Towards a Pattern Language for Hypermedia Applications

G. Rossi (*), D. Schwabe and A. Garrido (**)
Departamento de Informática. PUC-RIO, Brazil
E-mail: [rossi,schwabe]@inf.puc-rio.br

(*) Also LIFIA, Fac. Cs. Exactas, UNLP, Argentina and CONICET
(**) LIFIA, Fac Cs. Exactas, UNLP, Argentina
E-mail: garrido@sol.info.unlp.edu.ar

Abstract

This paper presents two design patterns for the hypermedia domain: 'Navigational Contexts' and 'Information on Demand'. They are applied in two different aspects of *hypermedia applications design*: the design of healthy navigational structures and the design of understandable and usable hypermedia interfaces, respectively. These two patterns are part of an effort for developing a Pattern Language for that domain.

1. Introduction. Designing High Quality Hypermedia Applications

Hypermedia applications provide the user with navigational access to an information base. These applications are usually built by specifying a set of nodes related through links. Nodes represent unstructured information such as text, images, video and animations and provide navigation anchors that are usually perceived in the interface as buttons, hotwords or other reactive interface objects. A successful hypermedia application allows a final user to perform a task exploring the navigational space by finding suitable paths to the desired information, without suffering disorientation or cognitive overhead.

In a previous work we presented two design patterns discovered while building a hypermedia framework [Rossi+96]. The framework allows constructing *hypermedia applications* as extensions of object-oriented (OO) information systems. This architecture is characterized as a *hypermedia system* because it is aimed at creating hypermedia applications. The problem that remained in this building process was giving the designer the suitable guidelines for building a well-designed hypermedia extension.

Furthermore, this problem applies in a wider context of general hypermedia application design: there is not a "design culture" around hypermedia though most design problems are well-known and have been repeatedly reported in scientific journals and meetings. Even when other non-OO hypermedia systems are used for creating hypermedia applications (e.g. Toolbook, HTML-based authoring tools, etc.), it should be possible to apply well-known software engineering practices during hypermedia design and implementation. The World Wide Web is a good source to find both well and poorly designed hypermedia documents.

In the last two years we have been developing the Object-Oriented Hypermedia Design Methodology (OOHDM) [Schwabe+95, Schwabe+96] which aims to provide a set of design models and guidelines to build high quality hypermedia applications.

Our purpose in this paper is to show how we can apply the ideas underlying patterns and pattern languages to the design of well-structured hypermedia applications. The paper focuses on two simple and powerful patterns: *Navigational Contexts* and *Interface on Demand*. We briefly explain the ideas behind OOHDM and then present both patterns.

2. Context. The OOHDM

The OOHDM considers the hypermedia application development process as a four-activity process that are performed in a mix of iterative and incremental styles of development; in each step a model is built or enriched.

During *Conceptual Modeling*, a model of the application domain is built using well-known OO modeling principles [Rumbaugh+91]. The product of this step is a class and instance schema built out of sub-systems, classes and relationships.

One of the key distinguishing features of hypermedia applications is the notion of navigation. In OOHDM, an application is considered as a navigational view over the conceptual model. This view is built during *Navigational Design,* taking into account the types of intended users, and the set of tasks they are supposed to perform using the application. Different navigational models may be built for the same conceptual schema, thus expressing different views (applications) on the same domain.

We describe the navigational structure of a hypermedia application by defining navigational classes that reflect the chosen view over the application domain. A set of base classes such as Node, Link and Index is provided. Nodes are described by two types of attributes: media attributes and anchors. Media attributes contain the information to be presented to the user while anchors are the origin of links. Anchors are usually implemented at the interface level as reactive objects that, when stimulated, trigger navigation. Links are described by their origin and target nodes.

During *Abstract Interface Design* an interface model is built. The interface model specifies which interface objects the user will perceive, and in particular, the way in which different navigational objects will look like, which interface objects will activate navigation, the way in which multimedia interface objects will be synchronized and which interface transformations will take place. In OOHDM interface objects are defined as composite objects reflecting the usual nested structure of hypermedia interfaces. A node is perceived through many (lower level) interface objects such as text or image fields, buttons, etc., as commonly found, for example, in WWW pages.

Finally, by mapping the navigational and abstract interface models into concrete objects, i.e., those available in the chosen implementation environment, the author produces the actual hypermedia system to be run. This is called *Implementation activity*.

In this paper we present two patterns that will help hypermedia designers in the activities of Navigational Design and Abstract Interface Design respectively.

The first pattern, Navigational Contexts, helps in organizing the navigational structure of the application in a clear and meaningful way for the intended readers. This is a recurrent problem over which the degree of success of hypermedia applications usually lies [Garzotto93].

The second pattern provides simple means of solving the problem of GUI organization. Most of the people can agree that the key to creating a good interface lies on the combination of elements (both in their number and in their spatial relations). Following, some guidelines may be given for a good organization of information, in such a way that elements interact in a effective manner (as Alexander's architectural elements). Information on Demand is one pattern we propose in that respect.

3. Design Patterns for Hypermedia

3.1 Navigational Contexts

Intent:

Provide the user with closed navigational subspaces containing context-related guidelines and relationships, so that information can better be presented and comprehended. The same node is also allowed to look different when seen as part of different contexts.

The Problem:

Hypermedia applications usually involve dealing with collections of nodes (e.g., 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 Paintings of an author, Paintings on a certain subject, etc., and it is desirable to give the user different kinds of feed-back in different contexts, while allowing him to move easily from object to object. From a design point of view this situation may arise for example when traversing 1-to-n links.

Suppose that in an application about Painters and Paintings we reach "Van Gogh", we choose to navigate to his paintings, and we then arrive at "Sun Flowers". However, we can also reach "Sun Flowers" while exploring Paintings about nature. It is clear that we will explore the same object under two different perspectives; for example while accessing it as a Van Gogh's work we would like to read some comments about its relationships with other paintings by Van Gogh and also to have easy access to the next painting he painted. Meanwhile, as a painting on nature, it would be fine to read (or see) something about that subject and be able to access other paintings on the same theme (perhaps not Van Gogh's). This means that we will need not only to present the information in a different way in both cases but also to provide different links or indexes.

One possible solution would be defining subclasses of Painting Nodes according to the context in which they will be explored and define corresponding links for implementing navigation inside a context. One of the drawbacks would be that a Painting will need to change its class when shown in a different context, which should not be a concern of class Painting.

Solution:

A better solution is to decouple the navigational objects from the context in which they are to be explored, and define objects' peculiarities as Decorators [Gamma+94], that enrich the navigational interface when the object is visited in that context.

OOHDM proposes *Navigational Contexts* as a way of organizing navigation. Navigational Contexts are composed of a set of Navigational Objects, i.e., Nodes (like Paintings) and Context Links (links that connect objects in a context). Nodes are further subclassified in Application Nodes, that contain the core of data to be shown in all possible views, and NodeInContext, that fulfills the role of a decorator for Nodes, adding 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 Paintings on Nature.

Getting back to the "Van Gogh" example, we may think about 'Painting' and 'Author' as subclasses of 'ApplicationNode'. We will have two possible navigational contexts 'Theme-related context' and 'Author-related context'. Accordingly, we may subclassify 'NodeInContext' with 'NodeInThemeRelatedContext' and 'NodeInAuthorRelatedContext', each subclass defining the

pertaining information that a painting will show on that context, and anchors for the 'previous' and 'next' painting on the same context. A simplified sketch of the example is shown in Figure 1.

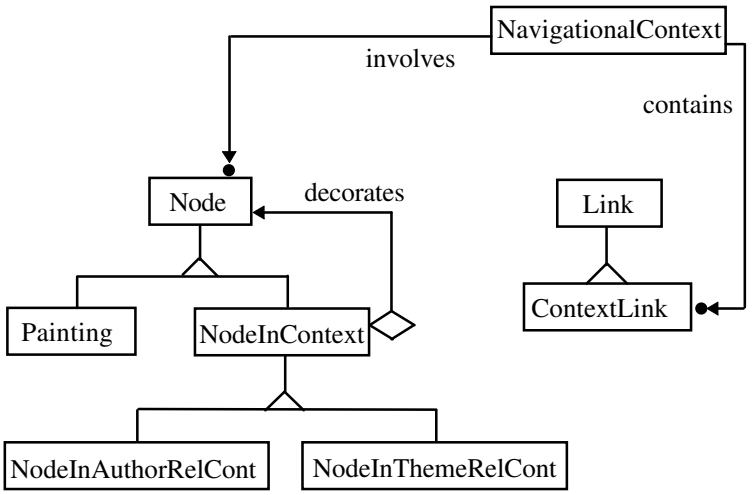

Figure 1: Example of Navigational Contexts pattern

Finding navigational contexts is important because they are high level architectural constructs that help organizing navigation and we can describe the navigational structure of a hypermedia application not only as a set of nodes and links but also as a set of contexts in which those nodes will be accessed.

We have identified different criteria for finding navigational contexts:

- *Class derived contexts*, i.e. instances of a node class fulfilling certain condition: all paintings about a subject, all paintings of a certain period, etc.;
- *Link derived contexts*, as the set of all targets of a Link when a link is 1-to-n;
- *Composite derived*, made out of all parts of a composite node, and
- *Arbitrary,* i.e. those navigational contexts built opportunistically by picking different nodes (perhaps from different classes). Arbitrary Contexts are quite useful when we want to provide the reader with a guided tour on some information items. For example, in "Le Louvre" (a popular hypermedia application about the famous French museum), we can make a tour exploring the most interesting aspects about the Museum's history and artworks in it.

Structure:

The structure of the pattern Navigational Context is presented in Figure 2 below. Main participants in this schema are NavigationalContext, Node, NodeInContext and ContextLink. NavigationalContext may be in fact a subclass of Node, conforming the Composite pattern [Gamma+94], though we draw it separately for the sake of clarity.

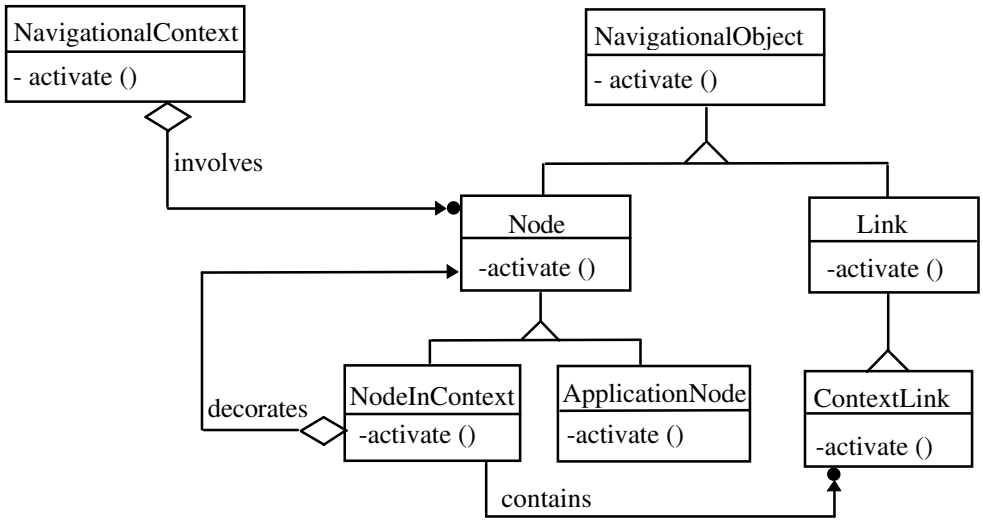

Figure 2: Navigational Contexts pattern

Participants:

A **navigationalContext** object contains a collection of nodes, connected by a collection of ContextLinks. NavigationalContext sub-classes implement different strategies for accessing elements in the context. For example IndexContext provide an index while SequentialContext provides a link for accessing the first element in the context. Other subclasses may define information about the context itself.

NodeInContext class is a decorator for **Node**. We may define subclasses of NodeInContext providing particular anchors or attributes for each element in the context. For example, NodeInContextIndex may only provide an anchor to the containing navigationalContext object and then navigation will proceed only by navigating via the index. Meanwhile, NodeInContextSequential provides two anchors "next" and "previous" for easy traversal of the elements in the context. We may further define sub-classes providing both kinds of anchors and other attributes related to the application semantics. Different node classes will be implemented as subclasses of ApplicationNode.

ContextLinks are Links that can be traversed only in the context they have been defined. They implement relationships that are only meaningful within that context.

Collaborations:

The collaborations among objects in this pattern can be easily understood when analyzing a particular interaction scenario. Let us follow with the example stated in the Problem section. Suppose that we want to open the node “Sun Flowers” in the “Author-Related Context”. The corresponding decorator object (an instance of class NodeInAuthorRelatedContext) will be first activated by the navigationalContext object. It will therefore delegate to the decorated node the responsibility of showing its contents and anchors for non-context links, and will add to them the corresponding anchors for that context. Navigation could be performed inner to the context, i.e., by selecting a context anchor that will cause a context link to be traversed, and another node belonging to the authorRelatedContext to be opened, or to an outer context, by selecting a link that connects

"Sun Flowers" with outer nodes. The following interaction diagram represents the activation of the node for "Sun Flowers in the "Author-Related context", and the selection of a context anchor.

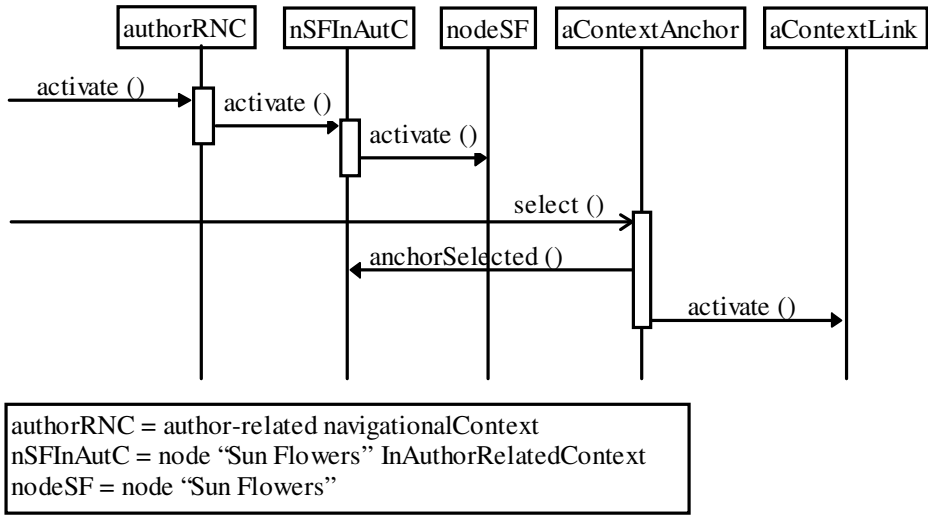

Figure 3: Navigational Contexts component's interactions

Consequences:

When organizing navigation with this pattern the set of navigable objects is partitioned in groups of closely related objects (paintings on the same theme, comments on a painting, documents related with the same painting, etc.), that are supposed to provide the user with an intuitive and more comfortable navigational path. From a pure design point of view, the design documents remain more compact using contexts as a structuring primitive than when implementing them at a lower level.

Implementation:

Navigational contexts can be easily implemented in hypermedia environments, even those not supporting full object-orientation. However, we may have to take into account features of the implementation environment; for example when using Toolbook we can use groups to organize information that will be meaningful only in a context and the script openInContext will use the corresponding group. Meanwhile in the WWW things are more difficult; we can choose to concatenate the decorator object and the decorated one for each pair decorator/decorated or build the corresponding page dynamically.

Known Uses:

Navigational Contexts has been used in many successful hypermedia applications. For example in Microsoft's Art Gallery, paintings of the London National Gallery are presented in different categories: by place, period and subject. A different interface is used when a painting is accessed in each context. In Portinari Project [Lanzelotte+93], a collection of most important painting of Brazilian painter Candido Portinari, different families of navigational contexts have been defined: for example for each technique we can reach all paintings drawn with that technique, for each Painting we can explore documents related with that painting, etc.. In [Garzotto+95]

Navigational Contexts are called “Collections”, and are described as an implementation construct rather than as a design concept, as we regard them here.

Related Patterns

NodeInContext acts as Decorator [Gamma+94], while NavigationalContext acts as a Facade to the set of objects belonging to the context.

This pattern may take great advantage of computed links and lazy creation of links endpoints, issues provided by the NavigationStrategy pattern [Rossi+96].

3.2 Information On Demand

Intent:

Organize the information presented to the reader in a legible and comprehensible interface, by letting him decide which items he wants to see further descripted, but without implying navigation. It does neither enforce authors to separate the information in different nodes.

The Problem:

We usually find ourselves struggling to decide how to show the attributes and anchors in a node in such a way that both esthetic and cognitive aspects are taken into account. 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.

When a node has a lot of information to be displayed we could define more than one page for the same node or use scrolling, but often this alternative does not yield an acceptable solution. Another alternative is to partition the node in such a way that we can use different windows for presenting the information by defining links among these new nodes. The problems with this solution are: first, in our attempt to match design with an implementation issue, we may define an attribute of a node as a separate entity, and in doing so we will end polluting the application's navigational structure. Consequently, the user will get the impression of dealing with multiple entities, becoming disoriented, while in fact he is accessing another part of the same conceptual entity.

One elegant solution is to present only a sub-set of the attributes and let the user decide if he wants more information by providing him with interface objects (e.g. buttons) controlling which information is presented in the screen. This just follows the "What you see is what you need" principle.

Solution:

Select the most important attributes (including anchors) and make them perceivable in the interface. Provide a set of interface objects (such as buttons) to let the user select other information to be shown. Use the same screen area to display different attributes.

In Figures 4.1 and 4.2, Information On Demand is shown in a practical example. Figure 4.1 shows a screen with an artwork in Portinari application. When the user clicks on “history”, “description” or “reference” he gets the (textual) contents of corresponding node’s attributes as shown in Figure 4.2. The attribute may further contain other "hotwords" (anchors) providing navigation to another hypermedia object.

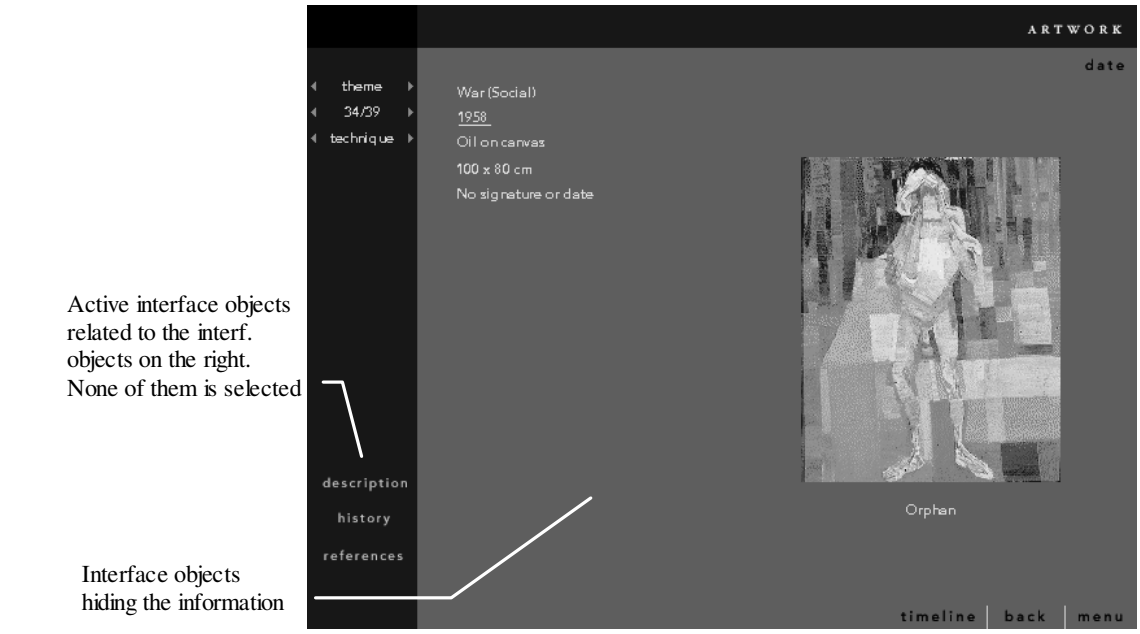

Figure 4.1: Active interface objects deselected

Figure 4.2: An active interface object is selected

Structure:

We next present the structure of Information On Demand with a class diagram in Figure 5, showing the nested structure of interface objects, the composition of nodes in terms of attributes, and the way attributes can be shown according to the user's demand.

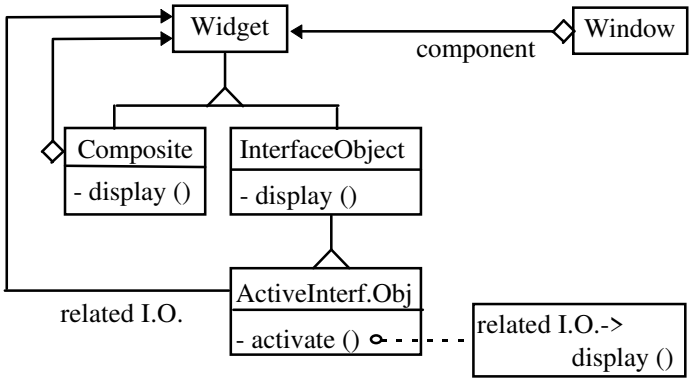

Figure 5: Information by demand pattern structure

Participants:

A window, as usual, is composed of widgets that can be nested following the Composite pattern [Gamma+94]. **Widgets** are nodes' Observers. The atomic widgets are **interfaceObjects** showing particular attributes of the node. A special kind of interfaceObjects, **activeInterfaceObjects**, reacts to external events (as mouse clicks) to show or hide a related interfaceObject containing the information. It would be possible to define a subclass of ActiveInterfaceObject that redefines the *activate()* message, for widgets that contain the information by itself and also react to external events, or for widgets that represent the interface of anchors and as such will trigger navigation by communicating with the corresponding node. It is also possible of course to define different methods for reacting to other events such as mouse on, mouse double clicked, etc.

Different subclasses of InterfaceObject (as TextInterfaceObject, GraphicInterfaceObject, etc.) will specialize the *display()* message.

Collaborations:

When an activeInterfaceObject receives the message *activate()* it communicates with the related interfaceObject asking it to *display()* itself. Display may be implemented by showing some textual information, playing an audio or video, starting an animation, etc. Complex collaborations among different interfaceObjects may be defined by making more complex implementations of the activate and display methods. For example we may allow two different interface objects to be displayed together (a text and a sound for example) but disallow others (see for example Figure 4.1 and 4.2). In both cases we may add responsibilities for controlling these situations either to the activeInterfaceObject or to the interfaceObject to be displayed. The same activeInterfaceObject may be also used to start and stop a display by sending the *display()* or *hide()* message according to its state.

Consequences:

When we use the Interface On Demand design pattern we have to clearly separate navigation objects from their interface objects in a way similar as when the Observer design pattern is used.

The Interface On Demand pattern raises an interesting discussion about what is navigation and what is an interface effect, i.e., when we navigate to another object and when we are just producing

an interface effect. This issue is interesting as it helps us to find "good" navigational objects by abstracting the way in which they are to be shown to the final user.

Implementation:

Information On Demand may be implemented in a number of different ways (involving perhaps other interface patterns). One possible approach is to provide cues about the objects we will show on demand; this is a common solution when dealing with large images. In the previous example (Figure 4.1 and 4.2), the small picture on the right acts as an active interface object such that when it is clicked we can see a full screen image of the artwork. In some cases we may need to use several pages to show a node's attributes. For example in “Le Louvre”, the user may choose to see a picture using a zooming tool. Both the "normal" image and the zoomed one are attributes of the same object: the painting; however we may get only one of them at a time.

Known Uses:

We have found the Information by Demand pattern in many different hypermedia applications. In Microsoft's Art Gallery it is used to provide further reading about a Painting. When you click the "See Also" button you get a set of related issues that are in fact attributes of the current painting representing anchors to the desired information. In Le Louvre, Information On Demand is used to provide different views on the same painting; for example we can see a small image together with information about its painter, a zoomed image that we can explore with a magnifying glass tool or see a simple animation on the creation process. All three alternatives represent attributes of the same navigational object. In the Portinari application, as previously shown, Information On Demand is used to provide access to different textual attributes of an artwork, or anchors leading to related objects.

Related Patterns:

Widgets use the Composite pattern [Gamma+94] for its nested organization.

4. Conclusions

In this paper we extend our previous work [Rossi+96] on finding recurrent problems in the design of complex hypermedia applications. In that opportunity we presented two design patterns, Navigation Strategy and Navigation Observer, used in the construction of a hypermedia *system* implemented as an OO framework. Users of that framework find them useful in order to understand its design and achieve better instantiations. Both patterns may be even used in the previous step of Navigational Design of OOHDM, while deciding to use fixed, computed, dangling or other strategies with which a link class will specify its endpoint (Navigation Strategy) or for configuring the desired record of navigation (Navigation Observer).

In this opportunity the patterns address the construction of hypermedia *applications*. They are intended to help both, users of the hypermedia framework that lack of hypermedia design experience, in order to instantiate it in a meaningful way, and hypermedia designers, to deal with recurrent problems of general hypermedia application design. Although this patterns are independent of the hypermedia system finally used, they require a minimum of flexibility for implementing them. For instance Navigational Context needs context links destinations to be computed dynamically, or created “just-in-time”. These are features provided by the Navigation Strategy pattern.

In our research and development activity on hypermedia design we have found many other interesting design patterns describing solutions to different navigation and interface problems. They are the ground of a Pattern Language for hypermedia applications we are developing from our experience in the field.

5. References

[Gamma+94] E. Gamma, R. Helm, R. Johnson and J. Vlissides: "Design Patterns: Elements of reusable object-oriented software", Addison Wesley, 1995.

[Garzotto+93] F. Garzotto, D. Schwabe and P. Paolini: “HDM - A Model Based Approach to Hypermedia Application Design”. ACM Transactions on information Systems, 11 (1), Jan. 1993, pp. 1-26.

[Garzotto+95] F. Garzotto, L. Mainetti and P. Paolini: "Hypermedia Design, Analysis and Evaluation Issues", Comm ACM, August 1995, pp 74-86.

[Lanzelotte+93] R.S.G. Lanzelotte, M.P Marques, M.C.G. Penna, J.C. Portinari, I.D. Ruiz and D. Schwabe: "The Portinari Project: Science and Art team up together to help cultural projects". Proceedings of the 2nd International Conference on Hypermedia and Interactivity in Museums (ICHIM'93), Cambridge, UK, September 1993

[Rossi+96] G. Rossi, A. Garrido and S. Carvalho: "Design Patterns for Object-Oriented Hypermedia Applications". Pattern Languages of Programs 2, Vlissides, Coplien and Kerth *eds.*, Addison Wesley, 1996.

[Rumbaugh+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W.Lorensen: "Object Oriented Modeling and Design", Prentice Hall Inc. 1991.

[Schwabe+95] D. Schwabe and G. Rossi: The Object-Oriented Hypermedia Design Model. Communications of the ACM, August 1995, pp 45-46.

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