Design and Analysis of Virtual Museums

Falquet, Gilles; Nerima, Luka; Guyot, Jacques

A&MI
Archives & Museum Informatics
158 Lee Avenue
Toronto, Ontario
M4E 2P3 Canada
info@ archimuse.com
www.archimuse.com

Published: March 15, 2001.

PAPERS
Museums and the Web 2001

Design And Analysis Of Virtual Museums

Gilles Falquet, Jacques Guyot, Luka Nerima, University of Geneva, Switzerland and Seongbin Park, Information Sciences Institute, USA

Abstract

Using the same data, which could come from local databases or external sources such as the Web, virtual museum designers can build different hyperspaces. It is possible that visitors would find some of them more useful than others. Therefore, virtual museums designers should be equipped with a tool by which various hyperspaces for virtual museums can be easily designed and examined.

In this paper, we view a virtual museum as a hypertext that consists of nodes and links and show that a database publishing tool called Lazy, which generates a hypertext view (i.e., derived hypertext) of a given database, can be used for designing virtual museums. The Lazy system consists of a declarative hypertext view specification language, a node schema compiler, and a node server that processes node requests. Since the language is purely declarative, it is fairly easy to construct and revise hyperspaces for a virtual museum. With this tool it becomes possible to adopt an iterative design methodology.

Given a database for a virtual museum, we first construct a hypertext using the procedure (Falquet & al., 1999) called an initial structure. We then proceed to analyze the initial structure and examine possible refinement operations that can enhance the usability of the created hypertext. For that purpose, we use a simple graph-based analysis and we show kinds of analysis that can be done using the graph-based approach. Once the structure is refined using the refinement operations, we apply grammar-based formalism (Park, 1998) to the refined structure to see whether we can obtain a simpler grammar that can generate the same hyperspace. Our goal is to explore various analysis techniques on the hypertext and give insights into designing a good hyperspace using the analysis results.

Introduction

In this paper, we consider the issues that arise in the design and implementation of virtual museums. A virtual museum is defined as "a logically related collection of elements composed in a variety of media, and, because of its capacity to provide connectedness and the various points of access available, [it] lends itself to transcending traditional methods of communicating with the user; it has no real place or space, and dissemination of its contents are theoretically unbounded" (Andrews, 1996). According to this definition, it is reasonable to view the structure of a

virtual museum as a hypertext since each element in a hypertext can be connected to others. However, it is well know that large hypertexts, in particular large Web sites, are very difficult to manage. Thus it has become common practice to store data in a database and to use some mechanism to automatically produce the hypertext.

In this paper, we will study the construction of virtual museums on top of databases. The content of a database can represent either real-world entities (e.g. existing works of art, or existing museums) or virtual entities (e.g. virtual exhibitions that exist only on the Web). When constructing a virtual museum, it is important to provide visitors to the virtual museum with a well-designed hyperspace so that they do not get lost while navigating inside the space. Although "lost in hyperspace" is a well-known symptom (Conklin, 1987), in our case, we have an underlying database that is structured by a database schema and we can avoid the symptom to some extent.

Given database contents, different hypertext structures can be created for the virtual museum, and visitors will find some structures more accessible than others. Therefore, the structure should be carefully designed, and it is important for virtual museum designers to easily define nodes and links that form hyperspaces and see what the results look like (examine different hyperspaces freely). In order to help designers who build virtual museums, we present a database publishing tool called Lazy (Falquet & al., 1998), by which different hyperspaces can be easily constructed. Using the Lazy system, designers can construct the web site for a virtual museum, and once the web site is created, they can analyze the structure of the virtual museum.

Some of the related works are as follows: a visual grammar-based formalism was introduced in order to analyze hypertext structures in (Costagliola & al., 2000), and an implementation in Prolog for analyzing hypertext that contains conditional linkage was sketched in [HT99 paper]. Recently, database publishing has attracted interest and some of the techniques are described in (Entin & al., 1998; Toyama & al., 1998). There are several approaches to create hypertexts from databases. In the procedural approach, the hyperspace designer must write programs (CGI programs in C or Perl, Java servlets, PHP scripts, etc.). These programs are generally large since the code must contain both tags and programming constructs. They are therefore difficult to read, and the hyperspace structure is hidden by the programming constructs. For these reasons, such a code is tough to maintain and update. The dynamic document approach consists in extending some document mark-up language (such as HTML) with specific tags for database querying, result processing and formatting, etc. See, for instance, Cold Fusion

(www.allaire.com). These tags introduce procedural parts into the document description. The declarative approach (Fernandez & al., 1998), (Atzeni & al., 1998), (Falquet & al., 1998) consists in specifying a hypertext structure and specifying how to build the hypertext elements from the database content. It is conceptually simple and tends to be closer to the information designer's conceptual level. Our work is also related to hypertext design methodologies such as RMM (Isakowitz, 1995) and HDM (Garzotto, 1993). It also takes into account adaptive features such as content adaptation, a well-known adaptation technique in the field of adaptive hypermedia (De Bra, 1998).

The paper is organized as follows: in the next section, we will introduce the concept of hypertext view with examples constructed with Lazy. Then, we propose a design process for hypertext views and show its application in the design of a virtual museum. We proceed to explain possible analysis on the hypertext. Finally, we conclude the paper with discussions about our approach for the design and analysis of a virtual museum.

Constructing Hypertext Views with Lazy

This section presents the Lazy hypertext view specification language. The presentation is rather informal and based on examples from the sphere of virtual museums.

A hypertext view is a set of nodes and links that represent (a part of) the contents of a database. In the declarative approach, the hypertext components (nodes and links) are derived from the database content (relation tuples) according to a hypertext view specification, as shown in Figure 1.

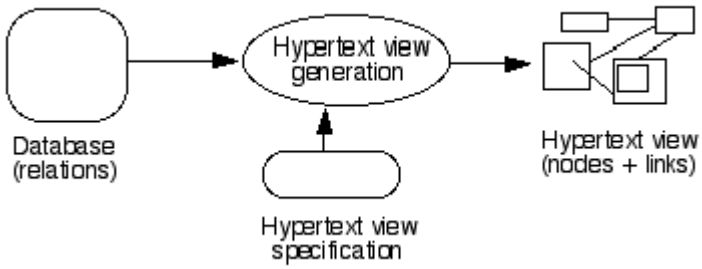


Fig.1: Generating hypertext views from a database and a hypertext view specification.

A hypertext view specification consists of a set of node schemas that specify the collection 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 node definition takes the following form:

node <node-name> [<parameters>]

pre <element-list>

items <element-list>

post <element-list>

from <collection>, ...

selected by <expression>

ordered by <expression>

An element of an element-list is an expression of the form, '<' type '>' '(' <element-list> ')' or a <simple-expression> . In the items part, a simple expression may involve literal constants (string, integer, etc.), attribute names, parameter names, operators and functions. An atomic element, specified by a simple expression, represents basic document data (CDATA in XML terms). In the pre and post part, attribute name may only appear within aggregate functions like min(), max(), sum(), etc.

Although the node specification language is generic enough to support relational as well as object-oriented databases, in this paper we will only consider the relational case since most existing databases are relational. In this case, the <collection> of a node is a relation (or the cartesian product of several relations) and each node item will represent a selected tuple of that relation.

Example:

Throughout this paper we will use the same database schema, shown in Figure 2, that represents a part of a virtual museum database. Note that this structure is close to a (simplified) real museum database schema. In fact, some of the relations (e.g. works) could be existing relations of a museum information systems, while others (e.g. exhibitions) could represent real as well as virtual objects.

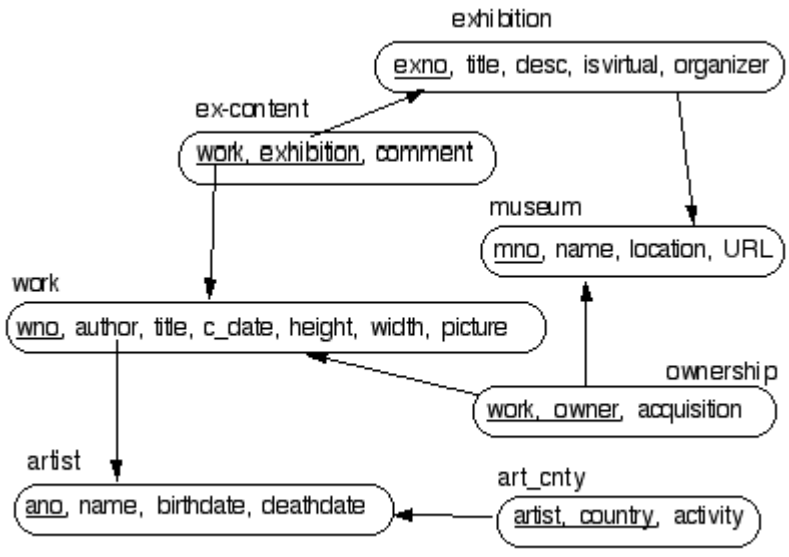


Fig. 2: Museum database schema

Consider the following node definition

```
node Artists_after[date]
 items
 <p>(name, "(born ", birthdate, ") ",
 href Works_by[ano] ("works")
 )
from artists
selected by birthdate >= date
order by name
```

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

- all the tuples t of table artist that satisfy t.birthdate >= d are selected
- an item is generated for each selected tuple, it contains an element of type <p> that is made of the artist's name, the text "(born ", the artist's birthdate, and the text ")".

This element also contains a reference link (works) to a node *works_by[ano]* which is intended to display the list of works of this artist. This node is defined as follows

```
node Works_by[artist]
 items
 <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 paragrahs (<p>), the first one showing an image of the work and the second one giving textual information (title, creation date, etc.). Figure 3 shows the content of an instance *Artist_after[1900]* and an instance *Works_by[...]* that can be reached by following an href link.

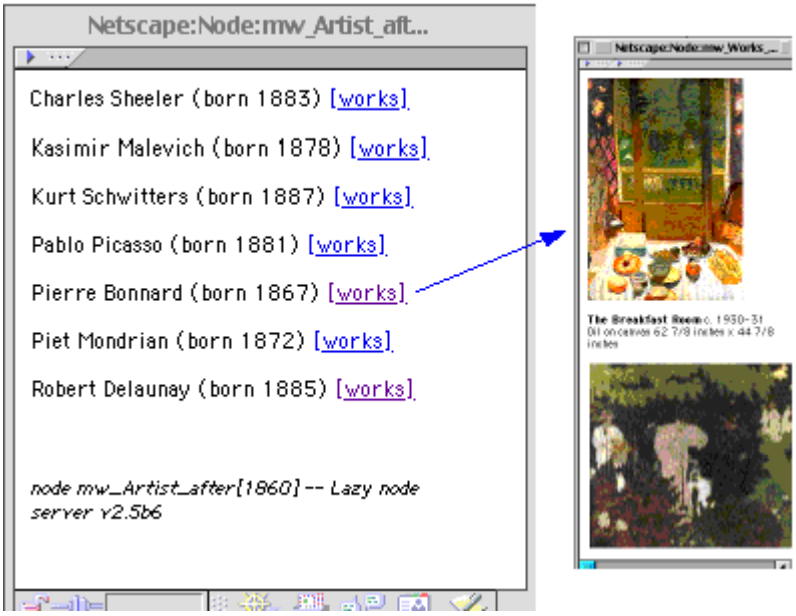


Fig. 3: Two node instance generated with the Lazy system.

The node definition language supports three kinds of links, which are "reference", "expand in place", and "include". 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 (schema name together with parameter values). 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 4 shows an instance of the following node schema that includes three other node instances (Countries, Work_list, and Contemporary (defined in section 3.2)):

```
node Artist_ext[id]
 items
 <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
```

Fig. 4: An node instance which includes other node instances .

Finally, 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.

The hypertext view generation system is composed of

- a node compiler that checks the syntax of node definitions and stores the node definitions (in a coded form) in 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 Web site development cycle consists in writing or editing source files that contain node schemas; compiling the definitions; and viewing (testing) the newly defined nodes in a Web browser. Since the system is dynamic, once a node definition has been modified and recompiled, the new version is immediately available to the clients (there is no site generation phase).

Every page that is viewed by a client is an instance of a node schema; thus any design problem can be readily located (as opposed to procedural approaches in which the same procedure may be used to manage several different Web pages).

Design

In this section, we explain how one can construct an initial hypertext structure that reflects the structure of a given

database and how that initial structure can be modified through refinement operations. We also present techniques that hypertext designers can use in order to implement an adaptive feature with the Lazy system.

Designing efficient and effective hyperspaces is a difficult task, probably because there are an extremely large number of paths that user can follow. It is thus difficult to ensure that the users will be able to reach any information node, that they 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. Since we are starting from an existing database, we already have a conceptual schema, declared by the relation schemes and the integrity constraints such as foreign key constraints. This schema shows the type of entities that are being considered and some semantic relationships (materialized by foreign key constraints) between these entities. However, a database schema is not sufficient to create good hyperspaces. Database design and hypertext design do not have the same objectives. If we rely on the database schema at the semantic level, it will be possible to create a hypertext structure that is efficient for reading and navigating. Our design method for hypertext views proceeds in two phases: 1) define a first hypertext structure based on the database schema; 2) refine this structure by applying various operations to the specifications of nodes and links.

Initial structure

For the construction of an initial structure, we assume that the database schema is given and fixed. One obtains the initial structure by defining a node schema for each relation of the data base. An instance of such a schema is intended to represent a single tuple of the relation. The node schema has a single parameter that represents an object of the class (in the relational case it is a set of parameters that forms a key value). The contents of the node items are formed of all the collection's attributes.

Links are formed by attributes or groups of attributes that refer to other relations (foreign keys). For instance, the initial node schema corresponding to the relation

relation work(wno, title, date, author, ...)

is

node Work[w]

items a wno, title, date, author, ...,

href Artist[author]

from work selected by wno = w

This structure accurately represents 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 completely navigable, i.e., due to the unidirectionality of links, it is not always possible to reach any node from any other node. Thus reverse links must be added.

For example, if a node N has a reference (href M[r]) to a node M, we add a link from M to N which corresponds to the traversal in the opposite direction. To carry out this operation, an intermediate node schema is defined, and this intermediate 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 objects o1 and o2 are connected (directly or indirectly) in the database, there exists some path in the hypertext to go from the representation of o1 to the representation of o2 (and vice versa).

Refinement operations

Refinement operations are intended to improve the navigability (or the legibility) of a given hypertext view. We list below some of these operations.

Link composition (short cuts)

One way to reduce the number of navigation steps in the hypertext view is to create "shortcut" links. This consists in combining two (or more) links into a new one. This is particularly useful to increase the navigability of the initial structure that typically contains nodes of the form:

```
node A [...]
 ... href B[key_attribute]
from T ...
node B [k]
 ... href C[x]
from T selected by key_attribute = k
```

where key_attribute is a key of T, In this case, any element of B (e.g. the link to C) can be incorporated into A to suppress a navigation step through B.

Inclusions

This operation consists in changing a reference link into an inclusion link. It allows us, 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 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 shortening navigation paths.

Summarization

When a node represents a large object having many attributes, it may be desirable to derive a "summarized" node by removing certain attributes of the initial definition. This summarized node will have a link to the complete node. It also should be decided for each link that 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 constraints). 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 corresponding to such derived relationships can be created in the hypertext schema using diverse schemes. 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 of 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 to a node whose contents do not correspond to the information we are seeking. This operation consists of creating a summarized node (in general with only a few attributes), as in the derivation operation, and adding to the initial reference link by including the summarized node.

Building indices and entry points

An index structure is a set of nodes that allows us, by successive selections starting from a root node, to reach a particular node. A simple, concrete, case is the creation of an index on an attribute A. This requires the creation of a 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.

Creation of linear paths

This operation creates links that make it possible to traverse all the node instances of a schema in a prescribed order (guided tour).

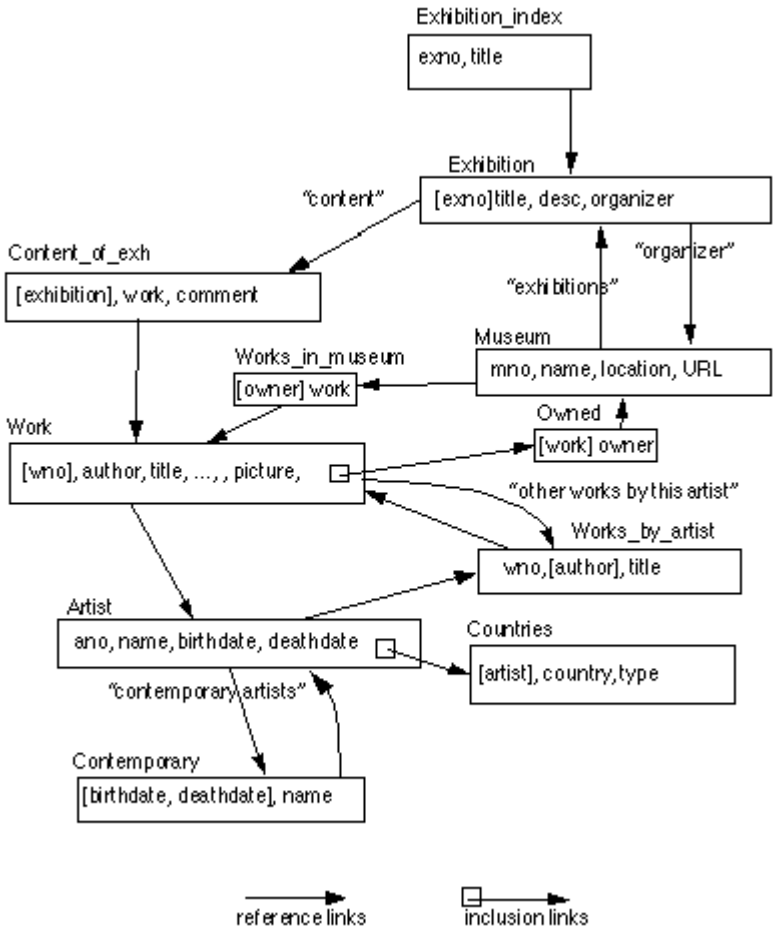


Fig. 5. A hypertext view structure after several refinement steps.

Designing Adaptive Nodes and Links

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

- some "profile" relation contains suitable information about the user profiles,
- a global variable USER exists 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 node 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 exhibits could be announced in some nodes during the weeks before their opening).

Path-awareness is another form of adaptiveness, and consists of having node contents that depend on the user navigation path. Current Web browsers offer a limited path awareness feature that consists of displaying anchors of previously 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 this type of behaviour, we need some

way to refer to the navigation history. The navigation history can be stored in a HISTORY parameter added to every node definition. Each node can then add its own identity to HISTORY and pass it forward to the nodes it refers to. The general schema is thus

```
node N[...parameters..., HISTORY]
 ...
 href AnotherNode[ ... parameters ..., HISTORY + "(N)"]
selected by ... conditions on attributes ...
AND condition on HISTORY
```

Analysis

Graph-based Approach

To analyze hypertext structures and see the effect of refinement operations, it is convenient to have a compact graphical representation. Our analysis is done on the node schema instead of node instances, and graphical representation is smaller than the generated hypertext (i.e., the set of node instances and links among them). Based on the analysis of the initial structure for the virtual museum, we can apply appropriate refinement operations.

Once we have a graphical representation of the node schemas, these are possible analyses that we can do on the initial structure.

(1) Identify links that do not exist in the initial structure, but might be helpful if we created them. Notice that those links that exist in the initial structure are directly come from the associations of data in the database (e.g., foreign key). After such links are identified, we can apply appropriate refinement operations to the initial structure.

(2) Check the number of navigation steps between the nodes and determine the semantic proximity (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 a better type than others; for example, it might be necessary to change "jump" 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 one node to another. Since the user does not see the entire hypertext, it is not obvious to the user whether a path is the shortest distance from one node to another. Information about the longest path that one can take to reach a destination node can inform us about how a user might get lost in the hyperspace.

Grammar-based Approach

After we determine the refined structure for a virtual museum, we can proceed to represent the node schemas in terms of grammar rules for a further analysis. We use grammars in two ways - one for the inner structure (i.e., the structure of a node) and the other for the outer structure (i.e., the structure of a set of nodes and links). Once node schemas are represented by a grammar, the following analyses can be done:

(1) Find a different grammar that is simpler than the original grammar, but generates the same hyperspace. The set of all virtual documents can be found first, and we can find another grammar that generates the same set.

(2) Investigate a property of the grammar, such as "inherently ambiguous", and determine the connection between that property and the navigational structure of the hypertext

a) Inner structure

The purpose of representing node contents with grammar rule is to obtain a compact representation of the node contents in terms of document structure and semantic content. For a node,

node N[p]

items item1, ..., itemk

the corresponding grammar rule (in BNF) is N ::= { s_1 ... s_k }, where s_i is (1) empty if $item_i$ is a constant (2) the attribute list a_1, ..., a_n if $item_i$ is an expression involving these attributes (3) the non terminal symbol M is $item_i$ is an inclusion of the form include M[...].

If we can prove that a node instance will always contain at most one item (this depends on the selection predicate), then the iteration indicators { } can be removed from the grammar rule. For example, the grammar rule corresponding to the node schema

node Artists_after[date]

items

<p>(name, "(born ", birthdate, ") ",

expand href Works_by[ano] ("works"),

include Biography[ano]

href ...

)

from artists

selected by birthdate >= date

ordered by ...

is Artists_after ::= { name birthdate [Works_by] Biography }.

The grammar of a node is the set of rules corresponding to this node and all the included and expanded nodes. For the above node this could yield (if nodes Works_by and Biography were so defined)

Artists_after ::= { name birthdate [Works_by] Biography }

Works_by ::= { title date support }

Biography ::= { country date activity }

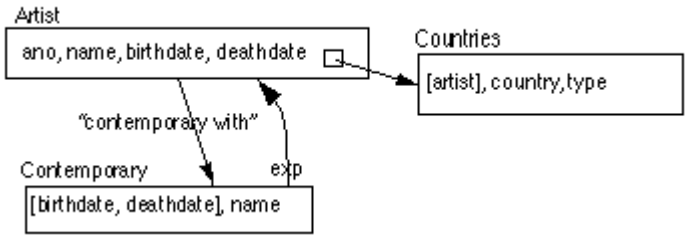
Since no rule in this grammar is recursive, this can be rewritten as a single rule:

Artists_after ::= { name birthdate [{ title date support }] { country date activity } }

In fact, this grammar shows the structure and semantics of a node. It is similar to a basic document type definition (DTD) for this node.

2) Outer structure

We explain the construction of grammar productions that describe the outer structure of a given hypertext using a simple example.



Example: figure 6

In this figure, we see that there is one href link, one inclusion link and one expand-in-place link. In order to describe the outer structure of this hypertext, we find possible paths from each node to others. We see that the Artist node has two

different types of links, one for inclusion (to Countries) and the other for href (to Contemporary). From Contemporary, there is one expand-in-place link to Artist. The node Countries does not have any link to other nodes.

Let the node Artist be described as two characters, a1d1, the node Contemporary as a2d2, and the node Countries as a2d3. (Each ai and di can be considered as "brackets" for each node.) If we attach prime symbols to each of those, that represents a node instance of the node schema. For example, a1'd1' is a node instance for the node a1d1. Then, each of the following products represents each link in the hypertext.

a1d1 -> a1'd1' | a2d2 | a1a3d3d1

a2d2 -> a2'd2' | a2a1d1d2

a3d3 -> a3'd3'

If we change a1d1, a2d2, and a3d3 as x, y, and z, respectively, and introduce a starting symbol S, we get the following products:

S -> x | y | z

x -> x' | y | a1z'd1

y -> y' | a2xd2

a1z'd1 -> x

a2x'd2 -> y

Notice that x', y', and z' is a1'd1', a2'd2', and a3'd3', respectively.

Once this first grammar is found, we can analyze the structure by examining the property of the grammar. We can also find a simpler grammar that represents the same hyperspace. One usage of this grammar formalism would be that one can use it as a site map for a given wWeb site so that users can get an idea of how the hyperspace is structured.

Conclusions and Future Directions

In this paper we presented a language to specify virtual museums in the form of hypertext views of databases. Since the 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 graph and grammar formalisms while the refinement is based on

several basic operations.

This development process is supported by software tools to compile the specifications and to dynamically generate Web pages (node instances) according to the specifications.

In the near future we plan to increase the adaptiveness capabilities of the generated hyperspaces, in particular the path-awareness. We are also starting experiments with a new version of the Lazy system to make the hypertext views active. This means that users will not only navigate in a virtual museum but will also act on this museum by updating the information nodes they see. For instance, users could create their own virtual exhibitions within the virtual museum, add annotations to objects, etc.

Acknowledgement

The authors would like to thank their colleagues Jean-Pierre Hurni and Claire-Lise Mottaz for their precious collaboration.

References

ACM (1995) Special section on hypermedia design. *Communications of the ACM* Vol. 38, No. 8.

Andrews, J. and Schweibenz, W. (1996). The Kress Study Collection Virtual Museum Project: A New Medium for Old Masters. Version 03-Dec-1996. http://www.arlisna.org/werner.html

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).

Conklin, J. (1987). Hypertext: An Introduction and Survey. *IEEE Computer*, Vol. 20, No.9, 17-42.

Costagliola, G., Dattolo, A., & Francese, R. (2000). A Visual Grammar-based Approach for the Analysis and Modeling of Hypermedia Structures. In Proc. *Multimedia Computing on the World Wide Web*, *2000* .

Dar, S., Entin, G., Geva, S., & Palmon E. (1998), DTL's DataSpot: Database Exploration as Easy as Browsing the Web, In *ACM SIGMOD 98 Proceedings*. ACM.

De Bra, P., Adaptive Hypermedia on the Web: Methods, techniques and applications, In *Proceedings of the AACE WebNet'98 Conference*, pp. 220-225, Orlando, Fl., 1998.

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., Vanoirbeek C., Rekik, Y. (1999). Des documents virtuels pour lire les bases de données. In M. Crampes & S. Ranwez (Eds) *Documents virtuels personnalisés, atelier de1a conférence francophone Interaction Homme-Machine*. Montpellier.

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

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.

Park, S. (1998). Structural properties of Hypertext. In K. Grønbaek, E. Mylonas, F. Shipman (Eds) *Hypertext'98*. Pittsburgh: ACM.

Toyama, M. (1998). SuperSQL: An Extended SQL for Database Publishing and Presentation, ACM SIGMOD 98.