Designing Web Information Systems

Daniel Schwabe

e-mail: schwabe@inf.puc-rio.br

Gustavo Rossi

LIFIA, Fac Cs. Exactas, UNLP, Argentina
CONICET and UNLM

e-mail: gustavo @sol.info.unlp.edu.ar

Alejandra Garrido

LIFIA, Fac Cs. Exactas, UNLP, Argentina
CONICET and UNLM

e-mail: garrido@sol.info.unlp.edu.ar

ABSTRACT:

This paper characterizes Web Informations Systems (WIS), idenfying distinguishing features and discussing their design requirements. Next, it presents the notion of Design Patterns, and shows how they can be applied to the design of WIS, drawing examples from sites in the WWW.

KEYWORDS: WWW, Hypermedia, Design Patterns

RESUMO:

Este artigo procura caracterizar o que são os Sistemas de Informação baseados na WWW. São identificadas as propriedades características de tais sistemas, e são analisados os requisitos de projeto para este tipo de aplicação. Em seguida, é introduzido o conceito de “Padrões de Projeto” (“Design Patterns”), que é então utilizado para o projeto destas aplicações. Ao longo do artigo, são utilizados exemplos reais encontrados na WWW.

Palavras Chave: WWW, Hipermidia, Padrões de Projeto

Este trabalho contou com o patrocínio parcial do CNPq.
This work was partially sponsored by CNPq.

Designing Web Information Systems

D. Schwabe (*), G. Rossi (**) and A. Garrido (***)

(*)Departamento de Informática. PUC-RIO, Brazil
E-mail: schwabe@inf.puc-rio.br
(**) (***) LIFIA, Fac Cs. Exactas, UNLP, Argentina
E-mail: [gustavo, garrido]@sol.info.unlp.edu.ar

(**) is also at CONICET and UNLM

1. Introduction

A working definition of a Web Information System can be as a set of WWW sites under the same administration, storing information to be used – created, accessed, and modified – by some identified community of users.

At least part of the phenomenal success of the WWW, and the increasing integration of "traditional" information systems into it is the recognition that it actually provides a richer communications channel between human beings that make up institutions, companies, schools, interest groups, or, more generally, society.

From its inception, the WWW was meant to be a way to help people access and use information, as stated by Tim Berners-Lee [Berners-Lee97] - "In fact the thing that drove me to do it (which is one of the frequently asked questions I get from the press or whoever) was partly that I needed something to organise myself. I needed to be able to keep track of things, and nothing out there, none of the computer programs that you could get, the spreadsheets and the databases, would really let you make this random association between absolutely anything and absolutely anything, you are always constrained. For example, if you have a person, they have several properties, and you could link them to a room of their office, and you could link them to a list of documents they have written, but that's it. You can't link them to the car database when you find out what car they own without taking two databases and joining them together and going into a lot of work. So I needed something like that."

A WIS can be seen then as an example of a "hybrid" system, a system conceived to be part of a man-machine team in solving a problem. This means that part of the task will be executed by the computer, and part by the human being. Since the human being will be performing part of the task, information must be presented to him in the most apropriate way - hence, multimedia and hypertext. This definition is quite flexible and can accomodate most existing WISs, since the boundary between the part performed by the computer and the part performed by the human being is movable – in one extreme one falls into traditional systems, where the computer does all the processing, and in the other extreme one falls into many current websites, where the computer just stores information and presents it to the human being, who then does the task.

"Universal access means that you put it on the Web and you can access it from anywhere; it doesn't matter what computer system you are running, it's independent

of where you are, what platform you are running, or what operating system you've bought and to have this unconstrained topology, which because hypertext is unconstrained it means you can map any existing structures, whether you happen to have trees of information or whatever. As people have found, it is very easy to make a service which will put information onto the Web which has already got some structure to it, which comes from some big database which you don't want to change, because hypertext is flexible, you can map that structure into it." (Tim Berners-Lee, [Berners-Lee97])

The WWW is based on the hypertext paradigm, inasmuch as it is composed of pages (in HTML) which can be linked to each other through URLs (links). Regardless of how a reader has reached a page, he will normally have the option of accessing the pages linked to the current page; by choosing a particular link, he will cause the page pointed to by the link to be exhibited; this process can repeats itself indefinetly. This succession of steps is know as "navigation", and is intrinsic to hypertext, and hence to the WWW.

This mode of access should be contrasted with more conventional database applications, where most accesses are achieved through queries – the user formulates a statement in some query language, describing the data he wishes to retrieve, and the system retrieves and displays the data. The user then may process this data in some fashion, and eventually issue another query to retrieve more information. In many cases, this sequence of steps is executed by an application program, not by a human being.

From the discussion above, we can intuit that there are distiguishing features in WISs that present new design requirements vis-a-vis traditional systems. In a broad sense, we can categorize them in three groups, which will be elaborated further later on.

The first group of design issues has to do with navigation, addressing issues such as

- What constitutes an "information unit" with respect to navigation?
- How does one establish what are the meaningful links between information units?
- Where does the user start navigation?
- How does one organize the navigation space, ie, establish the possible sequences of information units the user may navigate through?
- If we are adding a WWW interface to an existing system, how do we map the existing data objects onto "information units", and what relationships in the problem domain should be mapped onto links?

The second group of design issues has to do with the organization of the interface, addressing issues such as

- What will be the interface objects the user will perceive? How do these objects relate to the navigation objects?
- How will the interface behave as it is exercised by the user?
- How will navigation operations be distinguished from interface operations and from "data processing" (i.e., application operations)?
- How will the user be able to perceive his location in the navigation space?

The third group of design issues has to do with implementation, addressing issues such as

- How are information units mapped onto pages?
- How are navigation operations implemented?
- How are other interface objects implemented?
- How are existing databases integrated into the application?

It should be noted that we are separating navigation design from inteface design and from implementation design, which follows the OOHDM [Schwabe 96] architectural model. Similar separations (sometimes finer grained) can be found in other methodologies such as RMM and HDM.

There are many advantages in separating interface design from both implementation and navigation design. The WWW started as a standards-based architecture, where documents are described in HTML. Nevertheless, as soon as commercial interest stepped in, there was an escalating succession of (unilateral) HTML extensions by several manufacturers, leading to the point where many pages are only readable using one type of browser. This situation is further complicated by the possibility of extending functionality via scripting languages such as JavaScript and VBScript, or via plug-ins such as Shockwave or ActiveX.

As a consequence, the only way to protect the investment made in designing the interface is trying to represent the major design decisions that capture the "essence" of the interface in an implementation independent manner. If one is successful, even in the face of standards evolution, it is possible to maintain the core interface design, and reimplement a minimal part of the whole application.

Another advantage of separate interface design is the fact that, since communication with human beings is an important part of WISs, there are other disciplines that are required to make this effective, notably graphics design, multimedia (content) design, etc... The interface level is an ideal point where professionals from these other disciplines can contribute to the overall application design. Therefore, a separate interface design helps bridging the communication gap between computer professionals and professionals of other disciplines by helping focus on the aspects that must be jointly solved in order to reach an effective design.

In the next sections, we will describe how some of the questions raised above can be addressed using design patterns [Gamma95] as a presentation device. We then discuss some implementation issues related to WISs, and draw some conclusions.

2. Designing WIS with Patterns

Though originated in architecture [Alexander77] design patterns are being increasingly used in software design [Gamma95]. Design patterns are a good means for recording design experience as they systematically name, explain and evaluate important and recurrent designs in software systems. They describe problems that occur repeatedly, and describe the core of the solution to that problem, in such a way that we can use this solution many times in different contexts and applications. Looking at known uses of a particular design pattern we can see how successful designer solves recurrent problems.

In some cases, it is possible to give structure to single patterns to develop a pattern language: a partially ordered set of related patterns that work together in the context of a certain application domain. Design Patterns complement methodologies in that they address problems at a higher level of abstraction. Many design decisions that cannot be recorded through the use of the primitives of a method can be described using patterns'. In this way, we claim that using design patterns, WIS designers can profit from existing design knowledge in several communities such as hypermedia or user interface design.

We next present some patterns we have discovered and used in the context of developing WISs. These patterns address diferent design problems and they form the basis of a pattern language for WIS. For the sake of clarity they are organized in architectural, navigational design pattern and interface patterns. Architectural patterns help in deciding the overall WIS structure, for example by separating conceptual, navigational and interface design; navigational ones address aspects related with navigation inside a WIS whereas the latter are meant for designers of GUI.

We describe them using a single template describing the problem that originates the pattern, the motivation, the solution and an example in the context of WIS. This template allows expressing solutions in a "methodology-independent" way and so they can be used with different WIS design methods, such W3DT [Bichler97], RMM, OOHDM, WebDesigner [Takahashi 97]. A more detailed version of these patterns can be found in [Rossi97, Garrido97]

2.1 Node as a Navigational View

Problem: How to add navigation capabilities to the components of an existing application (for example a dbms one), therefore adding hypermedia functionality to it? How to combine conventional transactional processing with navigation?

Motivation: In many situations, existing applications can benefit from a hypermedia interface, such as when making them accessible via the WWW. Even when the application does not exist previously, it may be desirable to share a number of information items among several applications to be on the Web.

Solution: Define a navigational layer between the application to be enhanced and its graphical interface, build up of object's observers that are called *nodes*. Implement the navigational behavior in nodes. Then define each node's GUI by

adding means of activating node's behavior. Separate the application's conventional behavior from the navigational operations. The OOHDM methodology defines the concept of Node as a navigational view over a conceptual model [Schwabe96]; RMM has a similar construct in the "slice" primitive. This pattern can be used both during design to separate design concerns or during implementation (for example when using an object-oriented WIS development environment such as VisualWave [Vwave96]) to allow different views to be built from the same database. Intranets applications accessing a shared database must also use some version of this pattern to allow for different user profiles to access the database according to their needs. This pattern represents one of the most important architectural decisions, considering a WIS as a system that may add hypermedia functionality to more conventional applications.

Example: Many Internet News Agents use this pattern for providing personalized views of news database. In this case we find a combination of Node as a Navigational View with dynamic Page Creation Method (see 2.4). For example, it is possible to define Custom News at http://www.news.com, as shown in Figure 1.

Figure 1 - An example of the use of "Node as Navigational View" to customize a page. (http://www.news.com)

2.2 Node as a Single Unit

Problem: What constitutes an "information unit" with respect to navigation?

Motivation: A node should encompass a self-contained "unit" of information, that should make sense for a set of users performing a set of tasks in a given domain.

A node does not necesseraliy have to correspond to a single web page. For example, it is quite common to find web pages that are very long, including different merged topics, some of which may not be relevant to the task at hand. For instance, in http://www. cs.brown.edu/memex/, a single page shows a big picture at the beginning, then information of what Memex and Beyond is about, then a global index of topic, then a description of what is still missing comes after that, followed by details about the kinds of navigation that appear in the web site, etc. Conversely, if a node is complex (or is made out as a composite), it may be better to split it up into several pages, although care must be taken so as to not make reading and printing more difficult, since the reader must navigate from one fragment to the next in order to see the entire "unit".

Solution: Make a node a single unit, self-contained entity of information, focusing on a certain topic. All data that is relevant for that entity should be included the same node. Good object-oriented modeling of the application domain may give a good initial grasp for defining each node; when several different objects contain closely related information, define a single page (node) as a view over them.

Example: In the Amazon books WIS,(http://www.amazon.com) information about a book is shown in the way described by this pattern.

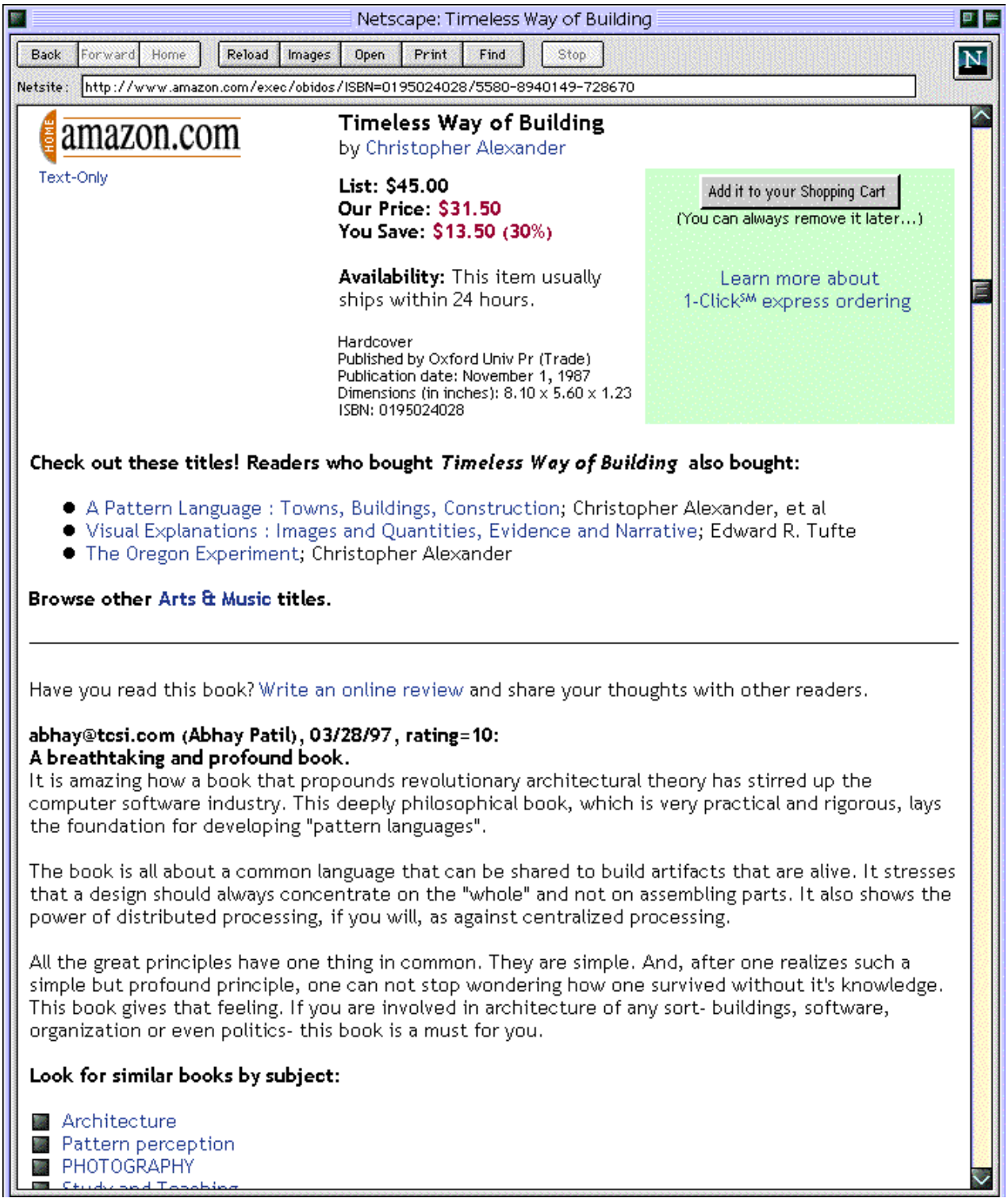

Figure 2 - An example of use of the pattern "Node as a single unit" in http://www.amazon.com

2.3 Navigational context

Problem: How to organize the WIS navigational structure, providing guidelines, information and relationships that depend on the current state of navigation, in such a way that information can be better presented and comprehended ?

Motivation: WIS usually involve dealing with collections (e.g., Records, Paintings, Cities, Persons, etc.). These collections may be explored in different ways, according to the task the user is performing. For example, we may want to explore Books of an author, Books on a certain period of time or literary movement, etc., and it is desirable to give the user different kinds of feed-back in different contexts, while allowing him to move easily from node to node.

Suppose for example that in a WIS for a bookstore, we reach "William Shakespeare", choose to navigate to his books, and then arrive at "Romeo and Juliet". However, we can also reach "Romeo and Juliet" while exploring Books written during Romanticism period, or navigating through some Publishing CompanyEditing House. It is clear that we will explore the same object under three different perspectives; for example while accessing it as a Shakespeare's work we would like to read some comments about its relationships with other works by Shakespeare (e.g., as it relates to "Love's Labour's Lost" because both incorporate sonnets into their structure), and also to have easy access to the next book he wrote (say, in chronological order. Meanwhile, as a Romantic work, it would be fine to read (or see) something about the Romanticism period and be able to access other works on the same movement (perhaps not Shakespeare's) that we can buy at the bookstore. This means that we will need not only to present the information in a different way in all cases but also to provide different links or indexes.

Solution: Decouple the navigational objects from the context in which they are to be explored, and define objects' peculiarities as Decorators [Gamma95], that enrich the navigational interface when the object is visited in that context.

Navigational Contexts are composed of a set of Nodes (like Books) and Context Links (links that connect objects in a context). Nodes are decorated with additional information about a particular context and additional anchors for context links. The navigational context may also contain information about the context itself (for example an explanation about Romantic Books) that will be shown in a particular Context Node. That node may provide an index to all nodes in the context or a link to the first one.

A diagram of the interacting elements is shown in Figure 1, where the context node provides an Index to the nodes and also each decorator provides an anchor to the 'next' node in the context.

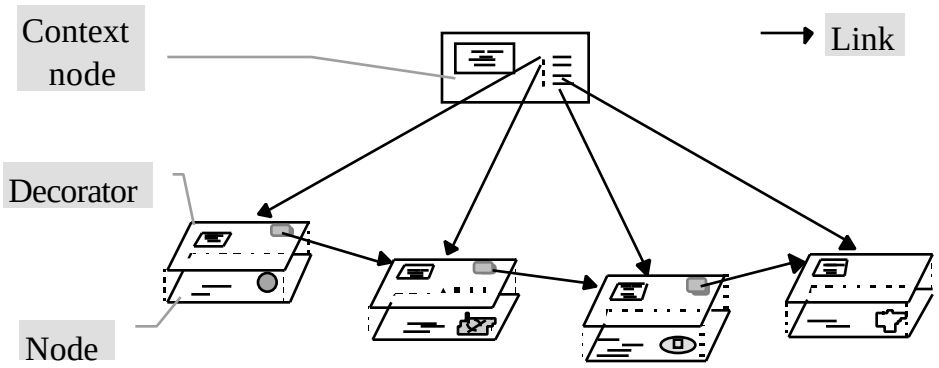

Figure 3 Diagram of Navigational Context pattern

Identifying navigational contexts is important because they are high level architectural constructs that help organizing navigation in such a way that we can describe the navigational structure of a WIS not only as a set of pages but also as a set of contexts in which those nodes will be accessed. There are different strategies for defining navigational contexts; for example the set of instances that satisfy some property of a class (e.g, "all books authored in 1997") may be traversed in a navigational context; a n-ary link also originates a context (e.g, "all books *writen by* Shakespeare").

Another direct consequence of defining contexts is the definition of access structures (indexes and guided tours) allowing access to the elements in a context. In this manner, contexts may also help the user to find the proper place where to start navigation.

This pattern is a primitive in OOHDM [Schwabe 96].

Example:

In Figure 4 on can see a page of the Portinari Collection site. This page describes one painting, which is being navigated in the "artworks by theme" context. This page is also part of other contexts, as indicated in the figure: "artworks by date" and "artworks by technique". A more detailed discussion can be found in [Schwabe 96].

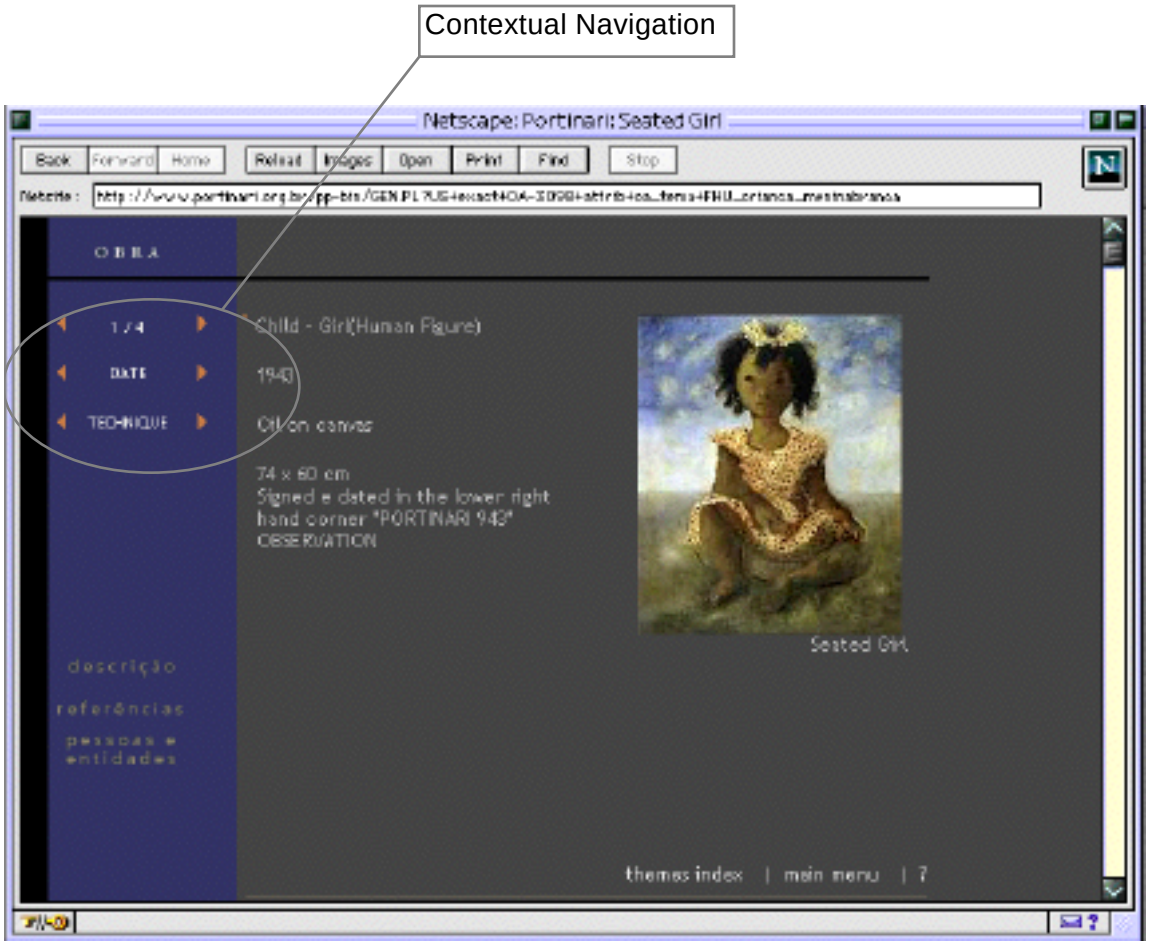

Figure 4 - Example of the pattern “Navigation Context” in the Portinari Collection application (http://www.portinari.org.br) Artowrks can be navigated by theme, date or technique..

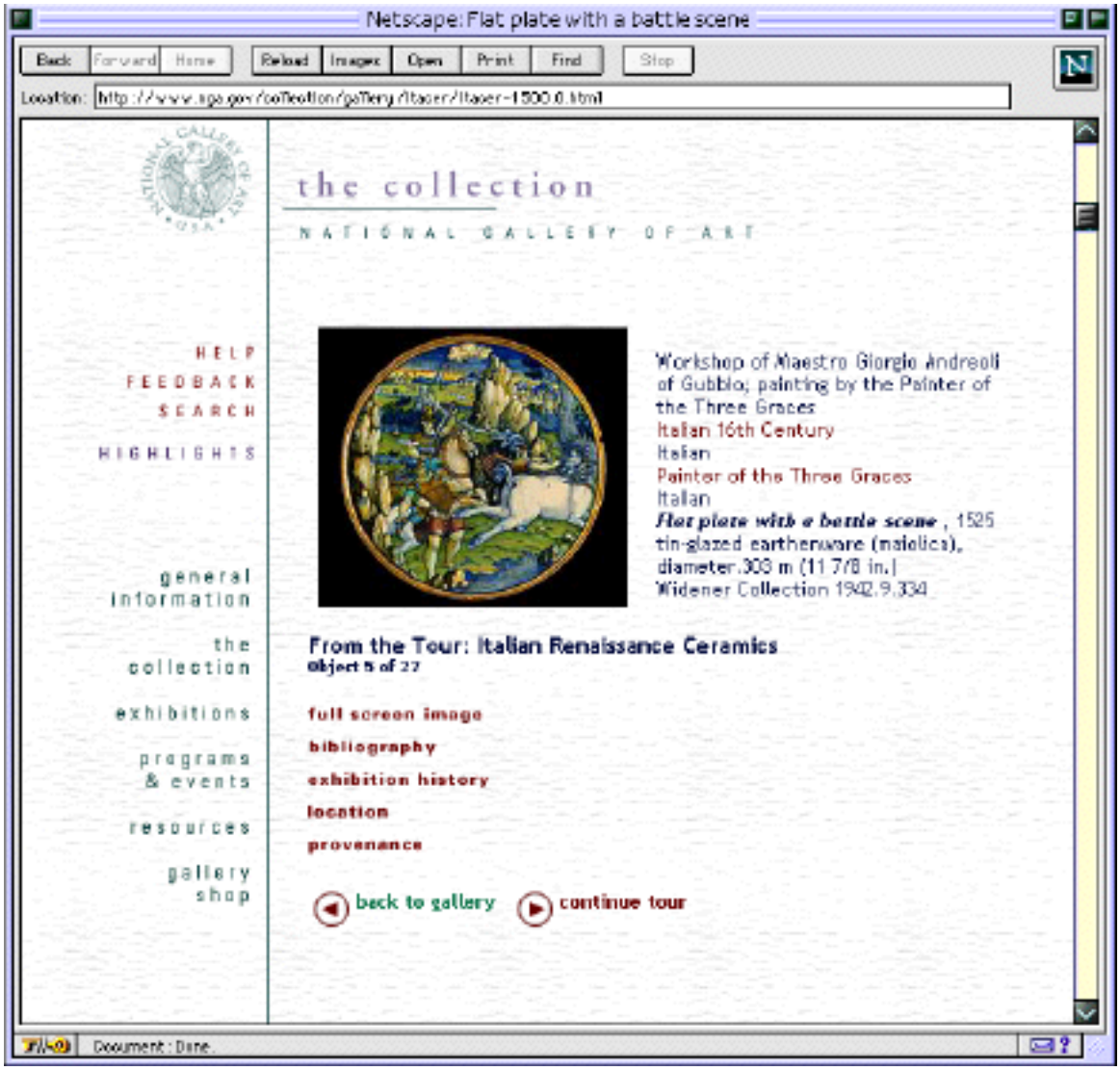

Figure 5 - Another example of the "Navigation Context" Pattern from the National Gallery of Arts site (http://www.nga.gov). A guided tour of "Italian Renaissance Ceramics" forms a context.

There are several other examples, as shown in Figure 2 and Figure 5. There are several sites that also employ this pattern to customize the advertisement banner depending on the way the user arrived at the page; this corresponds to having a decorator object depending on the context in which the page is being navigated.

2.4 Page Creation Method

Problem: When is it better to create pages statically, and when is it preferable to create pages dynamically?

Motivation: In many applications, the contents of a node (page) do not change, or change very little. On the other hand, in WIS, where information is captured from databases, nodes must be defined dynamically.

Solution : Pages must be dynamically created when the application's data and functionality will be constantly updated, when readers can modify, create, or compose new nodes, and when instant update is needed. Otherwise, when there is no underlying application, and the set of nodes is limited and fixed, manual creation is the most acceptable solution. In some cases the computation of nodes on demand (when they are the destination of a navigation step) can be a heavy process, and so it is better to pre-compute the nodes by running the defining queries at definition time, and storing the results. This approach is only feasible when data on the data base changes at a low enough rate.

Example:

There are inumerous examples on the WWW; for instance, most news sites and most online merchant sites generate pages dynamically. The example is shown in Figure 1 for the http://www.news.com site.

2.5 Link Creation Method

Problem: When is it better to define static links, and when is it preferable to create links through computations?

Motivation: Similarly to 'Node creation method', sometimes is hard to decide the pros and cons of defining links by hand (by directly defining the target URL or by means of a computation. In some cases invariant relationships in the problem domain are best represented by static links, while dynamic link creation is reasonable when nodes are creating dinamically (see the above pattern) and as a side ef

Solution: Static links are used in closed, static applications, when maintainability in case of future change is not an issue, and nodes are also statically created. Static links may be used in dynamic applications when the definition of a link-computation is too complex, the endpoint node is very difficult to be changed, or the link is only temporary. Computed links should be preferred in dynamic applications where new nodes are also created dynamically, data is volatile, and maintainability must be efficiently achieved. Use also dynamic links, even with statically created nodes, when it is possible to define link types, and when the number of links of each type to create is considerable (i.e., the effort require to write a computation to automatically create the links is less than the effort to manually instantiate all links in a relationship between large sets of nodes).

Example:

A prime example are search sites, such as http://www.altavista.com or http://www.excite.com. Links are included in each page as a result of executing a search query against a document database.

2.6 Information-Interaction Decoupling

Problem: How do you differentiate contents and various types of controls in the interface ?

Motivation: A page in a complex application displays different contents, and is related to many other pages, thus providing many anchors. Moreover, if the page supplies means of control activation other than navigation (such as poping-up some information or triggering some query), the user may experience cognitive overhead. It is well known that when too many anchors are provided in a text, the reader is distracted and cannot take profit of all of them (http://www.autoweb.com/ is an example of the occurrence of both problems).

Solution: Separate the input communication channels from the output channels, by grouping both sets separately. Allow the "input interaction group" to remain fixed while "the output group" reacts dynamically to the control activation. Within the output group, it is also convenient to differentiate the "substantive information" (i.e., content) from the "status information". This solution not only improves the perception of a node's interface, but also the efficiency of the implementation.

Example:

This is a common pattern found in the WWW. An example is shown in Figure 6 for the http://www.zdnet.com site, where general controls are on the left bar, and content specific links are on the right hand side; no links are included in the text itself. Most other examples in this paper also employ this pattern.

Figure 6 - An example of the "Information/Interaction Decoupling" pattern at the http://www.zdnet.com. General Links are on the left; content specific links are on the right. Notice there are no links in the text itself.

2.7 Behavioral Grouping

Problem: How to organize the different types of controls in the interface so that the user can easily understand them?

Motivation: A problem we usually face when building the interface of a WIS is how to organize control objects (such as anchors, buttons, etc.) to produce a meaningful interface. In a typical WIS there are different kinds of active interface objects: those that provide "general" navigation, such as the "back" button, or anchors for returning to indexes; objects that provide navigation inside a context;

objects that control the interface, etc. Even when applying Information-Interaction Decoupling there may be a lot of different kinds of control objects.

Solution: Group control interface objects according to their functionality in global, contextual, structural and application objects, and make each group perceivable in a different screen area. Provide similar interface appearance inside each group to enhance comprehension.

Example:

In Figure 7 we see an example of Behavioral Grouping in the context of http://expedia.msn.com. There are three screen areas for control objects: one in which "general" anchors are presented (at the top), another for context anchors (at the left) and another one for action item at the bottom.

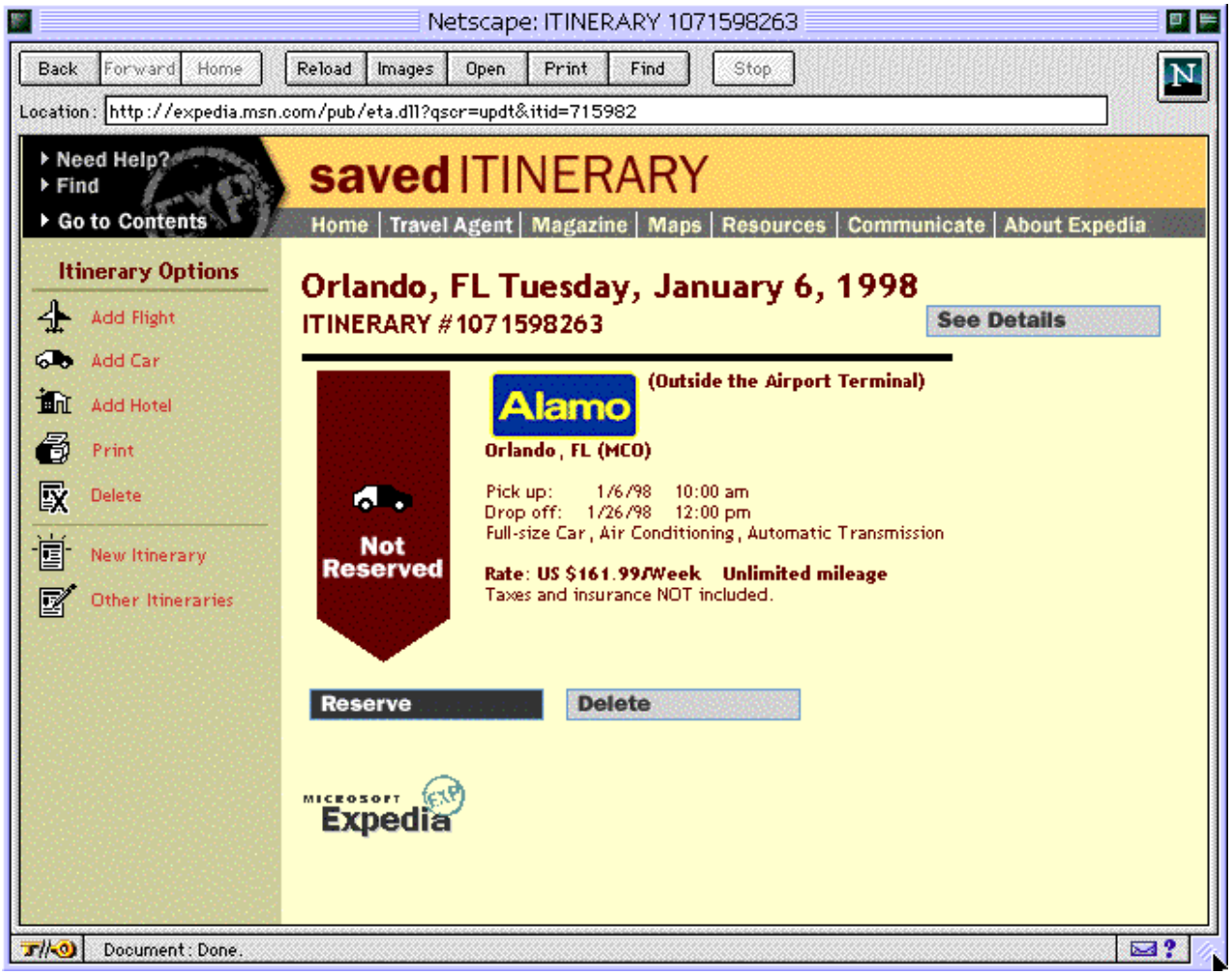

Figure 7 - An example of "Behavioral Grouping" in Microsoft's Expedia site (http://expedia.msn.com). Notice the controls at the navigation bar at the top; the itinerary options at the left and the actions at the bottom.

3. Implementation

During implementation, the designer will actually implement the design. Though it is not the focus of this paper to analyze implementation aspects, we will briefly discuss how some of the previously presented design patterns may be implemented.

When an object-oriented environment is used it is straightforward to implement the separation of concerns we propose in this paper: the domain model will be implemented by a set of classes, while navigational operations can be defined in classes mapping hypermedia concepts like node, link, etc. An external database can be used to contain the application information. The interface is naturally decoupled and implemented using HTML (or combinations of HTML, Java-Script, etc)

However in non-object oriented environments some patterns may require more complex implementations. There are basically two types of concerns that must be faced: how to implement the information objects (placeholders) and how to implement the interface objects.In the discussion that follows, we will briefly examine each of these concerns, in the light of the WWW implementation environment.The implementation of the information objects traditionally called nodes, will typically be done using some sort of database, or perhaps a set of files in the file system.

In addition to mapping information objects into whatever database model (relational, OO, etc...), it is also necessary to implement “within context” information, which functions as decorators within particular contexts. Typically, this entails enriching the data model used in the database to account for the added attributes, and defining control functions that make these attributes accessible in the appropriate contexts. If the implementation is based directly on the file system, these control functions will access additional files containing the contextual information.

Whereas the mapping of information objects into implementation objects is somewhat obvious, the implementation of contexts is more complicated. The supporting database model or set of files must also contain the context definitions. With the exception of arbitrary contexts (whose specification is an enumeration of its members), other types of contexts include a query or function specification that must be evaluated to derive the members of the context. This computation can be done either previous to application deployment or at runtime; the former is preferrable if the data changes infrequently, and the latter when data changes frequently and the user must always access the most recent data.

Navigation operations within contexts require keeping state information. For example, to determine “what is the next recommended restaurant in this area” requires knowing which area the user is currently looking at, and which restaurants make up the referenced context (“restaurants in a given area”), and what is the ordering defined for that context (e.g., “alphabetical” or “price range”).

In terms of the WWW this means that either this state information is kept within the database or file structure being used, or special control information is kept on the side to represent the navigation state. In this case, any of the better know

techniques for keeping state in the WWW may be employed: passing state information within URLs, from page to page; keeping state information in hidden fields passed on from page to page; or using cookies. All of these techniques require using CGI scripts to implement navigation.

A different technique that has also been employed is to represent information objects – both instance data and context information – as constants in JavaScript, which are manipulated through funtions in scripts inserted in the root document of a frameset. The scripts are executed to implement navigation operations within documents that are stored in other frames in the frameset. The advantage of this approach is that state maintenance is done entirely within the client machine; the disadvantage besides only working for browsers that understand JavaScript is that it breaks down for large systems as one reaches the limits in size for Javascript programs. It should be noted that this technique may also be used in combination with the previously mentioned ones.

In some situations, if the number of instances is small enough, it may be justifiable to "precompute" all contexts and all navigations paths, and generate separate page copies corresponding to different node instantiations within the various contexts. In this case, all links are represented as static URLs, and it is not necessary to use any of the mechanisms mentioned in the previous paragraph.

4. Conclusion

Designing successful WIS is hard; developers must deal with different concerns such as organizing access to external resources (like databases), defining the navigational structure of the application, building a good user interface, etc. Some methods have been proposed to help designers in this task like OOHDM [Schwabe96], W3DT [Bichler97] and RMM [Izakowitz95, Takahashi97]

We claim, however, that design methods solve just part of the problem and propose using design patterns as a way to record and reuse designers' experience while developping WIS. In this paper we have presented some design patterns we have found in different WIS development enterprises. These patterns address architectural, navigational and user interface problems and, as shown in this work, can be sistematically implemented in the WWW environment. We consider them as a basis of a pattern system for the WIS domain. Being WIS design an enterprise that needs the use of design knowledge from different fields (database, hypermedia, user interface design, etc),his pattern system should be an excellent medium for recording and transmitting this experience to WIS designers and for them to point the new, recurrent problems they find.

Once a rich set of design patterns for WISs is available, designers can begin their work from a higher level, possibly with the aid of (semi) automated tools. As a result, there will be an increase in quality of WIS designs being deployed in the market.

5. References

[Alexander77] Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King

and S. Angel: "A Pattern Language". Oxford University Press, New York 1977.

[Berners-Lee97] Berners-Lee, T.; “The World Wide Web - past, present and future”, Journal of Digital Information 1 (1) (http://journals.ecs.soton.ac.uk/jodi/Articles/timbl.html)

[Bichler97] Bichler, Ma.;Nusser, S.: “Modular Design of Complex Web-Applications with SHDT. In http://dec9.wu-wien.ac.at/w3dt/wetice/wetice.html.

[Gamma95] Gamma, R. Helm, R. Johnson and J. Vlissides: "Design Patterns: Elements of reusable object-oriented software", Addison Wesley, 1995.

[Garrido97] A. Garrido, G. Rossi, D. Schwabe: “Patterns Systems for Hypermedia”. Proceedings of PLoP’97, Pattern Language of Program, 1997. In http://st-www.cs.uiuc.edu/~hanmer/PLoP-97/

[Isakowitz95] T. Isakowitz, E. Stohr and P. Balasubramanian. “RMM: A Methodology for Structured Hypermedia Design”. *Communications of the ACM 38* (8), 1995, pp. 34-44.

[Rossi97] G. Rossi, D. Schwabe and A. Garrido: “Design Reuse in Hypermedia Applications Development” Proceedings of ACM International Conference on Hypertext (Hypertext’97), Southampton, April 7-11, 1997, ACM Press.

[Schwabe96] Schwabe, G. Rossi and S. Barbosa: "Systematic Hypermedia Design with OOHDM". Proceedings of the ACM International Conference on Hypertext (Hypertext'96), Washington, March 1996.

[Takahashi 97] K. Takahashi, E. Liang: “Analysis and Design of Web-based information systems” Electronic Proceedings of The Sixth International WWW Conference, Santa Clara, USA, 1997.

[VWave96] The VisualWave Programming Environment. Parc Place Systems. In http://www.parcplace.com/products/vwave/vwv_prod.htm.