links, groups of links, indexes, guided tours, etc. For instance, a node presenting a university department could be connected to nodes representing its faculty members through an index node that points to each faculty member node. Alternatively, the faculty member nodes could be organized in a guided tour form.
These methodologies have been criticized by several authors. As mentionned by Bernstein in (Bernstein, 1998) "rigid structure is often promoted for its efficiency and cost-effectiveness, particularly for large Web sites, but excessive rigidity can be costly". In particuler, the repetitive appearance of links leading to central points (home pages) may attract the users to these points, leaving many pages unvisited

4.3. Database Design vs. Hypertext Design

Recently, methodologies have been developed to design hypertext views of databases (Atzeni, Mecca, and Merialdo, 1998) (Fraternali and Paolini,1998) (Fernandez et al. 1998). In the case of hypertext views on databases, the design can take advantage of the data semantics expressed in the database schema. But, as mentioned in (Atzeni, Mecca, and Merialdo, 1998), the distance between the database schema and the hypertext structure is great. For instance, a good database structure should minimize data redundancy to avoid update anomalies. On the contrary,

redundancy may help the hypertext user by reducing the number of navigation steps to reach some information. In database design, the length of logical access paths is not important since the database is generally accessed through application programs or high level interfaces (e.g. forms).In hypertexts, since the basic action is the navigation step, the number of links to be followed is an important criterion to determine the usability of the system. Similarly, conceptual entities may have been decomposed, for normalization purpose. To recompose them would imply several navigation steps whereas a compact presentation, in a single node, is cognitively much more efficient. For these reasons, a database schema can only serve as a starting point to produce a first version of the hypertext navigation model. This model must then be refined and transformed to enhance its efficiency.

5. Designing hypertext views with Lazy

The design method we propose proceeds in two phases. The first phase is similar to structured hypertext design methodologies such as RMM (Isakowitz, 1995) and HDM (Garzotto, 1993). It takes the database schema as a starting point. The second phase consists in applying various refinement operations to incrementally enhance the hyperspace. It is based on experimental feedback obtained by rapidly developing prototypes and testing them. This phase is intended to solve navigability and orientation problems while making the hyperspace less regular and thus more attractive.

Although the presentation is based on the Lazy language and system, this method can be adapted to other hypertext view systems (declarative or procedural).

5.1. Initial structured phase

Initial hypertext structure from the database schema

The initial structure is made of one node schema for each database relation. For a relation schema **R**(A_1, …, A_r) with primary key attributes K_1, …, K_n , the corresponding node schema is

node R[p_1, ..., p_n]

{ A_1, …, A_k }

from R selected by K_1= p_1 and … and K_n = p_n

If a group of attributes F_1, …, F_m forms a foreign key of another relation **S**, R must contain a reference link href S[F_1, …, F_m] (…).

An instance of this schema represents a tuple of **R** with links pointing to tuples related through a foreign key.

Example. The initial node schema corresponding to the relation work(wno, title, c_date, author, …) is

```
node Work[w]
{ wno, title, c_date, href Artist[author](author), ...}
from work selected by wno = w
```

This structure represents accurately the contents of the database, i.e. the graph of all node instances and possible links is isomorphic to the graph of the database objects connected through the reference attributes. However, this structure is not really navigable because the links are unidirectional. In our previous example, it is possible to navigate from a Work node to the corresponding Artist node, but the converse is not true. At this stage, the association represented by a foreign key is navigable along the n-to-1 direction only.

Reverse links and index nodes

To implement the 1-to-n direction of an association based on a foreign key it is necessary to define index nodes. If relation **R** has a foreign key F that refers to **S**, the index node is defined as follows:

```
node R_by_F [f]
 { href R[K] (…) }
from R selected by F = f
```

where K is a key of **R**. The node schema corresponding to S must then be completed with a link of the form href R_by_F[L](…) where L is a key of **S**. For instance, the index node corresponding to the author relationship between Work and Artist is

```
node Work_by_author[a]
 { href Work[wno] (title) }
from work selected by author = a
```

This index node plays the same role as selection menus in systems that support links with multiple ends. Note that the initial structure together with reverse links yields a fully navigable view of the database. This means that if two tuples t1 and t2 are related (directly or indirectly) in the database through foreign keys, there exists some path in the hypertext to go from the representation of t1 to the representation of t2, and vice versa.

Entry points

Entry point nodes are nodes that are intended to help the user enter the hyperspace or to re-orient him or her during the navigation.

An index structure is a set of nodes that allows, by successive selections starting from a root node, to reach a particular node. A simple particular case is the creation of an index on an attribute A. It requires the creation of two node schemas: 1) a root node presenting all the possible values of A; 2) a node presenting a list of all the objects having the same value for attribute A. One can generalize this structure to create indices on several levels where each level corresponds to a different attribute. The traversal from the root downwards amounts to fixing an attribute value at each stage.

5.2. Refinement phase

The following refinement operations are intended to improve the navigability (or the legibility) of the hypertext view.. These operations are independent of each other and there is no fixed order of application. The idea is to produce a new prototype by applying one or more operations, then to test it and collect the experimental feedback to incorporate it in the next prototyping cycle. As in the first phase, some operations aim at remediating some limitations of the Web hypertext model and technology.

Link composition (shortcuts),

One way to reduce the number of navigation steps in the hypertext view is to create "shortcut" links. This consists in composing two (or more) links into a new one. When two nodes are defined on the same relation R this operation is straightforward. If a node N1 has a reference link to a node N2 and N2 has a link to a node N3, this last link can be directly incorporated in N1. This is particularly useful to increase the navigability of the initial structure which contains index nodes of the form

```
node R_from _X [...] // index node
   ... href R[K]
from R...
```

where K is a key of R, In this case any link in R can be incorporated into R_from_X to suppress a navigation step.If the two nodes N1 and N2 are defined on different relation then the definition of N1 must be changed before incorporating a link of N2 into N1.

Inclusions

This operation consists in changing the nature of a reference link into an inclusion link. It allows, for example, to represent complex entities in the form of only one node including sub-nodes. This operation is particularly interesting when the link has a semantics of the type "part-of "or

"compound-of". It is also a way of reducing the number of reference links in the hypertext and thus to shorten navigation paths.

Summarization

When a node represents a large object having many attributes, it may be desirable to derive "summarized" node by removing certain attributes of the initial definition. This summarized node will have a link towards the complete node. It should also be decided for each link which leads to the initial node if it is necessary to "redirect" it towards the summarized node.

Adding computed links

The database schema usually represents relationships between entities through foreign key constraints (or referential constaints). However, some interesting relationships are not represented directly in the database schema. For instance, the relation "contemporary" between artists is not represented but it can be computed since we know the birth and death dates of the artists. Links cooresponding to such derived relationships can be created in the hypertext schema using diverse schemas. For instance, a relationship "contemporary" between artists can be implemented by creating a node

```
node Contemporary[abirth, adeath]
  item name, href Artist[ano]
from artist
selectedy by deathdate > abirth+15 or bdeath > birthdate+15
```

and adding a new link in Artist

```
node Artist[id]
  items ...
    href Contemporary[birthdate, deathdate] ("contemporary artists")
  ...
```

Widening

The widening of a node consists in weakening its selection condition. As a consequence, other objects will be shown in the node. This is a way to contextualize information by presenting it together with related information. For example, a painting could be presented together with other paintings of the same period or of the same region.

Previewing

Previewing makes it possible to see part of the contents of a referred node without having to traverse the link. The objective is to avoid navigation towards a node whose contents do not correspond to the information the user is looking for. This operation consists in creating a summarized node (in general with only a few attributes), as in the derivation operation, and to complement the initial reference link with the inclusion of the summarized node.

5.3. Impact on database design

Certain things cannot be done with hypertext views only. It is necessary to add extra information in the database. For instance, to define a presentation order that cannot be inferred from the tuple values, or to create a visiting sequence of nodes.

In fact one can observe that real database schema often contain attributes, and even tables, that serve only presentation purposes. One could argue that presentation and ground information should be clearly separated. However, the presentation information has to be stored somewhere and if it is subject to frequent changes, it should better be in the database. A good design practice could be to store them in separate tables rather than as attributes of other tables.

6. The Analysis of hypertext view schemas

When a hypertext view schema becomes very large, it gets more difficult to test it on prototypes because the number of possible navigation paths grows much faster than the number of nodes and links. Similarly, the potential number of different node instances can be enormous. It is therefore important to have tools that help the designer to make design decision on the sole basis of the view schema. The aim of schema analysis is to find opportunities to apply refinement operations.

Node schemas can be represented as graphs and analyzed to determine the properties of the generated hyperspace and see the effect of refinement operations. Several kinds of analyses can be carried out during the design process: identify nodes that should be linked to improve the navigability of the hypertext, (e.g. define “guided tours”); check whether semantically related information are well connected with short paths); check the reachability of nodes (may be prove the (non) connectedness of the hypertext view); analyze cycles; etc.

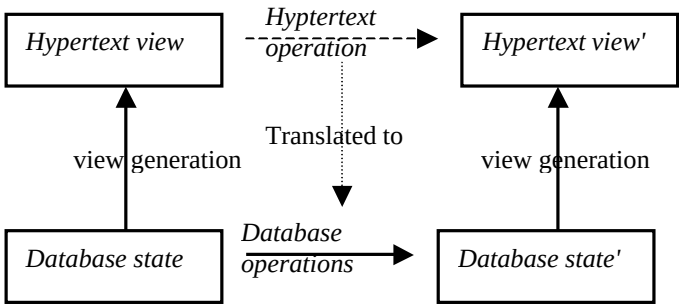
To analyze hypertext structures and see the effect of refinement operations, it is convenient to have a compact graphical representation (Falquet, Guyot, Nerima, 2001). The analysis is done on the node schema which is much smaller than the generated hypertext. The analysis process includes the following actions:

(1) Identify links that do not exist in the initial structure but might be helpful.
(2) Check the number of navigation steps between the nodes and see the semantic closeness (or, semantic distance). If they are semantically close, the number of navigation steps should be reasonably small.
(3) Explore different types of links between the nodes and select the best one; for example, it might be necessary to change "reference" to "include" or "expand in place".
(4) Check whether a node is reachable from a given node (i.e., accessibility analysis). Notice that the schema connectivity does not ensure that the hypertext itself (the node and link instances) is fully navigable. This depends on how objects are interrelated in the database. However, knowing properties of the links (like cardinalities), it is possible to prove the full connectivity of the hypertext view.
(5) Compute the maximum number of steps that a user can follow from a node to another node. Since the user does not see the entire hypertext, it is not obvious for the user to know whether the path is the shortest one from a node to another node. The information about the longest steps that one can take to reach a destination node can give some information about how a user might get lost in the hyperspace.

7. Making the Hypertext Views Active

7.1. The concept of active view

A fully functional hypertextual interface should enable the user to act on the underlying database through the hypertext view. This means that
1) typical hypertextual operations like modifying a node contents or linking and unlinking nodes, should be available;
2) it should be possible to trigger database procedures by acting on some (active) hypertext object.
Since views are virtual (computed), the only way to update a view is to update the underlying database and then recomputed the view, i.e. to translate any view update operations into database update operations, as shown on the following diagram



However, it is well known from the database domain, that translating view operations into database operations cannot be done univoquely. For a given view operation there can be several possible translations leading to different database states that all generate the same view (i.e. *Database state'* is not unique). This is called the view update problem in databases, and the same occurs for hypertext views.

In general, it is not possible to translate view operations into database operations because of ambiguities, so other auxiliary mechanism must be used. In Lazy we define active views by associating actions to reference links (active links). An active link is a reference link (href) that triggers a database action when traversed. In addition to usual elements, the source anchor of an active link can have one or more input elements and must have one action element. The general syntax of an active link is

active href *node-name*[*parameters*] (*standard-or-input-or-action-elements*)

An action element takes one of the following forms:

on "button-label" **do insert** *table-name*

on "button-label" **do delete** *table-name*[*key-attributes*]

on "button-label" **do update** *table-name*[*key-attributes*]

and an input element is

set *attribute-name = expression* | **textfield**(*width*) | **textarea**(*nb-lines, width*) | …

Attribute-name must refer to an attribute of the table mentioned in the action element.

Example. The following node enables the user to add a work *w* to an exhibition *e* and to insert a comment.

```
node mw_add_work_exh[e, w]
   <h3>("Add " , work.title , " to " , exhibition.title) ,
   active href mw_Exhibition[e] (
      set work = w , set exhibition = e ,
      <p>("Comment: " , set org_comment = textarea(10, 30)) ,
      on "Add" do insert ex_content
   )
```

```
from exhibition, work
  selected by exno = e and wno = w
```

Once equipped with such a mechanism, hypertext views become active and can serve as a basis for prototyping full Web applications

8. Making the Hypertext Views Adaptive

Adaptiveness in hypermedia systems consists mainly in taking into account the user's profile when deciding on what information to display, how to display it, and how to react to user actions. We distinguish two kinds of adaptiveness: the first one is based on some stored information about users while the second one is based on dynamic information, namely the navigation history of the user.

8.1. Adaptiveness and stored information

In the context of hypertext views, this type of adaptiveness means that the content of a node, and its links, should be generated according to a stored user profile. It can be implemented in a straightforward way, provided

- some "profile" relation contains suitable information about the user profiles,
- the system has a global variable USER that stores the user name (in the current implementation it is represented by a supplementary parameter in each node schema).

For instance, the following node schema displays information about a particular work of art. It includes a Details nodes that will present more detailed information but only if the user profile has detail_level > 2.

```
node Work[n]
  items title, c_date, include Details[n], ...
from work selected by wno = n

node Details[n]
  items width, height, acquired, ...
from work, profile
selected by wno = n and
   profile.user = USER and profile.detail_level > 2
```

If the user has detail_level = 2 the selection condition will be false for every tuple and thus the node will remain empty. With the inclusion mechanism it is thus possible to create contents and links that depend on user profiles or on other contextual information such as time, date, etc. (for

instance, forthcoming exhibitions could be announced in given nodes during the weeks before their opening).

8.2. Adaptiveness and the navigation history

Path-awareness is a form of adaptiveness, which consists in having node contents that depend on the user navigation path. Current Web browsers offer a limited path awareness feature which consists in displaying anchors of already visited nodes in a particular color. Although very simple, this mechanism proves efficient when exploring a new site. In fact, we can think of many situations in which we would like to have the content of nodes depend on previously visited nodes. For instance, we could have an anchor "latest news" in the heading of every node, as long as the "News" node has not been visited. Once is has, this anchor should disappear from all the nodes. In order to implement that type of behavior it is sufficient to systematically use a HISTORY parameter to memorize and transmit the navigation history.

9. Conclusion

We presented an declarative approach to specify web sites in the form of hypertext views on databases. Since the used language is non procedural and explicitly shows the structure of the generated hyperspace, it is well suited for an iterative design process. The existence of a hypertext schema makes it possible to check properties of the hypertext, such as path lengths or accessibility, without accessing the hypertext nodes themselves. The development process we propose consists in starting from an initial design and then entering an analysis-refinement cycle. The structural analysis of the hyperspace uses the graph formalism while the refinement is based on several basic operations.

Updating databases through hypertext views can be done through active reference links. It takes advantage of the navigational nature of such views, in particular to disambiguate operations which create or delete relationships between objects.

10. References

Communications of the ACM, Special section: Hypermedia Design. (1995). CACM, Vol. 38, No. 8.

Anderson, C., Lavy, A., Weld, D. (1999). Declarative web-site management with Tiramisu. In Proceedings of WebDB’99 Conference, Philadelphia.

Atzeni, P., Mecca, G., & Merialdo P. (1998). Design and Maintenance of Data-Intensive Web Sites. In Proceedings of the EDBT’98 Conference, Valencia (pp. 436-450).

Bernstein, M. (1998). Hypertext Gardens: Delightful Vistas. Eastgate Systems, Inc. http://www.eastgate.com/garden/.

Conklin, J. (1987). Hypertext: An Introduction and Survey. IEEE Computer 20 (1987): 17-41.

Ceri, S., Fraternali, P., Paraboschi, S. (1998). The IDEA Web Lab. In Proceedings of SIGMOD Conference '98.

Cluet, S., Simeon, J. (1998). Using YAT to Build a Web Server. In A. Mendelzon & P. Atzeni (Eds) The Web and Databases, Selected papers from WebDB'98 (pp 136-151). Berlin, Springer Verlag (LNCS 1590).

Dar, S., Entin, G., Geva, S., Palmon, E. (1998). DTL's DataSpot: Database Exploration as Easy as Browsing the Web, In proceedings Sigmod'98 Seattle.

EDBT (1998). Proceedings of the international conference on Extending Database Technology (EDBT'98), Valencia. Springer Verlag, (LNCS 1377).

Falquet, G., Guyot, J., & Nerima L. (1998). Language and tools to specify hypertext views on databases. In A. Mendelzon & P. Atzeni (Eds) The Web and Databases, Selected papers from WebDB'98 (pp 136-151). Berlin, Springer Verlag (LNCS 1590).

Falquet, G., Guyot, J., Nerima L & Park, S. (2001). Design and Analysis of Virtual Museums. In Proc of the Museum and the Web Conf. (MW2001), Seattle.

Fernandez, M., Florescu, D., Kang, J., Levy A., Suciu, D. (1998). Catching the boat with Strudel: experience with a web-site management system, In Proceedings of SIGMOD'98 Conference.

Florescu, D., Levy, A., Mendelzon, A. Database Techniques for the World-Wide Web: A Survey. ACM SIGMOD Record, Vol. 27, No. 3, Sept. 1998.

P. Fraternali, P. Paolini. (1998). A Conceptual Model and a Tool Environment for Developping More Scalable, Dynamic, and Customizable Web Applications. In Proc. of the EDBT'98 Conf., Valencia, 421-435,.

Garzotto, F., Paolini P., & Schwabe, D. (1993). HDM--a model-based approach to hypertext application design. *ACM Trans on Information Systems*, Vol. 11, 1 26.

Isakowitz, T., Stohr, A., & Balasubramanian, P. (1995). RMM: a methodology for structured hypermedia design. *Communications of the ACM 38, 9)*, 34 -- 44.

Lawrence, S., Giles, C. (1998). Searching the world wide web. Science, 280-4, pp 98-100,.

Mecca, G., Merialdo, P. Atzeni, P. Crescenzi, V. (1999). The (Short) Aranaeus Guide to Web-SiteDevelopment. In Proceedings of WebDB'99 Conference, Philadelphia.

Mendelzon, A. and Atzeni, P. (1998). The Web and Databases, Selected papers from WebDB'98, Springer Verlag, Berlin (LNCS 1590).

J. Nanard, M. Nanard. (1993)."Shoud Anchors Be Typed Too? An Experiment with MacWeb". In Proc. of the Hypertext'93 Conf., Seattle, 51-62, 1993

Nanard, M., Nanard, J. (1995). Hypertext design environments and the hypertext design process. CACM, Vol. 38, No. 8.

Nelson, T. (1995). The Heart of Connection: Hypermedia Unified by Transclusion. CACM, Vol. 38, No. 8, August.

Schwabe, G. Rossi and S. Barbosa. (1996). Systematic Hypermedia Design with OOHDM. Proceedings of the ACM International Conference on Hypertext (Hypertext'96), Washington.

Toyama, M., Nagafuji, T. (1998). Dynamic and structured presentation of database contents on the web. In Proc. of the EDBT Conference, Valencia.