Web Application Models are more than Conceptual Models

Gustavo Rossi [1,3], Daniel Schwabe[2], Fernando Lyardet[1]

[1] LIFIA, Departamento de Informática, UNLP. Argentina
E-mail: [gustavo,fer]@sol.info.unlp.edu.ar
[2] Depto de Informática, PUC-Rio, Brazil.
E-mail: schwabe@inf.puc-rio.br
[3] also at CONICET and UNLM

Abstract. In this paper, we argue that web applications are a particular kind of hypermedia applications and show how to model their navigational structure. We motivate our paper discussing the most important problems in the design of complex Web applications. We argue that if we need to design applications combining hypermedia navigation with complex transactional behaviors (as in E-commerce systems), we need a systematic development approach. We next present the main ideas underlying the Object-Oriented Hypermedia Design Method (OOHDM). We show that Web applications are built as views of conceptual models. We next present the abstraction primitives we use to design the conceptual and navigational structure of Web applications and describe the view definition language. We introduce navigational contexts as the structuring mechanism for the navigational space. Some further work on designing Web applications with OOHDM is finally presented.

1 Introduction: Web Applications are Hypermedia Applications

The emergence of the World Wide Web has made the hypertext paradigm more popular than ever. Web applications combine navigation through a heterogeneous information space with operations querying or affecting that information.

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 indefinitely. This succession of steps is know as “navigation”, and is intrinsic to hypertext, and hence to the WWW.

However, this second generation of hypermedia applications is rather different from the first one, in which applications, usually delivered in CD-ROMs, were not supposed to be updated and, in general, were not critical for any organization. Web applications, on the other hand, are constantly modified, are permanently enriched with new services, and new navigation and interface features are added, e.g., according to the organization’s marketing policy.

In this paper we argue that good Web applications should be, first of all, good hypermedia applications, i.e. they should provide easy navigational access to large information resources, preventing users from being lost in the cyberspace, and providing consistent navigation operations even when other kind of transactional behavior is involved. As navigation problems have been largely discussed in hypertext literature (see for example [12]) we should be able to reuse existing knowledge on building good Web applications.

Unfortunately, state-of-the art conceptual modeling approaches neglect navigation modeling as they do not provide useful abstractions capable of easing the task of specifying applications that embody the hypertext metaphor. For example, they do not provide any notion of linking and very little is said about how to incorporate hypertext into the interface. For example, we could easily model the domain of an electronic commerce application using UML [UML97]. However we can not specify critical aspects for this kind of application, such as which nodes will be navigated or which paths or indexes the application will contain. Even if we specify all this hypermedia functionality using UML primitives, we will be using low-level primitives whose semantics were not intended to model navigation.

At the same time, we could model this kind of applications by considering navigation as just another kind of interface behavior; this is the approach followed by some recent (object-oriented) tools like VisualWave [27]. In this case, applications built using the well-known model-view-controller interface metaphor are published in the Web by just translating views into HTML pages; only some aspects related with concurrent access with shared databases are taken into account. However this approach fails to consider the most powerful feature of the Web: its linking capabilities.

If we want to profit from the potential of the Web platform we need to consider both aspects of Web applications: navigation and transactional (or other kind of conventional) behaviors.

Web applications provide a powerful mechanism for building different views (in fact navigational views) to corporate databases. For example, while customers access the Amazon.com bookstore using a particular Web interface, managers or technical staff can access the same information resources through a different Web application (and obviously different access rights) in an Intranet. However, these views are more than simple database views as they involve different navigation paths, indexes, etc. In this paper we show how to design Web applications as views of (shared) conceptual models. In addition, it will be argued that the links provided for navigation are more than a representation of conceptual relations, as a more naive approach would suggest.

To summarize the discussion above, we can intuit that there are distinguishing features in Web applications that present new design requirements vis-a-vis traditional systems. In a broad sense, we can categorize them in three groups. The first group of design issues has to do with navigation, addressing questions 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?
- 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 questions such as:

- What 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 questions 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?

In this paper we will concentrate on discussing our approach for solving the first group. A discussion on the third group of issues can be found in [Segor99].

The rest of this paper is structured as follows: we first introduce the Object-Oriented Hypermedia Design Method. We next discuss how we build navigational models as views on conceptual models; then we introduce navigational contexts as a structuring mechanism for navigation. Finally, we discuss some ongoing work on mining navigation patterns and present some further work on designing and implementing these kind of systems.

2 The OOHDM Design Framework

The Object-Oriented Hypermedia Design Method [Schwabe 98, Schwabe 96] is a model-based approach for building large hypermedia applications. It has been extensively used to design different kinds of applications such as: Web sites and information systems, interactive kiosks, multimedia presentations, etc. It should be stressed the OOHDM has been applied outside the academic environment, such as in government agencies, telecommunications companies, oil companies, IT service companies, etc…

OOHDM comprises four different activities namely, Conceptual Design, Navigational Design, Abstract Interface Design and Implementation. During each activity a set of object-oriented models describing particular design concerns are built or enriched from previous iterations.

We explicitly separate conceptual from navigation design since they address different concerns in Web applications. Whereas conceptual modeling and design must reflect objects and behaviors in the application domain, navigation design is aimed at organizing the hyperspace taking into account users' profiles and tasks. Though applications views are not new in the literature [4], the hypermedia paradigm as it appears in the Web raises additional concerns such as orientation, cognitive overhead, etc. that should be treated in a separate design activity.

Considering conceptual, navigational and interface design as separate activities allows us not only to concentrate on different concerns at a time, but mainly to obtain a framework for reasoning about the design process, encapsulating design experience specific to each activity. As we explain below, navigational design is a key activity in the implementation of Web applications and it must be explicitly separated from conceptual modeling

OOHDM design primitives can be mapped onto non object-oriented implementation settings using some simple heuristics [22]. We next discuss the first three activities in more detail.

2.1 Conceptual Modeling

During this step we build a model of the application domain, using well known object-oriented modeling principles and primitives similar to those in UML [24]. The product of this step is a class schema built out of Sub-Systems, Classes and Relationships.

We chose UML because it is a modeling standard whose syntax and semantic are clear and well-understood. The major differences with UML are the use of multiple-valued attributes, and the use of directions explicitly in the relationships. Aggregation and generalization/specialization hierarchies are used as abstraction mechanisms.

Conceptual Modeling is aimed at capturing the domain semantics as "neutrally" as possible, with little or no concern for the types of users and tasks. When the application involves some sophisticated behavior in conceptual objects, it may evolve into an object model in the implementation environment. However it can be implemented in a straightforward way in current Web platforms combining for example a relational database with some stored procedures. The main thesis in this paper is that the conceptual model may not reflect the fact that the application will be implemented in the WWW environment, since the key application model will be built during navigational design. This view allows using the same strategy for implementing "legacy" applications in the Web, by considering their conceptual model as the product of this OOHDM activity.

Classes in the conceptual model will be mapped to nodes in the navigational model using a viewing mechanism and relationships will be used to define links among nodes. It will also be shown that there are other links that do not correspond to relationships in the conceptual model.

Using a behavioral object-oriented model for describing different aspects of Web applications allows to express a rich variety of computing activities, such as dynamic queries to an object-base, on-line object modifications, heuristics-based searches, etc. The kind of behavior required in the conceptual model depends upon the desired

features of the application. For many Web applications, in particular those implementing plain browsing (i.e. read-only) functionality, class behavior beyond linking functionality is unnecessary and does not need to be specified.

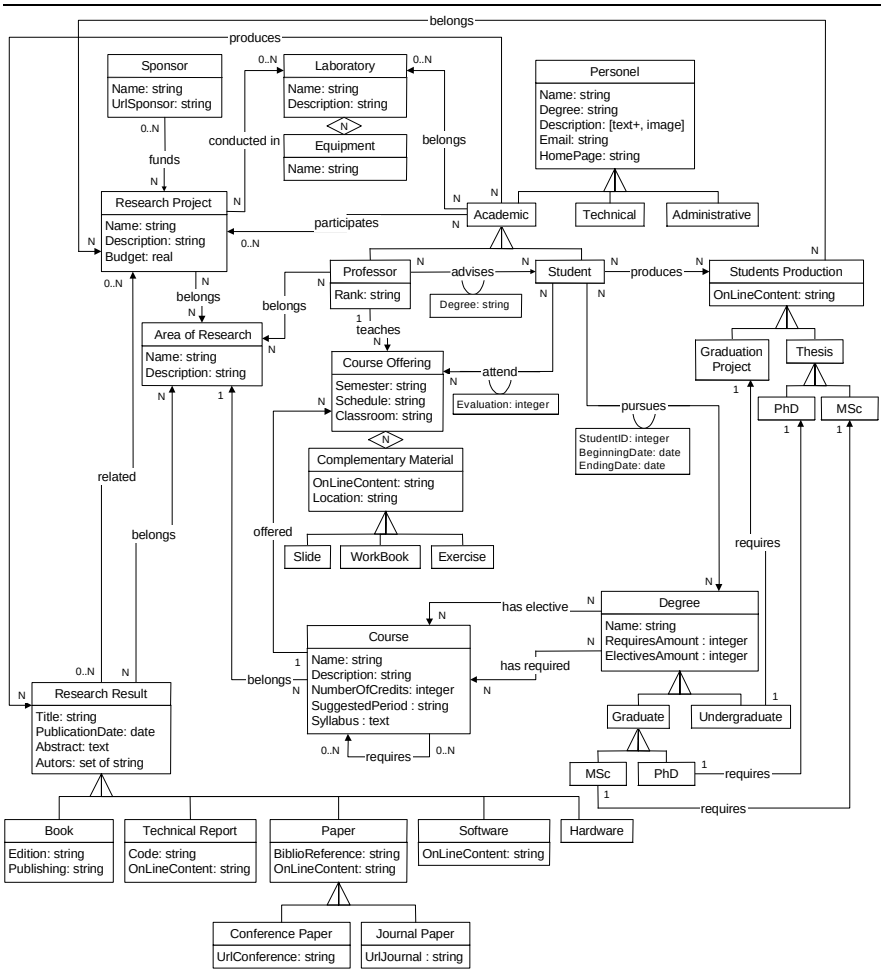

Fig. 1. Conceptual Schema of the Academic Department Site.

Figure 1 shows the Conceptual Schema for an Academic Department Web site. Perspectives (multiple valued attributes) are denoted by enumerating the possible types, with a + next to a default type. Thus, *description: [text+, image]* means that attribute *description* has a text perspective (always present), and may have also an image perspective.

2.2 Navigational Design

In OOHDM, an application is seen as a navigational view over the conceptual model. This reflects a major innovation of OOHDM with respect to other methods, at it recognizes that the objects (items) the user navigates are *not* the conceptual objects, but other kinds of objects that are "built" (through a view mechanism) from one or more conceptual objects. Moreover, it is important to stress that the user navigates through links, many of which cannot be directly derived from conceptual relationships.

For each user profile we can define a different navigational structure that will reflect objects and relationships in the conceptual schema according to the tasks this kind of user must perform. The navigational class structure of a Web application is defined by a schema containing navigational classes. In OOHDM there is a set of pre-defined types of navigational classes: nodes, links, anchors and access structures. The semantics of nodes, links and anchors are the usual in hypermedia applications. Access structures, such as indexes, represent possible ways for starting navigation.

Different applications (in the same domain) may contain different linking topologies according to the user's profile. For example, in the Academic Web site we may have a view to be used by students and researchers, and another view for use by administrators. In the second view, a professor's node may contain salary information, which would not be visible in the student's view. In section 3 we detail how we specify nodes and links using a view definition language.

The most outstanding difference between our approach and others using object viewing mechanisms is that while the latter consider Web pages mainly as user interfaces that are built by "observing" conceptual objects, we clearly favor an explicitly representation of navigation objects (nodes and links) during design.

2.3 Abstract Interface Design

In the Abstract Interface Design activity, we specify which interface objects the user will perceive and how the interface will behave. For each node's attribute (either contents or anchors) we must define its appearance. By distinguishing between navigation and interface design we can build different interfaces for the same application and besides achieve implementation independence.

During this activity we define 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, we use the Abstract Data View (ADV) design approach for describing the user interface of a hypermedia application [3]. Though not completely related with the aim of this paper, it is important to stress that building a formal model of the interface of Web applications is a rewarding activity as user interfaces tend to change even faster than navigation topologies. We clearly need a precise design specification to be able to support changes smoothly. A complete description of our approach for specifying user interfaces can be found in [14].

2.4 Implementation

During the implementation activity we map conceptual, navigation and interface objects onto the particular runtime environment being targeted. When the target implementation environment is not fully object-oriented, we have to map the conceptual, navigational and abstract interface objects into concrete objects, i.e. those available in the chosen implementation environment. This may involve defining HTML pages (or, for example, Toolbook objects in non Web-based environments), scripts in some language, queries to a relational database, etc. Notice that even in object-oriented environments like VisualWave [27] there may be no significative difference among conceptual and navigation objects which will act as models of Smalltalk's interfaces. Meanwhile, in a more "hybrid" environment, conceptual objects will be mapped to a persistent store (files or relational databases) while the interface and navigation objects will be implemented as conventional Web pages.

In the following sections we discuss the OOHDM approach for defining the navigational structure of Web applications.

3 Specifying Navigational Objects as Views on the Conceptual Schema

One of the cornerstones of the OOHDM approach is the fact that most navigational objects (nodes and links) are explicitly defined as views on conceptual objects and according to each different user profile. These views are built using an object-oriented definition language that allows to "copy and paste" and/or filter attributes of different (related) conceptual classes into the same Node class and to create Link classes by selecting the appropiate relationships.

In the academic site example we may want that nodes representing Courses contain an attribute with the name of the Professor that teaches that course, an eventually use that name as an anchor to the Professor's home page. It is clear that in the conceptual model the name of the professor is an attribute of Class Professor and should not be included in Class Course. Meanwhile, in a different view, we may want to filter some attributes (such as the professor's salary, for example) or include new relationships as links.

Node classes are defined using a query language similar to the one in [10]. Nodes possess single typed attributes, link anchors, and may be atomic or composite. Anchors are instances of Class Anchor (or one of its sub-classes) and are parameterized with the type of Link they host. The object-oriented nature of nodes and anchors allow re-defining their opening and activation semantics allowing customization to different application domains.

The syntax for defining Node classes is shown next:

NODE name [FROM className: varName] [INHERITS FROM nodeClass]

attr1: type1 [SELECT name1] [FROM class1:varName1, classj: varNamej WHERE logical expression]

attr2: type2 [SELECT name2]...

...,

attrn: typen [idem]

....

anch1: Anchor [linkType1]

anch2: Anchor [linkType2]

...

END

Where

- *name* is the name of the class of nodes we are creating.
- *className* is the name of a Conceptual Class (from which the node is being mapped). It is called the Subject class.
- *nodeClass* is the name of the super-class
- *attr* are the names of attributes for that class, type the attribute's types.
- *name* are the subjects for the query expression and var are mute variables used to express logical conditions.
- -*logical expression* allows defining classes whose instances are a combination of objects defined in the conceptual schema when certain conditions on their attributes and/or relationships hold.
- -*anch* are names of anchor variables. Anchor is the abstract class for all anchors
- *linkType* is a link type qualifying anchors.

Nodes implement a variant of the Observer design pattern [5] as they express a particular view on application objects. Changes in conceptual objects are broadcasted to existing observers while nodes may communicate with conceptual objects to forward them events generated in the interface.

As an example we would define the Node class CourseOffering including as one of its attributes the name of the professor who teaches it and an anchor for the link that connects both nodes. We say that the conceptual class CourseOffering is the subject of Node class CourseOffering. Notice that in OOHDM we defer the decision of defining the anchor's appearance until the abstract interface design activity.

NODE CourseOffering [FROM CourseOffering:C]

professor: String [SELECT Name] [FROM Professor:P WHERE P teaches C]

.... *(other attributes "preserved" from the conceptual class CourseOffering}*

taughtBy: Anchor [TaughtBy]

Links connect navigational objects and may be one-to-one or one-to-many. The result of traversing a link is expressed by either defining the navigational semantics procedurally as a result of the link's behavior, or by using an object-oriented state transition machine similar to Statecharts. Since Web applications usually implement simple navigation semantics (closing the source node and opening the target), we do not discuss this issue further.

Access structures (such as indices or guided tours) are also defined as classes and present alternative ways for navigation in the application. Application Links are also defined as views on conceptual relationships (see the discussion on Context Links in section 4). Access structures are usually defined in Navigational Contexts (see Section 4), and they are specified by defining the target navigational objects and the selectors (usually attributes of the targets). The syntax for defining Link classes is shown below (we avoid describing link's attributes and behavior for the sake of simplicity).

LINK name

SOURCE: sourceNode: sourceVar

TARGET: targetNode: targetVar

WHERE logical expresion

END

- name indicates the name of the Link Class
- sourceNode is the name of the source Node class
- targetNode is the name of the target Node class
- sourceVar, targetVar are mute variables used in the logical expression
- logical expression indicates a condition that involves the Subjects of Source, Target and perhaps other conceptual classes.

Using the syntax above we may define the Link class TaughtBy as shown below (the qualifier "S." indicates the subject of the corresponding node classes, in this case CourseOffering and Professor). Notice that the conceptual relationship "taught by" between CourseOffering and Professor may not exist in the conceptual schema and we should carefully plan the final implementation of this view.

```
LINK TaughtBy

SOURCE: CourseOffering: c
TARGET: Professor: p
WHERE  S.p teaches S.c
END
```

The navigational schema contains a diagrammatic description of the relationships among nodes. Each navigational schema represents the model of a different Web application.

It is important to stress the similarities and differences among the conceptual and navigational schema. They are similar because both are abstract and implementation independent and they represent concepts of the underlying application domain using objects. However, while the former should be neutral with respect to navigation, the latter expresses a particular user's view (in the navigation sense) that is strongly influenced by the tasks he is supposed to perform.

OOHDM enforces a clear separation between the specification of navigation and other application's behavior. However, in complex Web applications it may be necessary to integrate both kinds of behaviors (electronic stores are a good example of this need). As nodes implements Observers they can communicate easily with their conceptual counterpart in order to delegate actions they can not perform (as for example, modifying a persistent store).

Nodes and Links are the basic primitives of Web applications. However we need higher level abstractions to build meaningful and usable navigation structures, since we may need to introduce nodes and links that do not reflect conceptual entities or relationships and that may be defined opportunistically for improving navigation. We next introduce Navigational Contexts, a powerful mechanism for structuring the navigational space.

4 Structuring the Navigational Space: Navigational Contexts

Web applications usually contain collections of pages dealing with similar concepts, e.g.: books from an author, CDs performed by a group, hotels in a city, etc. These collections may be explored in different ways, according to the task the user is performing. For example, in an electronic bookstore he may want to explore books of an author, books on a certain period of time or literary movement, etc. It is also desirable to give him different kinds of feedback in different contexts, while allowing him to move easily from item to item. For example, it is not reasonable that if he wants to explore the set of all books written by Shakespeare, he has to backtrack to the index (the result of a keyword search for example) to reach the next book in the set.

As a result of organizing navigation objects into sets, many navigation operations refer to intra-set navigation, most notably "next", "previous" and "up". Therefore, sets

define links that allow such navigations, and these links have no direct counterparts in the conceptual model. In other words, there is no conceptual relationship that directly translates into intra-set navigation links

Unfortunately, most modeling approaches ignore sets as first-class citizens and therefore operations such as “next” and “previous” are not usual while traversing sets. To make matters worse, the same node may appear in different sets: e.g. a book written by Shakespeare may appear in the set of Romantic books or in the set of books written in England. We may intend to include some comments about the book in the corresponding context, e.g.: when accessed as a romantic book, some comments about the role of the book in the romantic period.

OOHDM structures the navigational space into sets, called Navigational Contexts represented in a Context Schema. Each Navigational Context is a set of nodes and it is described by indicating its internal navigational structure (e.g. if it can be accessed sequentially), an entry point and associated indexes. Generally speaking, contexts are defined by properties of its elements, which may be based on their attributes or on their relations, or both. There are four special cases that occur more frequently when stating such properties to define contexts in OOHDM:

1 *Simple class derived* – includes all objects of a class that satisfy some property ranging over their attributes; e.g., “ professors with rank = associate”, “paintings with painter= Van Gogh.

Graphically:	

2 *Class derived group* – is a set of simple class derived contexts, where the defining property of each context is parameterized; e.g. “professors by rank”, “paintings by painter” (rank and painter can vary). Graphically, same as 1

3 *Simple link derived* – includes all objects related to a given object; e.g., “courses taught by Professor Smith”, “exhibitions where Sun Flowers was presented”. Graphically, same as 1.

4 *Link derived group* - a set of link derived contexts, each of which is obtained by varying the source element of the link; e.g. “courses taught, by professor”, “exhibitions by painting” (professor and painting can vary). Graphically, same as 1.

Besides the context definition forms above, there are the following additional forms:
5 *Arbitrary* - The set is defined by enumeration. For example, a guided tour showing some pictures in a museum or some outstanding research projects. Graphically, same as 1.

Contexts may also vary during navigation, either because the reader can create or modify information elements (navigation objects), thus affecting the elements of a derived context, or because they can explicitly insert or remove objects to the context. Such contexts are said to be dynamic; examples are history and shopping baskets.

Graphically:	

In any of the above, if there is an access structure defined for it, the corresponding graphical notation contains a small black square in the upper left corner.
Associated to the contexts, there are access structures (indices). They are denoted graphically by:

Simple Index:	
Dynamic Index:	
Index with multiple orderings:	

The Navigational Context Schema represents contexts and their access structures. In Figure 2 we show the context schema for the academic site. Notice that for each Node class (i.e. Student, Professor, Research Result, Research Project, Laboratory, etc) we have indicated different kinds of contexts and indexes, such as the Main Menu, the Personnel Category Menu, etc. Arrows indicate both navigational relationships and possible transitions among navigational contexts.

When the same node (e.g. Research Project, Professor, etc.) may appear in more than one set (context) we need to express the peculiarities of this node within each particular context. We may take as a default that “next” and “previous” anchors and links are automatically defined for traversing each set; but we may also want that some context-sensitive information appears when accessing a Professor by research area (for example giving access to the papers he wrote in that area).

In OOHDM this is achieved with InContext classes; for each Node class and each Context in which it appears, we can define an InContext class that acts as a Decorator [5] for nodes when accessed in that particular context. Decorators provide a good alternative to sub-classing, and prevent us from defining multiple sub-classes of the base Node class. InContext Classes are organized in hierarchies with some base classes already provided by the design framework; for example InContext classes defined as sub-classes of InContextSequential inherit anchors for sequential navigation and for backtracking to the context index. When we do not define InContext classes, a default one is assumed according to the type of context defined. It is important to stress that (similarly to context links) InContext classes are not directly mapped from the conceptual schema as they address a “pure” navigational concern.

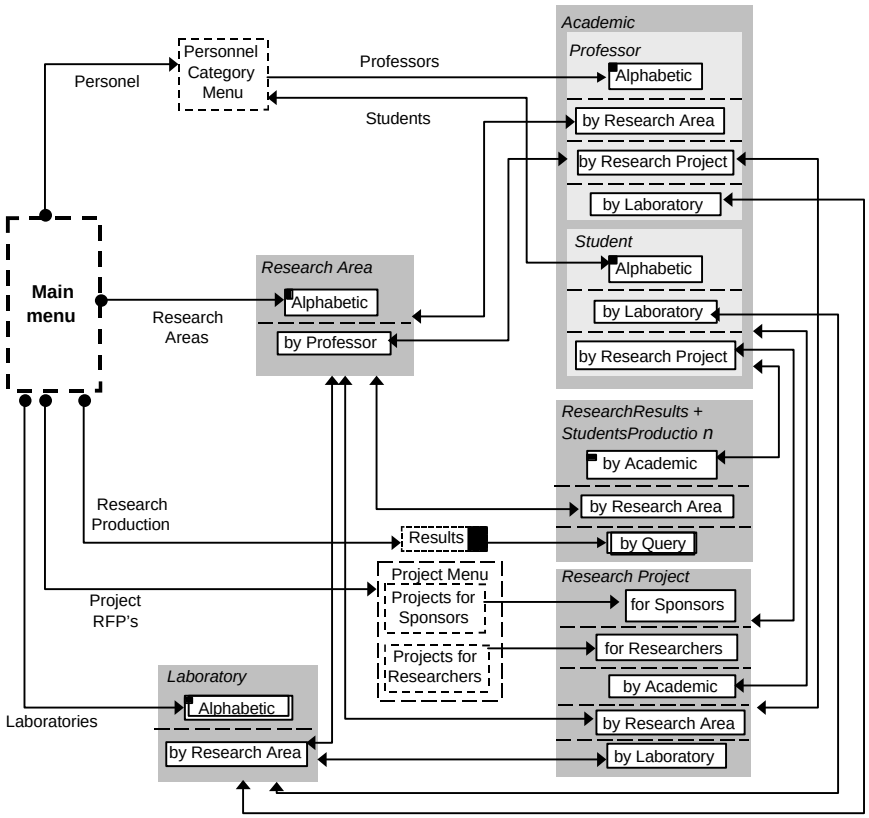

Fig. 2. Navigational Contexts in an Academic Web site.

The Navigational Contexts Schema complements the Navigational Schema by showing the way in which nodes are grouped into navigable sets. Additional nodes behavior can be implemented in InContext classes. In Amazon.com for example when we access a book in the context of a query we have an option to move it to the shopping basket. When we access the same book in the context of the shopping basket we should have other different operations to perform. In the example of the academic website, presenting a "Research Project" for a sponsor will show different attributes than those presented when showing the same project to another researcher.

Navigational Objects (nodes, links, contexts, indexes, etc) are documented using a set of cards (like CRC cards [Wirfs-Brock 90]) that provide complete information for implementers.

Though OOHDM does not pre-suppose a particular implementation strategy for mapping contexts to a run-time setting, there are many alternatives whose main differences are the amount of “intelligence” in client pages and/or the Web server (see [22] for a discussion). Taking into account the increasing trend towards “object-orienting” the Web [8], implementing complex navigational structures in Web

applications directly with object technology is already feasible, for example using Java (see, for instance, [Pizzol 99]).

5 Concluding Remarks and Further Work

In this paper, we have argued that we need several different design models for building Web applications. We have presented the OOHDM approach that comprises four different activities, namely: conceptual modeling, navigational design, abstract interface design and implementation. We have focused on the design of the navigational structure of Web applications, and have shown that these applications are built as views on conceptual models. We have described navigational contexts as a structuring mechanism for improving navigation. In OOHDM the navigational schema describes which classes will be navigated and how, whereas the navigational contexts schema provides additional information when dealing with collections of nodes.

As in most complex design domains, just a method (and a set of modeling primitives) is not enough for coping with the inherent complexity of Web applications. We need to understand which recurrent problems we solve and be able to reuse good design solutions to those problems. Design Patterns [5] are a good strategy for recording and communicating design expertise about recurrent problems.

During the last four years we have been mining design patterns in the hypermedia field: we call them hypermedia patterns [15, 16, 18, 19]. We have identified many recurrent problems and well known design solutions and have recorded them in the form of patterns. Using these patterns we can improve our ability to build Web applications; we also simplify the navigational schema as we can use patterns as higher level constructs, thus reducing the number of connections we have among navigational classes.

As an example the Landmark navigational pattern indicates that when some sub-system should be easily reached from every node in the application, we should treat it in an special way and make it perceivable with a standard interface. Examples of Landmark can be seen for example in the Book, Music, Gift and Auction subsystems at Amazon.com or in global navigation bars in most Web applications.

When we identify a Node as being a Landmark it is understood that we would be able to navigate to that node from every page so we do not need to define all links pointing to the Landmark. In such cases not only do we obtain a well-behaved application, but we also simplify the navigational schema. Landmarks show another kind of links that are not derived from conceptual relationships but are defined opportunistically to reduce navigation effort and complexity.

We are now working in a project associated with ACM-SigWeb (the ACM special interest group on Hypermedia and the WWW) for building an online repository of hypermedia and Web patterns, together with examples, known-uses, implementations of those patterns, etc.

We are also enriching OOHDM by introducing the concept of Web applications frameworks (i.e. set of abstract design structures that can be instantiated for different applications in the same domain); we are defining a notation for describing and

instantiating frameworks [23]. In this way we can build even more abstract conceptual and navigational schemas that may comprise families of related Web applications. We believe that our approach may help to obtain greater levels of reuse in the design of Web applications, and therefore this will reduce development times and costs by simplifying evolution and maintenance.

6 References

1. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel:"APattern Language". Oxford University Press, New York 1977.
2. Bieber, M; Vitali, F.;"Toward Support for Hypermedia on the World Wide Web" *IEEEComputer* 30(1), January 1997. Also available at: http://www.cs.unibo.it/~fabio/bio/papers/1997/IEEEC97/January/IEEEC0197.html
3. D. D. Cowan; C. J. P.Lucena, “Abstract Data Views, An Interface Specification Concept to Enhance Design for Reuse”, IEEE Transactions on Software Engineering, Vol.21, No.3, March 1995.
4. M. Fowler: "Application Views: Another technique in the analysis and design armoury", JOOP, Vol 7 N 1., pp 59-66.
5. Gamma, R. Helm, R. Johnson and J. Vlissides: "Design Patterns: Elements of reusable object-oriented software", Addison Wesley, 1995.
6. H. Gellersen, R. Wicke, M. Gaedke: “WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle” Electronic Proceedings of The Sixth International WWW Conference, Santa Clara, USA, April, 1997.
7. A.M. Hester; R.C.Borges; R. Ierusalimschy; “CGILua: A Multi-Paradigmatic Tool for Creating Dynamic WWW Pages”, Proceedings of the XI Brazilian Software Engineering Symposium (SBES’97) pp.347-360, Fortaleza, Brazil, 1997 (available at http://www.tecgraf.puc-rio.br/~anna/cgilua/ cgilua.ps.gz)
8. IEEE Internet Computing. Special issue on Object-Orienting the Web. January/February, 1999.
9. R. Ierusalimschy, L. H. de Figueiredo and W. Celes, "Lua - an extensible extension language", Software: Practice & Experience 26 #6 (1996) 635-652. (see also http://www.tecgraf.puc-rio.br/lua/).
10. W. Kim, "Advanced Database systems", ACM Press, 1994.
11. M.C. Meré, G. Rossi: “Specifying navigational transformations in hypermedia. A temporal logic framework”. In Bodo Urban (ed) Multimedia’96. pp. 20-31. Springer Computer Science, Springer Verlag New York, 1996.
12. J. Nielsen: “Hypertext and Hypermedia”. Academic Press, 1990.
13. Pizzol, A.M; Schwabe, D., "A Java Framework for Implementing OOHDM Designs", Proceedings of the V Brazilian Symposium on Hypermedia and Multimedia (SBMidia 99), Goiânia, Brazil, May 1999 (In Portuguese).
14. G. Rossi; D. Schwabe; C.J.P. de Lucena; D.D. Cowan, “An Object-Oriented Model for Designing the Human-Computer Interface of Hypermedia Applications”, Proc. of the International Workshop on Hypermedia Design (IWHD'95), Springer Verlag Workshops in Computing Series. (available at <ftp://ftp.inf.puc-rio.br/pub/docs/techreports/ 95_07_rossi.ps.gz>).

15. 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.
16. 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.
17. G. Rossi and A. Garrido: "Capturing Hypermedia Functionality in an Object-Oriented Framework", to appear in Object-Oriented Frameworks, Wiley 1999.
18. G. Rossi, F. Lyardet and D. Schwabe: “Patterns for designing navigable spaces”To appear in Pattern Languages of Programs 4, Addison Wesley, 1999.
19. G. Rossi, D. Schwabe, F. Lyardet: “Improving Web information systems with navigational patterns” To appear in International Journal of Computer Networks and Applications, 1999.
20. D. Schwabe and G. Rossi:, “The Object Oriented Hypermedia Design Model”, Comm. of the ACM, Vol. 38, #8, pp45-46 Aug. 1995. (available at <http://irss.njit.edu:5080/cgi-bin/bin/option.csh?sidebars/schwabe.html>).
21. 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.
22. D. Schwabe, G. Rossi: “An object-oriented approach to web-based application design”. Theory and Practice of object Systems (TAPOS), October 1998.
23. D. Schwabe: “Just Add Water” Applications: Hypermedia application frameworks. Proceedings of the 2nd Workshop on Hypermedia Development, Darmstad, February 1999. Available at: http://ise.ee.uts.edu.au/hypdev/ht99w/submissions/SchwabeHT99Workshop.pdf.
24. UML Document Set. Version 1.013 January, 1997, Rational, 1997. (available at http://www.rational.com/uml/references/index.html)
25. C. Varela, D. Nekhayev, P. Chandrasekharan, C. Krishnan, V. Govindan, D. Modgil, S. Siddiqui, , D. Lebedenko, M. Winslett: “DB: Browsing Object-Oriented Databases over the Web”. Proceedings of the Fourth International World Wide Web Conference. pp. 209-220, 1995.
26. The Visual Work Programming Environment. Parc Place-Digitalk, 1996.
27. The VisualWave Programming Environment. Parc Place Systems. In http://www.parcplace.com/products/vwave/vwv_prod.htm.
28. R. Wirfs-Brock et al: “Designing Object-Oriented software”. Prentice Hall, 1990.