Multimedia Tools and Applications 8, 293–308 (1999)

Discovering and Using Design Patterns in the WWW

FERNANDO LYARDET fer@sol.info.unlp.edu.ar
LIFIA, Departamento de Informática, UNLP, Argentina

GUSTAVO ROSSI gustavo@sol.info.unlp.edu.ar
LIFIA, Departamento de Informática, UNLP, Argentina; also at UNLM and Conicet

DANIEL SCHWABE schwabe@inf.puc-rio.br
Departamento de Informática, PUC-Rio, Brazil

Abstract. In this paper, we discuss how the idea of design patterns can be used in the context of the World Wide Web, for both designing and implementing web sites or more complex information systems. We first motivate our work by discussing which are the most outstanding problems in designing Web-based information systems. Then we briefly introduce design patterns and show how they are used to record and reuse design information. We next present some simple though powerful design patterns and show known uses in the WWW. Finally, we outline a process for building applications by combining a design methodology (OOHDM) with design patterns.

Keywords: hypermedia, patterns, design methods, navigation, WWW

1. Introduction

Web-based applications are hard to build. Even simple web sites involve solving a myriad of problems such as organizing the structure of pages, presenting them in a meaningful way, etc. Moreover, as more corporations re-design their information systems such that they can be accessed from web browsers, we face new design and implementation problems. In a broad sense, we can categorize these problems in four groups, which will be elaborated further later.

The first group involves design issues that have to do with the overall application architecture and we have to answer questions such as:

- 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?
- How we combine navigation (as widely known in the WWW) with more conventional transactional behavior.

The second 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, i.e., establish the possible sequences of information units the user may navigate through?
- How do we avoid disorientation in a large information space?

The third group of design issues has to do with the organization of the interface:

- 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?

Finally, the fourth 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?

Though all these problems are most of the time managed in an environment-centric way (i.e., according to the restrictions of the final implementation platform), we claim that the most outstanding design decisions must be made explicit using some design notation. In this paper we argue that design patterns complement design methods by allowing us to focus on the solution of recurrent design problems in an abstract way.

It should be noted from the description above that we are separating navigation design from interface design and from implementation design, which follows the OOHDM [12] architectural model. Similar separations (sometimes finer grained) can be found in other methodologies such as RMM [8] and HDM [7] or W3DT [2]. 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. The rapid emergence of Java as a tool for building dynamic Web-sites and the huge set of interface features now appearing can only be managed by treating interface design as a separate activity.

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 re-implement 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 Web-based applications, 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.

Meanwhile, building a healthy and usable Web application requires a careful design of the navigation structures; deciding when to use hierarchical structures, where to use indexes, improving backtracking, providing navigation histories or maps are design decisions that must be documented. Unfortunately, design methods like OOHDM, RMM or W3DT though providing primitives for representing most design decisions, fail to express the rationale of those decisions.

In the next sections, we will describe how some of the questions raised above can be addressed using design patterns [5] as a presentation device.

2. Why design patterns in the WWW

Though originated in architecture [1], design patterns are being increasingly used in software design [5]. 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. Patterns are not just good solutions that we invent to solve particular problems; they are solutions that appear repeatedly in different applications; patterns are not invented, they are discovered.

Design Patterns complement methods in that they address problems at a higher level of abstraction. Many design decisions that cannot be recorded using the primitives of a method can be described using patterns. We claim that using design patterns, web application designers can profit from existing design knowledge in several communities, such as hypermedia or user interface design, since many of the problems we mentioned in Section 1 also appear in hypermedia or more general interactive applications. However, since the WWW is a completely new applications environment, new recurrent problems will appear and it is possible that a whole pattern catalogue should be developed in this field. This paper has two motivations: to introduce the field to researchers and practitioners in the WWW area and to motivate the WWW community to look for new patterns in web applications.

An interesting issue that is often discussed in the pattern community is how patterns compare with guidelines or with good development practices such as for example those found in [10] and [3]. Patterns usually express recurrent problems whose solutions may depart from the basic elements of a paradigm. Besides, they usually state a context in which those solutions may be applied, so that applying a pattern also generates a kind of trade-off that serves us to evaluate its benefits or drawbacks. In the object-oriented design field,

this difference is easily perceived because patterns involve different cooperating objects (or classes) while guidelines or heuristics tend to be less structured. Patterns may be simple, such as, for example, Behavioral Grouping (in Section 3.2.2) or more complex like Navigational Context (in Section 3.1.2). The former only involves a set of simple rules and relationships, and can be thought of as an elaborated guideline; the latter involves more complex object relationships.

We next present some patterns we have discovered and used in the context of developing web applications. These patterns address different design problems and they form the basis of a pattern language for the WWW domain. For the sake of conciseness we discuss navigational design pattern and interface patterns. Some architectural patterns can be found in [6].

3. Patterns in the WWW

3.1. Navigational patterns

The patterns presented in this section, face common problems in the organization and access to the information. There is an underlying idea connecting the patterns presented here that stressed the importance of a clean separation between interface and. Some methodologies such as OOHDM [12] propose an architectural model that follows this principle. Similar separations can be found in other methodologies such as RMM [8] and HDM [7].

3.1.1. 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 in 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; RMM has a similar construct in the slice primitive. This pattern can be used both during design and during implementation. During design it is used to separate design concerns; during implementation (for example when using an OO Web Information System (WIS) development environment (such as VisualWave [14]), to allow different views to be built from the same database. Intranet applications access a database according to their needs.

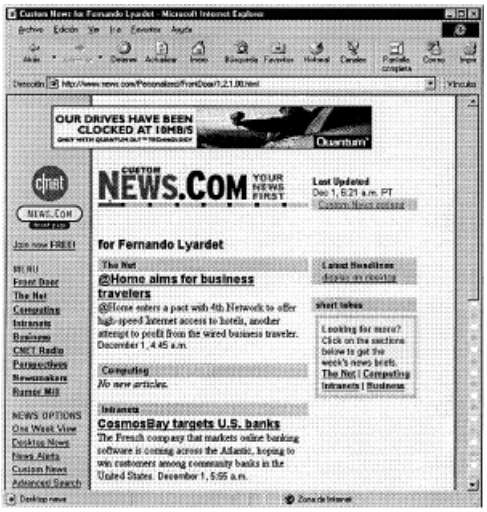

Figure 1. Example of Node as Navigational View over a news information databases (`http://WWW.news.com`).

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.

Known uses. Many Internet Newsagents use this pattern for providing personalized views of news database. For example, it is possible to define Custom News at `http://news.com` as show in figure 1.

3.1.2. Navigational context.

Problem. How to organize the website 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. Browsing information, usually involve dealing with collections (e.g., Concepts, 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. It is desirable to give the user different kinds of feedback in different contexts, while allowing him to move easily from node to node.

Solution. Decouple the navigational Objects from the context in which they are to be explored, and define objects' peculiarities as Decorators [5], that enrich the navigational interface when the object is visited in that context.

Navigational contexts are composed of a set of Nodes (like Books or Inventions) and Context Links (links that connect objects in a context). Nodes are decorated with additional information about the particular context and additional anchors for contexts links.

This organization strategy helps to group nodes in meaningful sets that can be easily explored. It also address the problem that arises when the same node belongs to more than a set by indicating that in each case the node must be decorated according to the context.

Implementation. Navigational contexts are naturally implemented on sites where pages are created dynamically like `http://WWW.portinari.org.br/`, where the same information can be navigated under different perspectives. Nevertheless, if the amount of information is relatively small with a small change rate, simple contexts can be built on static content sites, like WWW.nga.gov.

Known uses. A good example is available at the US National Gallery of Art, (see `http//WWW.nga.gov/collection/gallery/ggsculpt-12189.0.html`), where a guided Tour of "Marble Sculpture from France" forms a context.

Related pattern. Node as a Navigational View can implement navigation through information with contexts.

3.1.3. Active reference.

Problem. How can we provide a perceivable and permanent reference about the current status of navigation, combining an orientation tool with an easy way to navigate to a set of related nodes, at the same or higher level of abstraction?

Motivation. In many hypermedia applications (particularly those involving spatial or time structures) we need to provide the reader with a way to understand where he is and help him decide where to go next. The usual solution would include an index (or other access structure) to the elements we intend the user to navigate. However, this solution will require the user to backtrack from the current node to the index to see where he is or to move to another node, while ensuring that its current position is highlighted in the index. These navigational operations: moving backward to the index and forward to the target increases the cognitive overhead due to the context switch and may disorient the user.

Solution. A good solution is to maintain an active and perceivable navigational object acting as an index for other navigational objects (either nodes or sub-indexes). This object remains perceivable together with target objects, letting the user either explore those objects or select another related target. In this way, we will be able to interact with both the index and the target nodes. Active reference provides an effective way for both providing indexes to the information and making those indexes co-exist with the information. Notice that Active Reference may be complemented with Information on Demand to allow a better organization of the screen. As an example, in figure 3 the reader has a permanent reference about where he is located while exploring cities in the world.

Implementation. Active reference is usually implemented using a visual representations of menu-like structures as shown in figure 3 on sitebuilder.com or stack lists such as in excite.com, where the lower items are the most recent (or more deeply nested).

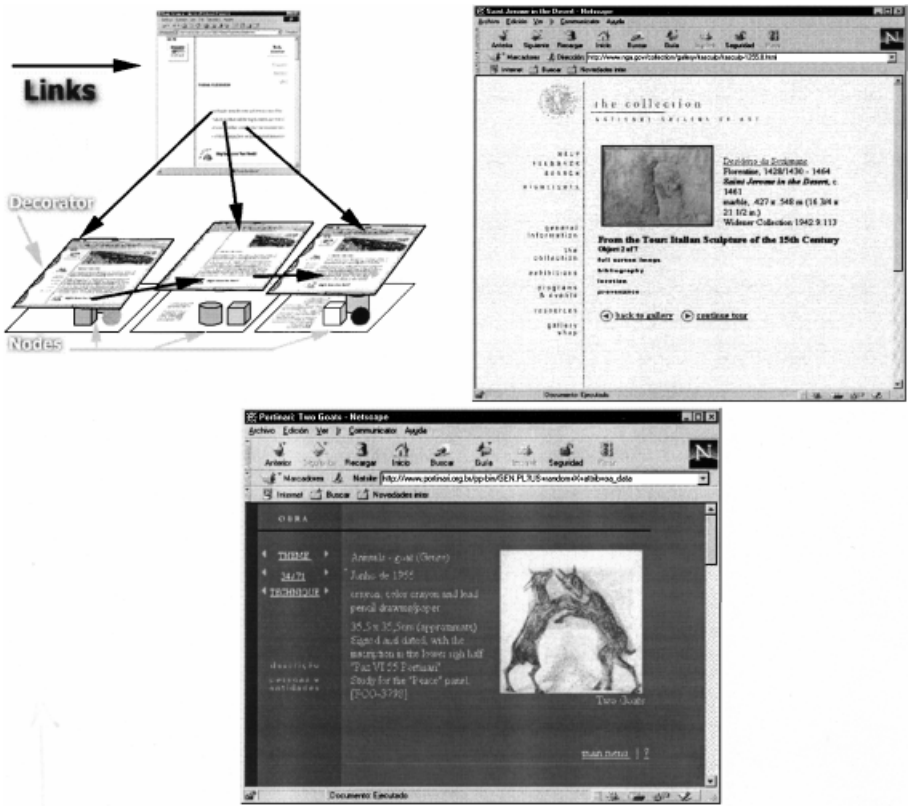

Figure 2. Diagram of "Navigational Context" pattern and two working examples: `http://WWW.nga.gov/collection/gallery/itasculp/itasculp-1255.0.html` (statically defined contexts) and `http://WWW.portinari.org.br/` (dynamically defined).

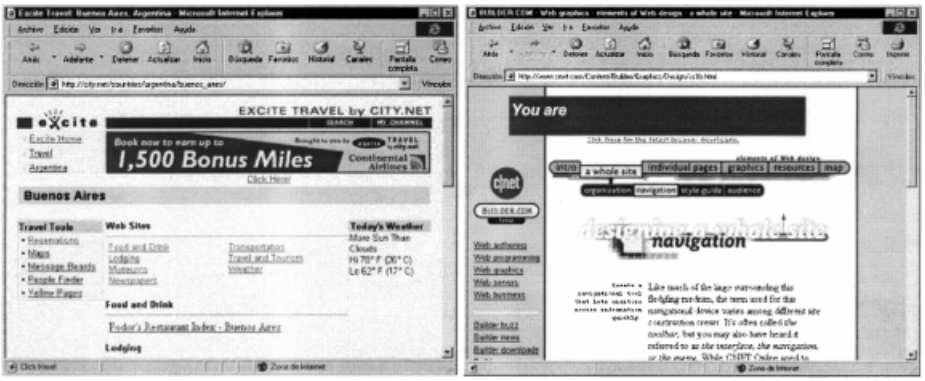

Figure 3. Two examples of "*Active Reference*" in `http://city.net/countries/argentina/buenos_aires/` and `http://WWW.cnet.com/content/builder/`.

Examples. In figure 3, we present a couple of examples of "Active Reference".

3.2. Interface patterns

3.2.1. 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, therefore providing many anchors. Moreover, if the page offers other ways of control activation in addition to navigation (e.g., 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 profit from of all of them.

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.

Implementation. In general, sites should be divided into five sections, give or take two. (This is a basic guideline; a site like CNET, with its wide variety and huge amount of content, could warrant more sections, while a "content-light" site might not need as many)

Example. In figure 4, all links to related information on the current topic are displayed on the left. The graphics/video relevant to the current topic is displayed in the middle. Notice there are no links in the text itself.

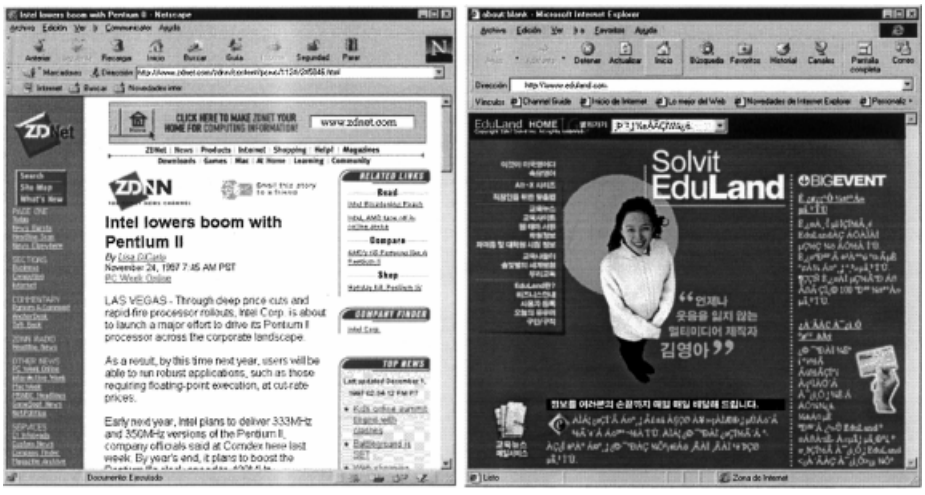

Figure 4. Examples of Information/Interaction Decoupling from ZDNET and Eduland. In the first example, general links are on the left; content specific links on the right. Notice there are no links in the text itself.

3.2.2. Behavioral grouping.

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

Motivation. A problem we usually face when building the interface of web application is how to organize Control Objects (such as anchors, buttons, etc.) to produce a meaningful interface. In a typical web application. There are different kinds of active interface objects: those that provide "general" navigation, such as "back" button, or anchors for returning to indexes; objects that provide navigation inside a context; and objects that control the interface. Even when applying Information-Interaction decoupling, there may be many different kinds of control objects.

Solution. Group control interface objects according to their functionality in global, contextual, structural and application objects, so that each group enhances comprehension.

Implementation. No matter if the website has been designed using frames or tables, the technique is to group more closely all related control objects (buttons/links). There should be clear separation between different groups with space or line separators when they share a single column.

Known uses. There are hundreds of websites that could be cited as example. In figure 5 we present one form Hewlett Packard and other from Microsoft's Expedia websites.

3.2.3. Information on demand.

Problem. How to organize the interface in such a way that we can make all the information in a node perceivable, taking into account both aesthetic and cognitive aspects?

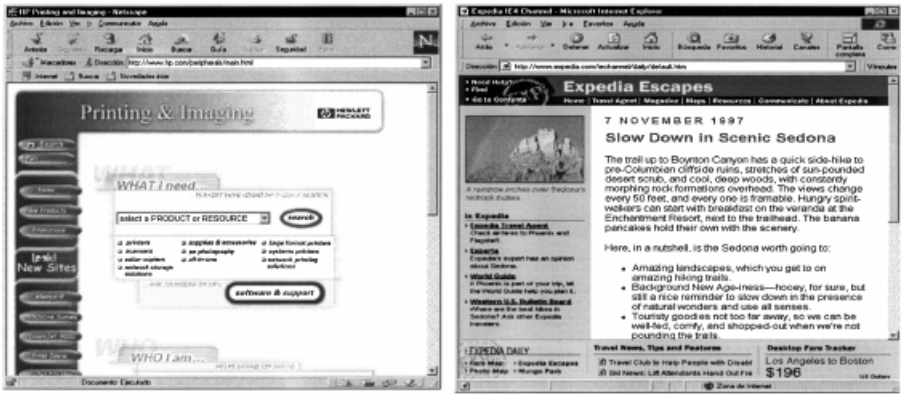

Figure 5. Behavioral Grouping in HP and Microsoft's Expedia website. Notice in the first example the controls separated in three different groups according to their functionality. The second example shows the controls on the navigation bat at the top.

Motivation. We usually find ourselves struggling to decide how to show the attributes and anchors in a node. Unfortunately, the screen is usually smaller than what we need and many times we cannot make use of other media (such as simultaneously playing an audio tape and showing an image) either for technological or cognitive reasons.

Solution. Present only a sub set of the most important ones, and let the user control which further information is presented in the screen, by providing him with active interface objects (e.g., buttons). The activation of those buttons does not produce navigation; they just cause different information of the same node to be shown. This just follows the "What you see is what you need" principle.

Implementation. This can be implemented through the use of standard HTML topic links, one for each subset of information that belongs to the same node. This approach has the advantage of avoiding scattering the information through different nodes and from the user's perspective is very simple to print all the information at the same time instead of doing so page by page. The main disadvantage is that the user may feel discomfort from the local up/down navigation.

With the upcoming of DHTML, the information displayed to the user can be changed dynamically (an example is available in Microsoft's Home Page `http://WWW.microsoft.com/`, using ie4). This technique can be used to activate in turn, the subset of the node's information to be displayed without forcing further and unnecessary navigation (see figure 6).

Related patterns. This pattern is a natural complement to the "Node as a Single Unit" since not all the information that belongs to a single information unit may be displayable altogether. Another related pattern is "Information/Interaction Decoupling" to help organizing the links specific to the current displayed node from more general links.

3.2.4. Behavior anticipation.

Problem. How do you tell the user the effect or consequence of activating an interface object?

Motivation. Many times, when building an interface, it is necessary to combine different interface elements such as buttons, hotwords, media controls or even custom-designed controls. It is usual to find readers wondering what happened after they activated a control, and the exact consequences of the action performed.

Solution. Provide feedback about the effect of activating each interface element. Choose the kind of feedback to be non-ambiguous and complete: different cursor shapes, highlighting, small text-based explanations called "tool tips". In addition, these elements can be combined with sound and animations.

If we are using the behavioral Grouping interface pattern, we can select different kinds of feed-back, according the kind of behavior provided. When the interface controls refer to a particular media such as animation, we could use a small status field for that family.

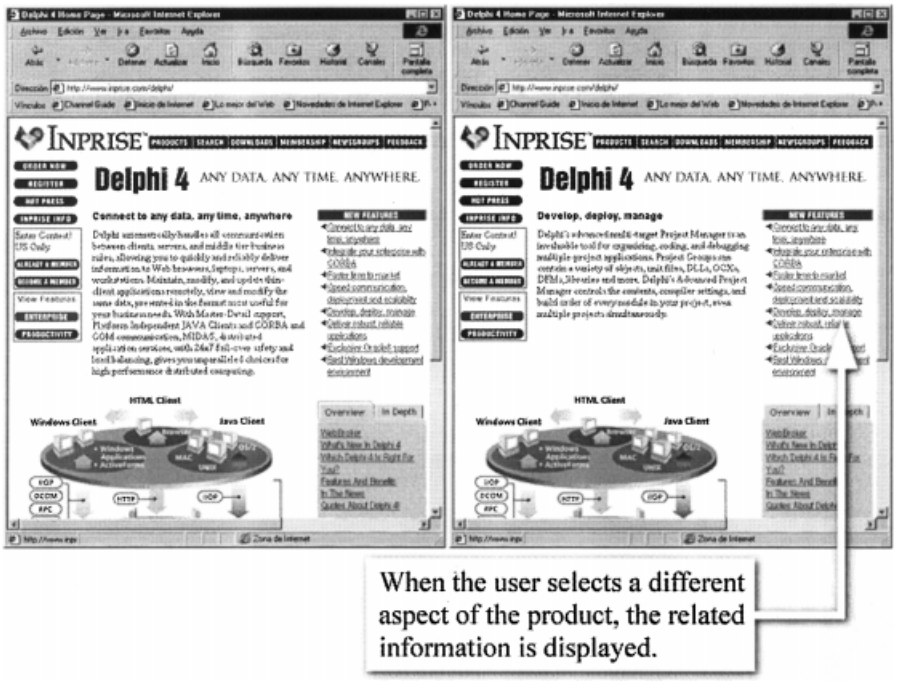

Figure 6. An example of Information on Demand in a node containing information about "Delphi 4". When the user selects an item from the "New Features" index, the information on the desired topic is automatically displayed without having to navigate to another page. Notice that the node is conceived as a "single unit" of information instead of scattering the contents in several pages. (`http://WWW.inprise.com/delphi/`).

Implementation. Several websites provide feedback to the users by standard GUI hints (ToolTips) using a feature that most browsers now provide that uses the ALT information of a picture. Other implementations more sophisticated has been made using JavaScript to present additional information on the status bar and graphical animations.

Known uses. A web site using standard GUI ToolTips and JavaScript combination to show information on the status bar is `http://WWW.ibm.com`. Other sites have built custom tips such as `http://WWW.nervemag.com/` (the ToolTip appears at the bottom of the page), see figure 7.

4. Patterns+design: From patterns to applications

As we previously explained, design patterns are useful to record recurrent design themes in software systems. Studying successful applications and reflecting on how they solve recurrent problems usually discovers them.

Being knowledgeable of a catalogue of patterns, a designer can work in a higher abstraction level because he does not solve every problem from scratch but apply existing

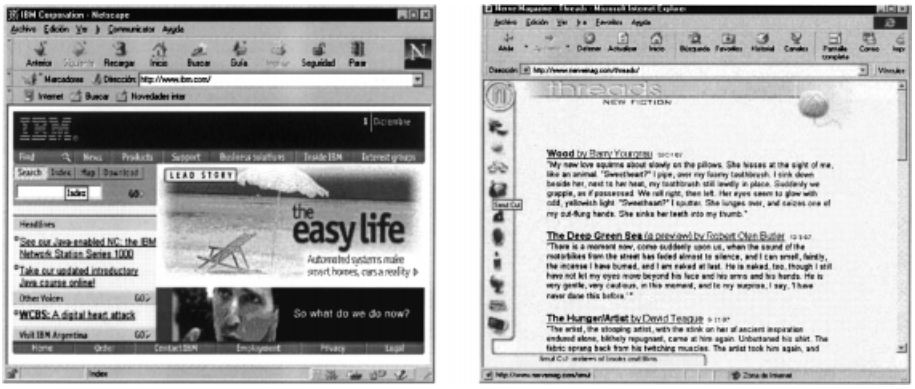

Figure 7. Two examples of Behavior Anticipation. Both websites provide hints to the user to prevent the excess of navigation and thus the inherent information overhead.

experience to the particular situation he is facing. However, integrating the use of design patterns into the software development life cycle is not easy. In some domains, it is possible to define a pattern language: a set of patterns that not only comprise the whole set of design problems, but are also related, and give guidelines during the development life cycle (see for example [9]).

We have developed the Object-Oriented Hypermedia Design Method (OOHDM), and have used it to design and implement different Web applications [13]. As a result of our design experience, we have incorporated some patterns into our method. OOHDM separates conceptual from navigational and interface design. In this way by treating nodes and links as object-oriented views of conceptual objects we provide an interesting architectural style for Web applications that separates navigation from other kind of computations (for example those existing in legacy applications). The underlying pattern here is a combination of the Observer and the Decorator in [5]. Since this strategy is quite usual when porting existing applications (or databases) to the Web environment, the design notation in OOHDM provides a rich set of primitives for showing the way in which navigational objects (nodes and links) are related with conceptual ones (database objects, for example). Though being object-oriented, the method can be easily accommodated to deal with other implementation settings such as relational databases.

Navigational Contexts are also used in OOHDM as design primitives; in fact the overall structure of a web (or more general hypermedia) application is specified as a set of navigational contexts [12]. In OOHDM, the interface is also specified using a set of design primitives, Abstract Data Views (ADVs) [11]. ADVs allow expressing both static and dynamic aspects of the interface using a simple object-oriented model that uses composition and nesting of interface objects as first-class primitives. Some interface patterns that involve complex interface objects interactions (such as those appearing in Interface on Demand) may require intricate diagrams. Fortunately, the object-oriented nature of ADVs allows us to incorporate these interface patterns either as additional design primitives or as textual annotations in the interface model.

There are thus two ways of incorporating design patterns into the process of designing and implementing Web applications. The first is by enriching existing methods with higher level primitives that allow expressing some patterns with a single notation. The other is to use patterns as active guidelines in the design enterprise.

Due to the very nature of patterns, for each application domain the set of applicable patterns may vary. Their introduction has a great impact within the whole design process, providing a framework of good and proven design schemes rather than a set of constraints, while maintaining the freedom to innovate on stronger basis. Several questions arise at this moment:

- When do patterns come into play?
- How are patterns applied in the design?
- Should all patterns be applied?

Patterns influence the creation process from the very beginning during the conceptual process. There are two patterns that, even though they are navigational patterns, influence the domain or conceptual modeling stage: "Node as a Single Unit", where it is suggested that each node comprises all related information, and "Node as a Navigational View" because data usually is stored on relational databases. The combination of these two patterns helps identifying entities involved, and the way they are related to each other, regardless of whether these entities belong to an object-oriented model or a relational schema. This is a brief example of how patterns influence is present before design decisions are made.

Often, the serious constraint imposed by a limited amount of displayable information, leads to mistakenly scattering information on the basis of "screens" and forcing the conceptual model to reflect this issue. When we use an interface design solution like "Information On Demand", we have a clear understanding about which are the design objects, which shows which "pure" interface objects we need in order to make our interface manageable to the user. Note that in this case, when we apply the design pattern we improve our understanding about what is navigation and what is an interface effect (see for example figure 6).

The process of applying patterns in the context of a given problem is not linear. It is not a process of addition, in which pre-formed parts are combined to create a whole, but a process of unfolding that goes step by step, one pattern at a time. It is clear that within the iterative set of modeling tasks and the consequent deeper understanding of a domain model, the design may pose new problems, and thus the use of other patterns as new requirements appears. Because of this, it is not possible to establish beforehand which patterns are to be used given the fact that their adoption is ruled by the creational activity (which in turn, might be influenced by other patterns). In figure 8 we present a diagram showing the development process, some of the patterns involved, and the relationships among them.

The discovery of patterns is the consequence of reflection on design. The key idea behind them is to abstract the core solutions that can be implemented in several ways on different platforms but aiming to address the same problem with the same strategy. Good patterns are usually very simple, some of them may be have been inadvertedly used for years.

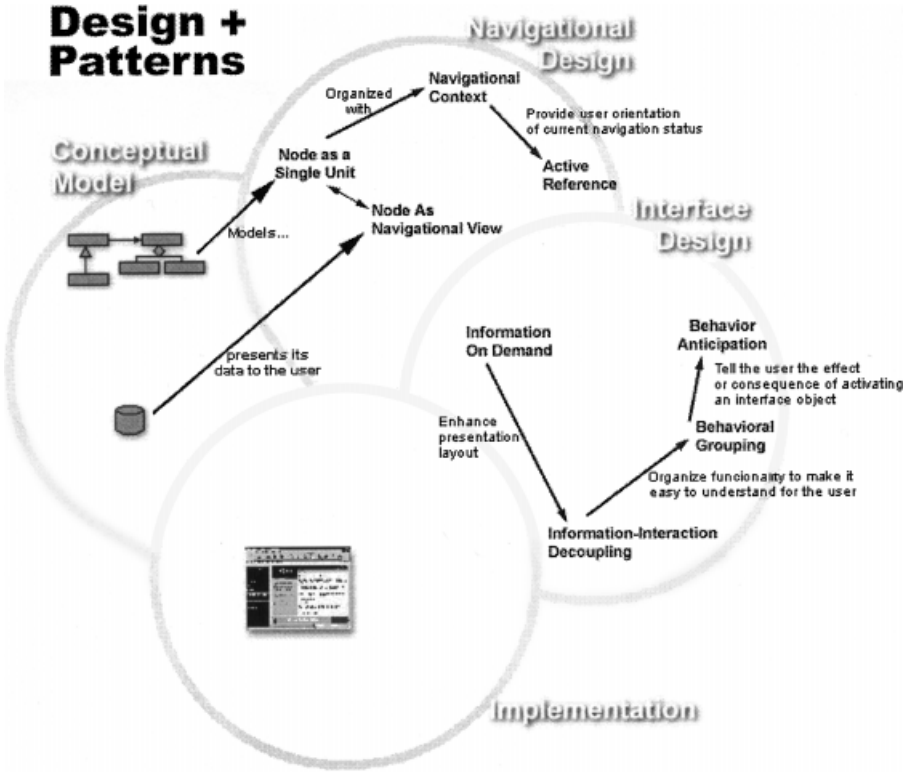

Figure 8. Diagram of the OOHDM design activities+Design Patterns and interactions between different patterns.

5. Concluding remarks

We have discussed the use of design patterns in web applications; a simple taxonomy: architectural, navigation and interface patterns were introduced. We exemplified our ideas describing some important patterns that are usually found in web applications. We briefly discussed the way in which patterns may be incorporated in the design process. Navigation and interface patterns represent an appealing way for recording existing design experience in the hypermedia or computer-human interface communities, and for transmitting this experience to the Web community, Nevertheless, we still think that more comprehensive examples of real applications developed by re-using design patterns are needed. The Web, meanwhile is a good place for discovering good solutions to recurrent problems, even for more general interactive applications.

Whereas navigating through the Web to find good navigation or interface structures has always been a common practice for designers, design Patterns help to add more reflection to this, sometimes anarchic, process. Discovering a new pattern and finding that it is clearly recurrent in the Web gives the designer a richer vocabulary to express his design structures. We think that this is a collaborative activity that should lead us to a set of standard patterns

accepted by the community as, it happened with [5] in the field of object-oriented design. Our work is a step towards that direction.

We are now working in the development of a catalogue that includes patterns and pointers to their known uses in the web. We believe that while patterns are essential in the development process, real-world examples showing their applicability are the key to understanding why and how patterns should be applied.

References

1. C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel, A Pattern Language, Oxford University Press: New York, 1977.
2. Ma. Bichler and S. Nusser, "Modular design of complex Web-applications with SHDT"; `http://dec9.wu-wien.ac.at/w3dt/wetice/wetice.html`.
3. B. Scheiderman, "Designing information-abundant web sites: issues and recommendations," IJHCS Journal, October 10, 1997.
4. `http://WWW.cnet.com/Content/Builder/Graphics/Design/ss1a.html`
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design patterns: Elements of reusable object-oriented software, Adisson Wesley, 1995.
6. A. Garrido, G. Rossi, and D. Schwabe, "Pattern systems for hypermedia," in Proceedings of PloP'97, Pattern Language of Programming, 1997.
7. F. Garzotto, D. Schwabe, and P. Paolini, "HDM—A model based approach to hypermedia application design," ACM Transactions on Information Systems, Vol. 11, No. 1, pp. 1–26, Jan. 1993.
8. T. Isacowitz, E. Sthor, and P. Balasubramanian, "RMM: A methodology for structured hypermedia design," Communications of the ACM, Vol. 38, No. 8, pp. 34–44, 1995.
9. G. Meszaros, "A pattern language for improving the capacity of reactive systems," in pattern Languages of Program Design II, Addison Wesley, 1996, pp. 575–591.
10. Jakob Nielsen, "The alertbox: Current issues in web usability"; `http://WWW.useit.com/alertbox/`
11. G. Rossi, D. Schwabe, C. Lucena, and D. Cowan, "An object-oriented model for designing the human-computer interface of hypermedia applications," in Proceedings of IWHD'95, Springer Verlag Workshops in Computing, 1996.
12. D. Schwabe, G. Rossi, and S.D.J. Barbosa, "Systematic hypermedia application design with OOHDM," Proceedings of Hypertext'96 (HT96), Washington, March 1996.
13. Some examples of websites designed with OOHDM: **Portinary**: `http://WWW.portinari.org.br/`; **Embratel**: `http://WWW.embratel.com.br`; **Sebrae**: `http://WWW.sebrae.org.br`
14. The VisualWave Programming Environment. Parc Place Systems; `http://WWW.parcplace.com/products/vwave/vwv-prod.htm`.

Fernando Lyardet is an advanced student of the Licenciatura en Informática, that has been working in the fields of hypermedia and object orientation since 1992, first as a development group member and later as member of the research team at LIFIA. While his focus is on the design methodology and design patterns for hypermedia systems, he has been leading the development of a CASE tool for the OOHDM methodology. Other topics of study are application frameworks.

Gustavo Rossi's group at Universidad Nacional de La Plata has worked on hypermedia design methods since 1994. Together with Daniel Schwabe, they have developed the Object-Oriented Hypermedia Design Method (OOHDM) and a set of design heuristics and patterns for building high quality hypermedia applications. He is now working on integrating design patterns into design methods.

Daniel Schwabe is a researcher at Depto de Informática in PUC-Rio, Brazil. He works in design methods for hypermedia applications since 1990 when he developed HDM together with people from Politecnico de Milan, Italy. He is now working in design approaches for Web applications.