Abstraction and Reuse Mechanisms in Web Application Models

Gustavo Rossi*, Daniel Schwabe and Fernando Lyardet ***

*LIFIA Facultad de Informática. UNLP.
La Plata, Argentina
{gustavo, fer}@sol.info.unlp.edu.ar

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

Abstract. in this paper we analyze different abstraction and reuse mechanisms that should be used in Web applications to improve their evolution and maintenance. We first review the OOHDM approach for defining a Web application model, in particular the separation of the navigational model from the conceptual model. We next focus on abstraction and composition mechanisms in both models showing how to combine OOHDM's views with the concept of node aggregation. We introduce navigation and interface patterns and show the way in which patterns generate the architecture of Web design frameworks. We strongly argue that in the currently state of the art of Web applications we can build models of families of similar applications to improve design reuse. Next, we present our notation for specifying Web frameworks, giving some examples in the field of E-commerce. Some further work is finally discussed.

1 Introduction

Building complex Web applications is a time consuming task as they must provide navigational access to critical information resources, not only allowing the user to browse through the potentially large universe of information but also to operate on it. In some areas such as electronic commerce, customers' actions trigger a sophisticated workflow that must be integrated with the core business software. The first obvious consequence is that we must not only design the navigational architecture carefully but also integrate it effectively with the underlying business model.

To complicate matters, Web applications should be developed with zero defects, with short deployment and maintenance times. In this context, we should use not only systematic engineering techniques but also be able to improve reuse during the whole development cycle. The key for obtaining reusable designs or components is to be able to build extensible and reusable conceptual models. However, while reuse techniques have been widely explored for conventional applications [Meyer94], the very nature of Web applications seems to prevent designers from being able to cope with design and implementation reuse.

The purpose of this paper is to present different design reuse mechanisms that should be used while building Web application models. We stress mechanisms such as navigation patterns and Web frameworks, particularly those that apply to Web applications, such as contexts. In this sense, our goal is not to present novel design primitives, though we introduce some, like aggregates and generic contexts; we rather seek to motivate discussion on the problem of reuse in Web application models.

Though we use OOHDM [Schwabe98, Rossi99b, OOHDM00] as the base design method, the ideas in this paper can be easily applied to other modeling approaches. In section 2 we characterize Web application models as the combination of conceptual and navigational models. In section 3 we show how different abstraction and composition mechanisms in OOHDM work together to achieve elegant and reusable design models. In section 4, we briefly address abstract design reuse by reviewing navigation patterns. Since patterns generate architectures, we go further in section 5 and present Web design frameworks as a way to achieve reuse of entire domain models. In section 6 we present OOHDM-Frame, a notation for specifying Web design frameworks. Some further work is finally discussed.

2 Web application models: Conceptual + Navigation models

The key concept in OOHDM is that Web application models involve a Conceptual and a Navigational Model [Rossi99b]. The conceptual model aims at capturing the domain semantics using well-known object-oriented primitives and abstraction mechanisms. In an electronic store for example, the conceptual model will contain core classes such as Product, Order, Customer, etc. with their corresponding behaviors. We use UML as the notation to specify the conceptual model. Since the conceptual model is an object-oriented model, we can use existing reuse approaches in object-orientation [Fayad99, Meyer94].

In the OOHDM approach the user does not navigate conceptual objects but navigation objects (nodes). Nodes are defined as views on conceptual objects, using a language that is similar to OODB view-definition approaches [Kim90]. Nodes are complemented with links that are themselves specified as views on conceptual relationships. The navigational schema shows the node and link classes that comprise the navigational structure of the application. For each particular user profile we build a navigational model as a view of the shared conceptual model. In this way, we can reuse the conceptual model in a family of similar applications. Moreover, as shown in section 3, we can define different views in the context of a single application.

In Fig. 1 we show part of the conceptual model of an electronic store. Notice that some classes in the model will be mapped onto the navigational model (i.e. they will be explored as nodes) while others, such as PaymentMethod, will not.

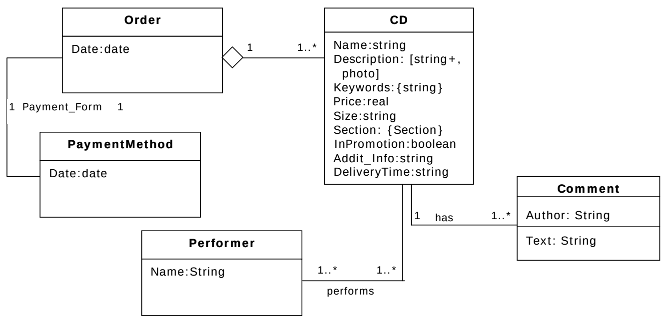
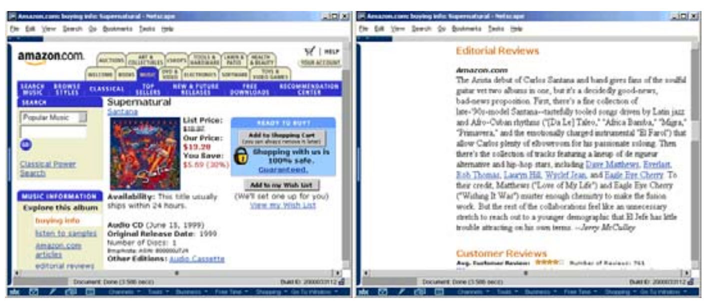


Fig. 1. Conceptual Model of CD store.

If we are designing the customer view of the electronic store, we will specify node classes for products. As shown in Fig. 2, these nodes may combine some attributes of conceptual class CD with attributes from conceptual class Comments and Performer. Notice that in good object-oriented software specifications (such as the one in Figure 1), products, comments and performers belong to different classes -. Nodes meanwhile implement opportunistic views of conceptual classes (following the Observer design pattern [Gamma95]). The precise syntax for defining views can be found in [Rossi99b].




```
Node CD FROM CD:C
    name: String, price: Number
    performer: String SELECT name FROM Performer: P WHERE C isPerformed by P
    comments:Array[Text] SELECT text FROM Comment: R WHERE
        C hasComment R
    other attributes and anchors
```

Fig. 2. CDs including comments in Amazon.com and the OOHDM definition.

The Navigational Schema is complemented in OOHDM with a Context Schema that shows the navigational contexts and access structures (indexes) in the application. A navigational context is a set of objects that are usually explored sequentially; for example: Books of an author, CDs by a rock band, etc. There are different kinds of navigational contexts: class derived, link derived, arbitrary, etc [Schwabe98]. Access Structures act as indexes to group of related objects; they are specified by indicating the target objects and the selector to be used in the index.

In Figure 3 we show part of the context schema for the electronic store. The notation in Figure 3 shows in a compact way, which sets the user will explore, and how they are related with each other. Navigational contexts are a novel design primitive for specifying sets in a concise way, specifically developed for exploring hyperspaces.

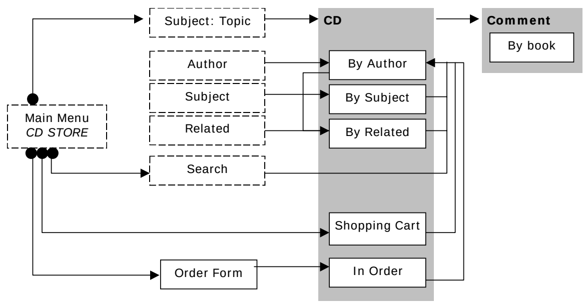


Fig. 3. Context Schema for the CD store.

Dashed boxes in Figure 3 show access structures (indexes) while boxes inside Class CD (and comment) indicate possible contexts in which a CD (respectively a comment) can be accessed. A node may appear in different contexts, showing different information according to the context within which it is reached. In this situation, we use Decorators [Gamma95] to decouple the base information in the node from the different "faces" this node exhibits. Consequently, navigational contexts combine two navigational patterns, Set-based navigation and Nodes in Context [Rossi99a]. The navigational and the context schemas play an important role when reusing application models in a family of applications in the same domain. We will discuss this kind of reuse in section 6.

3 Combining views with aggregate nodes

Complex Web applications provide multiple ways of reaching the information they contain. In e-commerce applications for example customers receive different kinds of advising such as hot-lists, recommendations, new releases, etc. In Figure 4 we show an example of a home page that contains different kind of links to products in an electronic store. In OOHDM we can aggregate nodes to specify this home page. An aggregate allows gluing different information items (other nodes) and access structures (like indexes) in the same node.

Fig. 4. A node representing a home page.

The specification of part of the node for the home page in Figure 4 reads as follows:

Node MusicHome
news: Array [CDView]
search: SearchTool
categories: IndexOfCategories
topSellers: IndexOfTop
landmarks: IndexOfStores
...
other attributes

Node CDView FROM CD: C
name: String
performer: String SELECT name FROM Performer: P WHERE C *isPerformed by* P
description: Photo
shortComment: Text

Notice that the specification of type CDView above takes profit of the viewing mechanism and it can be reused in other parts of the site (for example the Artists Essentials section uses a similar summary for each CD). Aggregates allow specifying composite nodes in an opportunistic way (as it is usual in Home pages). However, aggregate nodes combine with the viewing mechanism in a way that goes beyond simple composition mechanisms in object-orientation. This synergy is complemented with the linking mechanism that allows different views of the same object to be connected with each other. For example you can easily navigate from the summaries of CDs in Figure 4 to the corresponding CD. In Figure 5 we show in a diagram how to reuse one object's view and how this view is linked to another one of the same object.

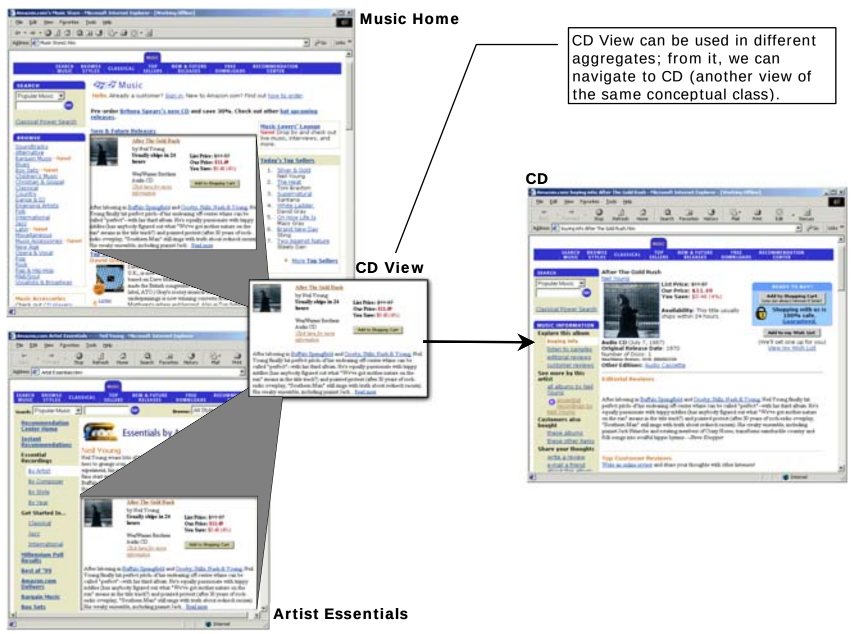


Fig. 5. Aggregates and view reuse in a navigational schema.

This simple example raises some interesting issues and questions related with design reuse:

1. Can we generalize the basic idea behind the previously shown home page? What design problem are we solving when building this kind of aggregate node? Can we apply this same solution in other Web applications?
2. Is the structure of this application similar to others in the same domain? In other words: how can we profit from our intellectual investment while designing the conceptual and navigational models in similar applications?

These questions show some non-trivial design reuse problems. While composition, viewing and inheritance allow improving reuse and maintenance in a single application, they are not enough for expressing reusable aspects in a family. We next introduce two novel approaches for design reuse in Web applications: navigation patterns and Web design frameworks.

4 Design Reuse using Navigation Patterns

Patterns record design experience by expressing in an abstract way recurrent problems and proven solutions. They are a wonderful tool for capturing, conveying and reusing design experience.Patterns complement design methods by showing solutions that go beyond naive uses of the methods' primitives. Patterns improve communication among designers by enriching the design vocabulary with terms that express non-trivial design structures. They formalize well-known solutions in such a way that novice designers can profit form experts' knowledge. We have mined patterns for Web applications and have documented them using a template similar to Alexander's one [Rossi99a]. In fact hypermedia and Web patterns are similar to the original urban patterns as they express recurrent structures for building usable navigable spaces; they show design solutions that help the user find his way through the hyperspace. The hypermedia community have proposed dozens of new patterns [Garzotto99], and it is now pursuing a project for expressing these reusable solutions in a shared catalogue [HypPatterns99].

Continuing with the previous example we may define two simple but effective patterns for dealing with (part of) the application's complexity: Portal and Landmark. We briefly describe them, stating the problem they address and the (widely used) solution.

4.1 Portal

In many Web applications, particularly in E-commerce we want to give the user a comprehensive description about what he will find in the site including daily news, suggestions, opportunities, etc. If we follow a naive hypermedia design view, the "home" page should map some conceptual object, or may just be an index to services or products. The solution is to design the home (or homes) as aggregates of different information items, anchors and access structures, Dedicating space to news, suggestions to the user, general indexes, special offers, etc. This home page may even contain information that may not be "semantically" connected. A portal is an opportunistic design solution that allows increasing the site's number of visitors as it is easier and quicker for them to find what they want. Portals are widely used in all e-commerce sites such as amazon.com, netgrocer.com and more general sites such as netscape.com. Portals generalize the design solution in Figure 4.

4.2 Landmark

Many Web applications contain sub-sites that provide specific functionality (different shops, search facilities, etc). When we describe the navigational schema (i.e. the network of nodes and links types), we try to follow closely those relationships existing in the underlying object model; for example we can navigate from an author to his books, from a CD to the list of songs it includes. However, we may want that at any moment the reader can jump to the music or book (sub) stores or to his shopping basket. The solution is to define a set of landmarks and make them accessible from every node in the network. making the interface of links to a landmark look uniform. In this way users will have a consistent visual cue about the landmark. We may have different levels of landmarks according to the site area we are visiting. Landmarks are different from indexes as they appear in every node in the application. This pattern is widely used in Web applications for indicating relevant sub-sites and functionality.

Patterns do not stand by themselves. They must be integrated into the development method in order to be effective. They must be combined to create higher level abstractions. In the context of OOHDM we have defined notations for some navigation patterns such as Set-Based Navigation and Nodes in Context [Rossi99b] and Landmarks [Rossi99a]. In Figure 6 we generalize the preceding example by showing a navigation model incorporating the idea of Landmarks. Notice that instead of a tangled diagram we get a simplified one in which links to landmarks are omitted. CD Store, BookStore and Toy Store in Figure 6 are Landmarks (indicated with an arrow with a bullet as source). Notice that, within CD Store, “Subjects”, “Search”, “Shopping Cart” and “Order” are second level Landmarks.

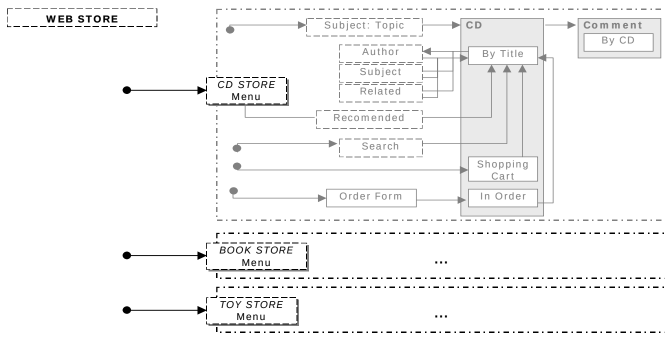


Fig. 6. Using Landmarks in the Navigational Schema.

Incorporating patterns into the design armory helps to reduce the complexity of diagrams thus making reuse more feasible. However, when we design complex applications we need more powerful reuse approaches.

In the e-commerce domain for example we can easily find that most virtual stores offer similar services to the customer: most of them allow finding products by searching or hierarchical navigation, all of them provide a shopping basket for making selections persistent, etc. Moreover we can find commonalties even in the core application behavior: for example, the set of actions triggered when a customer makes a check-out operation are basically identical: verifying user data, creating an order, sending a confirmation mail, sending another mail when products are shipped, etc. We should be able to define architectures that abstract these commonalties and that can be extended smoothly to cope with variations in each particular application. We next introduce Web design frameworks and show how they relate with navigation patterns.

5 From Web patterns to Web frameworks

Frameworks are reusable designs for a family of applications in a particular domain. They act as skeletons of a set of applications that can be customized by an application developer. When many different applications must be constructed in the same domain, application frameworks provide "templates" for supporting their commonalties, and accommodating individual variations (differences). While patterns provide abstract reuse of design experience, frameworks allow reusing concrete designs in a domain [Fayad99]. Frameworks are composed of a set of abstract and concrete classes, which contain the specification of generic behaviors (usually specified using a particular programming language) in the intended domain. A key aspect for designing frameworks is identifying its hot spots (i.e.: the points in the framework where variations will appear). Following with the preceding example, we can generalize the conceptual model (in Figure 4) to reflect abstract classes and collaborations in virtual stores. The model should include an abstract class Product, different kinds of Orders and Payment Methods, Comments, etc. A designer developing a particular store will need to define new concrete classes (for example sub-classes of Product) and specialize some behavior such as order processing, to accommodate it to the particular application (for example, selling other products using different business rules). In virtual stores (such as Amazon.com) the approach will work for defining new sub-stores in the company that may have, for example, different shipping or payment policies.

Designing frameworks is a difficult but rewarding task. We need to understand the domain and produce a generic design that can be instantiated into different applications. To apply this approach to Web application models, we need to take into account different kinds of variability: those related with the domain model (e.g. different payment policies) and those related with navigation architectures (e.g. different indexes, contexts, etc). Besides, programming-language-centric approaches (common in application frameworks) are difficult to apply in the Web, given the large number of combinations of languages and tools that are often used in Web application development and implementation.

We define a Web *design* framework as a generic design of possible Web application architectures, including conceptual, navigational and interface aspects, in a given domain. We have used the OOHDM model as the basis architecture for specifying Web design frameworks. Web design frameworks comprise a generic conceptual model (that may be itself an object-oriented framework), a generic navigation schema and a generic context schema.

Web *design* frameworks are different from *application* frameworks because while the latter are programmed in a specific language, Web *design* frameworks are environment and language-independent. Web design frameworks include an additional perspective with respect to conventional application frameworks: the generic navigation architecture.

Web design frameworks can be mapped either to an application framework to be later instantiated into a running application or can be instantiated into “pure” OOHDM models and then implemented as a single Web application [Schwabe00]. We next present a notation for improving Web application models with the kind of abstractions needed in Web design frameworks.

6 OOHDM-Frame: A notation for Web frameworks

In order to specify Web design frameworks, we have defined a new notation, called OOHDM-Frame that extends OOHDM smoothly. It is not our objective in this paper to give the detailed syntax of the notation but rather to analyze how to improve existing abstraction and composition mechanisms in conceptual modeling in order to express generic Web functionality. We will present the notation briefly to stress each particular modeling feature. As previously explained, the specification of a framework's model in OOHDM-Frame is comprised of generic Conceptual and Navigational Models specifications, together with instantiation rules. We next analyze each one pointing out novel abstraction mechanisms.

6.1 Abstraction and Genericity in the Conceptual Model

Variability in Web applications may appear in the conceptual model. In Figure 7 we show part of a generic model for electronic stores. Notice that we have included some abstract classes like Product and specialized Comment and Payment Method.

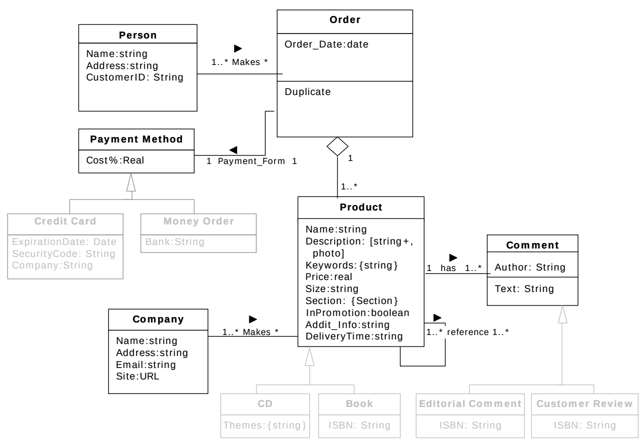


Fig. 7. A generic conceptual model for virtual stores.

Genericity in object-oriented models has been largely discussed in the object-oriented community and one can use existing notations to express generic classes and behaviors [Pree94], so we don't discuss it further here.

6.2 Specifying Generic Navigational Models

A generic Navigation Model in OOHDM-Frame is made up of a Generic Navigation Schema, a Generic Context Diagram, and a set of mapping and instantiation rules. The Generic Navigation Schema generalizes the idea of views (or observations in the terminology of [Gamma95]). It is similar to the Navigation Schema, except for the fact that Node attributes may be optional (marked with an "*") and Relations (links) can be optional (drawn with a dashed line), as shown in Figure 8. An optional attribute (respectively Link) may or may not appear in an instantiated application. Notice that as the navigational model will be often mapped into a non object-oriented implementation, we are not constrained to “pure” notations, e.g. we can always specify optional features (attributes or links) by defining appropriate class hierarchies, though in a less concise way. For the sake of simplicity we have not included those sub-classes in Figure 8.

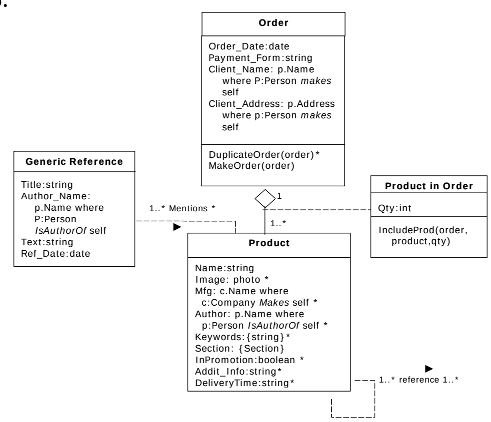


Fig. 8. Optional attributes and Links in the generic navigational schema

Sub classing in the Generic Navigational Schema allows a more subtle way of achieving genericity. In the example above, we may create a sub-class of Product and either add an attribute or anchor or we may even need to specialize the view specification for a particular attribute, as shown below.

Suppose for example that we have two sub-classes of Comment (as shown in the generic conceptual schema of Figure 7); if we want to generalize the store to a Books and CDs store (in the context of a framework for virtual stores), we may require that some of the navigational Product sub-classes show comments from only one (conceptual) sub-type. Accordingly, we show the specification of part of the abstract node class Product, and how we specialized the definition of the attribute *comments* for Books. The Refine operator takes the query in the corresponding super-class and replaces Comment with its sub-class EditorialComment. We are thus indicating that books only show Editorial Comments.

Node Product from Product: P
comments: Array[Text] SELECT text FROM Comment: R WHERE P *hasComment* R

Node Book from Book:B
REFINE comments WITH EditorialComments

Generic Context Diagrams meanwhile represent another kind of hot spot in Web applications, showing in an abstract way which contexts and access structures may appear in a particular domain. Notice that as navigational contexts are sets of nodes, defining generic contexts is equivalent to specifying generic sets. Thus, achieving generictiy in a context diagram is not straightforward with usual object-oriented abstraction mechanisms, i.e. though context and indexes may be finally mapped into classes, expressing their variability may require using complex diagrams. Instead, we preferred to generalize Context Diagrams and to complement them with a generic context specification card providing a guide for the implementers indicating possible restrictions. In Figure 9 we show a simplified generic Context Schema for our virtual store framework. Dashed boxes and rounded boxes indicate generic access structures and contexts. For example the generic context "Product by Property" will be typically instantiated into one or more contexts that allow navigation among products according to certain properties (e.g., "Product by price"; "Product by author"; etc…). Once within any of these, it is normally possible to navigate to other "Related Products" (e.g., accessories, matching products, etc…). There are several access structures that lead the reader into these contexts; typically, these are hierarchical access structures that reflect product sections (departments) in a real world store. Notice that we have also specified some Landmarks (like Shopping Card, Order Form and Search). A second way to look at products is within arbitrary groupings obtained opportunistically. Typically, these will correspond to some person's (or publication) recommendations, or some guide, such as "N.Y. Times Bestsellers List". This grouping is modeled through the generic context "Products by Reference".

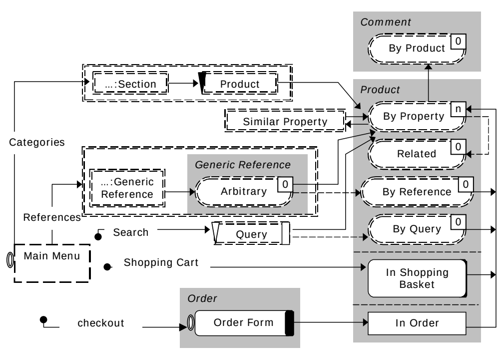


Fig. 9. Generic Context Schema for virtual stores.

The context diagram in Figure 3 is an instantiation of the generic diagram in Figure 9, where the generic context "Product by Property" has been concretized into "CD by Subject" and "CD by Author". Generic Context Schemas show concisely different ways of providing Set-based navigation in Web applications for a particular domain.

7 Concluding Remarks and Further Work

We have discussed in this paper different abstraction and reuse mechanisms in the context of Web applications. We have shown that even the simplest techniques like composition and inheritance offer subtle combinations to the designer when dealing with non-trivial navigation models. In particular, the OOHDM viewing language can be used synergistically with aggregation (and sub-classification) to produce compact and reusable navigation designs. We have discussed reuse of design experience by briefly analyzing navigation patterns. Although patterns provide design reuse at a fine granularity, we have shown how to combine them to obtain larger reusable models. We have introduced Web design frameworks, explaining how generic and reusable conceptual and navigational models can be described using the OOHDM-Frame notation. Web design frameworks show how the combination of patterns (like Set-Based Navigation, Landmark, and Observer) may yield a generic design for a family of applications in a particular domain.

Even though the focus of this paper has been put on design, it is important to stress that all primitives and mechanisms previously presented can be implemented using current Web technologies [Schwabe00]; in addition, mapping design frameworks to “pure” object-oriented settings is straightforward. We are mining Web patterns in specific domains such as e-commerce, and studying ways to enrich the framework design notation with these new patterns. Several implementation aspects should also still be studied, such as efficient ways to implement views and contexts in Web applications. Another aspect that we did not address is the use of support tools both for drawing diagrams and generating code. While UML tools like Rational Rose can be used we still have to build similar editors that support the slight syntactic and semantic differences among UML and OOHDM.

We believe that the growing interest in Web applications requires ways to build easily extendable and reusable conceptual models. Web applications present novel features that need to be considered in order to apply well-known abstraction and composition mechanisms to this new field. The ideas underlying this paper may serve as the background for studying abstraction and reuse in Web models.

8 References

1. [Fayad99] M. Fayad, D. Schmidt and R. Johnson (editors): “Building Application Frameworks”, Wiley 1999.
2. [Gamma95] E. Gamma, R. Helm, R. Johnson and J. Vlissides: "Design Patterns. Elements of reusable object-oriented software". Addison Wesley, 1995.
3. [Garzotto99] F. Garzotto, P. Paolini, D. Bolchini and S. Valenti: “Modelling by patterns of Web applications”. Proceedings of the First International Workshop on Conceptual Modeling and the WWW, Paris, France, November 1999,Lecture Notes in Computer Science, Vol. 1727, Springer, 1999, 293-306.
4. [HypPatterns99] Hypermedia Patterns repository: http://www.designpattern.lu.unisi.ch.
5. [Kim90] W. Kim, "Advanced Database systems", ACM Press, 1994.
6. [Meyer94] Bertrand Meyer, “Reusable Software” - The base object-oriented component libraries. Prentice Hall 1994.
7. [OOHDM00] Daniel Schwabe and Patricia Vilain: “The OOHDM notation”, available at http://sol.info.unlp.edu.ar/notacaoOOHDM/
8. [Pree94] W. Pree: “Design Patterns for object-oriented software”, Addison Wesley, 1994.
9. [Rossi99a] G. Rossi, F. Lyardet and D. Schwabe: “Patterns for designing navigable spaces” Pattern Languages of Programs 4, Addison Wesley, 1999.

10. [Rossi99b] G. Rossi, D. Schwabe, F. Lyardet: “Web application models are more than conceptual models”. Proceedings of the First International Workshop on Conceptual Modeling and the WWW, Paris, France, November 1999, Lecture Notes in Computer Science, Vol. 1727, Springer, 1999, 239-253.
11. [Schwabe98] D. Schwabe, G. Rossi: “An object-oriented approach to web-based application design”. Theory and Practice of Object Systems (TAPOS), Special Issue on the Internet, v. 4#4, pp.207-225, October, 1998.
12. [Schwabe00] D. Schwabe, G. Rossi, L. Emeraldo, F. Lyardet: “Web Design Frameworks: An approach to improve reuse in Web applications. Proceedings of the WWW9 Web Engineering Workshop, Springer Verlag LNCS, forthcoming.