Annals of Software Engineering 13, 97–110, 2002

Object-Oriented Design Structures in Web Application Models

GUSTAVO ROSSI gustavo@sol.info.unlp.edu.ar
LIFIA Facultad de Informática, UNLP, La Plata, Argentina

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

Abstract. In this paper, we discuss different object-oriented design structures that should be used in the process of building Web applications. We base our discussion on the OOHDM approach for defining a Web application model, in particular, the separation of the navigational model from the conceptual model. We focus on the systematic application of different design patterns (such as Observer and Decorator) for decoupling different aspects of a Web model. We briefly discuss some specific patterns that may appear in this kind of applications and we introduce additional concepts such as Web frameworks as a conceptual approach to maximize design reuse in Web applications.

1. Introduction

Engineering Web applications is a complex enterprise. The ubiquity of the Web has pushed forward a number of new requirements such as building interfaces for mobile devices, integrating Web applications with legacy database and systems, providing friendly navigable metaphors, etc.

To make matter worse, Internet businesses tend to be different from "conventional" ones; applications must be built quickly and with zero defects. Evolution happens in shorter time frames than before. Consequently, while most software engineering precepts still stand up, we must improve our development practices in order to make them efficient for this new environment.

It is interesting to notice that while implementation tools and architectures have evolved quickly during the last four years, software engineering strategies for the Web are still being developed. Moreover, if we look at Web architectures and we compare them with what we had some years ago, it is easy to notice that evolution continues at a good speed. Compare, for example, an implementation using HTML and JavaScript in the client side with CGI and a database in the server side, with more modern approaches using XML, J2EE and Servlets. Unbelievably, we already have legacy applications in the Web.

Considering this last comment, we strongly think that it is important to focus on using good design practices, primitives and notations in order to cope with changing implementation settings. Key activities for a Web application software engineer are to

understand the design decisions that must be made while building these applications and finding ways to describe and reuse high-level abstractions.

We have been exploring object-oriented design structures techniques in Web applications using the Object Oriented Hypermedia Design Method (OOHDM) for some years [Schwabe and Rossi 1998]. OOHDM considers Web applications as *navigational views* over an object model [Rossi *et al*. 1999] and provides some basic constructs for designing the navigational model (contexts, indexes, etc.) and the user interface model.

In this context we have done research on different aspects of abstraction, reuse mechanisms and strategies in the development process such as patterns and frameworks [Rossi *et al*. 2000a,b; Schwabe *et al*. 2001a,b]. Our purpose in this paper is didactic, as we aim to present different object-oriented structures that should be used while building Web application models, which we present mostly as modeling notation rather than as implementation constructs.

Some of these structures can be implemented in a straightforward way in most Web implementation settings, and as a whole, they are useful for obtaining sound, reusable design models. As a side effect, the discussion in this paper is also interesting as a way to reason on the development process of a Web software artifact.

The structure of the paper is as follows: we first introduce our view on the development process of Web applications, and our approach for building Web models as views of conceptual models. We next analyze the use of the Observer design pattern [Gamma *et al*. 1995] in the navigation model and how composition is handled in this model. Then we show how to use the Decorator design pattern [Gamma *et al*. 1995] both for achieving flexibility in the definition of views for different user profiles and for defining navigational contexts. We briefly discuss some design structures that appear while building personalized Web applications, and finally we discuss Web design patterns and frameworks. Concluding remarks are discussed in the end. Though we use OOHDM as the reference design method, the ideas in this paper can be easily applied to other modeling approaches. Since the intent of this paper is to reflect on some design structures rather than introducing a design notation (see [Schwabe and Rossi 1998]), we will only refer to the OOHDM notation if necessary.

2. Our view on the Web applications development process

Web applications allow users to interact with an implementation of a business model and eventually modify its state using a Web browser. However, Web applications are different from "conventional" applications not only because the interface platform is different but also because they introduce the concept of navigation, as it is used in hypermedia [Nielsen 1995], which is an inherent characteristic of the Web itself.

According to this view, Web application models comprise a conceptual, a navigational model, and an interface model [Rossi *et al*. 1999]. The conceptual model aims at capturing the application semantics and functionality using object-oriented primitives. The idea behind the conceptual model is that it helps to identify core abstractions and behaviors in the application domain. These may be later implemented using an object-

oriented language and become the basis for the specification of perceivable artifacts, nodes and links, in the Web application. The conceptual model being object-oriented, it is obvious that many key design structures in the whole application will be defined as part of this OO model. In this paper we will stress those structures that are specific to Web applications and that are not usually found in other kinds of applications.

In a Web application, users interact with nodes and links – more precisely, with their interfaces. Nodes are information containers; links implement meaningful relationships among nodes. In the OOHDM approach, nodes are defined as views on conceptual objects, while links represent views on relationships. For each particular user profile, we build a navigational model as an observation of the conceptual model, as explained in section 3. The navigational model comprises a navigation schema that shows the nodes and links structure, and the navigational context schema that contains different sets of objects intended to be navigated in some order – called contexts in OOHDM, and the access structures (indexes) that provide navigation paths to reach these objects.

Finally, for each navigational model we build an interface model using the Abstract Data Views – ADV design approach [Rossi *et al*. 1995]. Notice that for a particular user profile we may build different interfaces, as well as for different interface devices, e.g., a Web browser, a palm computer or a cellular phone, and in this process reusing the specification may be paramount. ADVs also represent Observers with respect to navigation objects; for the sake of conciseness we will not further explain the interface model in this paper.

3. Viewing Nodes as Observers on conceptual objects

3.1. Our approach

Creating specialized applications as views of shared model is not new [Fowler 1993]. In the OOHDM approach, we conceive a Web application as an opportunistic observation on a particular domain. In the field of Conference Review Web Applications, for example, we may have different user profiles such as “PC Chair”, “PC member”, “Reviewer”; each of which will have a subjective view on the domain. For example, the PC Chair will be able to read all reviews, whereas each PC member will access only those reviews that are authorized by the PC Chair. Different behaviors will be allocated to each one of these views. Notice that all users will navigate through information contained in the same conceptual objects, but organized in different ways according to each one’s tasks.

We realize these views by specifying nodes and links according to the specific user profile and task. Following well-known object-oriented design techniques, we aim at decoupling the view from the viewed object. We use a variant of the Observer design pattern to specify corresponding navigational classes, as shown in figure 1.

Each node class in the navigational model is built using a particular conceptual class as its subject (in the terminology of [Gamma *et al*. 1995]). However, in our specification language we allow views (nodes) to reflect attributes “pasted” from different conceptual classes. For example, for each paper we may have the names of (and links

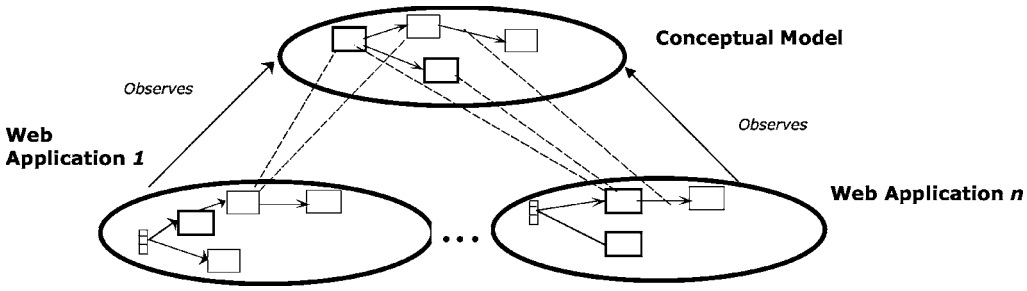

Figure 1. Observers in the relationship between conceptual and navigational model.

```
NODE Paper [FROM Paper: P]
Topics: String   [SELECT Name] [FROM Topic: T WHERE P belongsTo T]

....  (other attributes "preserved" from the conceptual class Paper}
Papers: Anchor  [Related]
```

Figure 2. A paper node with attributes from different objects.

```
LINK Related
SOURCE: Paper: p
TARGET: Paper: p1
WHERE S.p is reviewed by r and S.p1 is reviewed by r
END
```

Figure 3. Defining links opportunistically.

to) reviewers that evaluated this paper. This strategy allows us to reduce navigation overhead by putting attributes of different conceptual objects together in the same node, as shown in figure 2. Node classes (as in the example in figure 2) may be implemented in an object-oriented setting as type-objects [Johnson and Woolf 1988], thus eliminating the need to create multiple classes for each type of observers.

In figure 2, the attribute "Papers" indicates an anchor for a link from a paper to other papers. Each link class provides the basis for navigating among nodes in an application. Again, links may be defined opportunistically, for instance allowing more than one "conceptual jump" with only one navigational operation. For example, on can provide links going from one paper to papers evaluated by the same reviewer, as shown in figure 3 (using a simplified notation). Notice that in the conceptual model there may not be a relationship with this semantic. The From clause indicates the variable representing the observed object and it is used to express the definition of variables in a declarative way. The notation "S.p" indicates the Subject of a Node (in terms of the relationship Observer–Subject in the Observer Pattern).

A set of node and link classes defines a navigation class schema, and in some domains we may have many different such schemas for the same conceptual model, as

shown in figure 1. Notice that this situation poses interesting requirements with respect to reuse and evolution. For example, what happens when many views can share some definition of nodes and link attributes? We will discuss these aspects in sections 5 and 6.

Nodes also contain anchors for links whose source is the node itself. Anchors are intermediate objects mediating among nodes and links, and are usually implemented in the interface as active objects, i.e., those that react to certain interface events. The behavior of nodes thus allows them to route some messages to their contained anchors, which in turn activate links. Nodes may also contain additional behaviors that will be triggered from other active objects, such as submitting a review for a paper. In this case, the observer plays an interesting role with respect to the observed objects by forwarding a message to the conceptual object.

3.2. Comparison with similar approaches

A similar use of Observers for building application views can be found in the model-view-controller architecture. The Model-View-Controller (MVC) metaphor for building interactive applications has been widely used for building conventional software [Krasner and Pope 1988] and has been recently adapted to the Web domain [Kassem 2000]. We next describe the most important features of the MVC components when used in Web Applications and compare it with the OOHDM approach.

The MVC provides a clean separation among application, interface and interaction issues in the design, located respectively in the model, view and controller components.

While the MVC provides a set of structuring principia for building modular interactive (in particular Web) applications, it does not completely fulfill the requirements of Web applications for providing rich hypermedia structures. This happens because it is based on a purely transactional view of software and it does not take into particular consideration the navigation aspects that we have argued should be appropriately supported. Moreover, the MVC has been applied so far as an implementation architecture more than as a design structuring mechanism.

OOHDM meanwhile uses two layers of observers (the navigational and the interface models). An MVC-based implementation of an OOHDM model could be used for example when mapping OOHDM designs to the J2EE environment [Kassem 2000]. In this case, conceptual objects are mapped into the Model component, nodes and interface behaviors are implemented in the Controller module, and interface appearances are dealt with in the View module. For a more detailed comparison, the reader is referred to [Jacyntho *et al*. 2002].

4. The meaning of aggregation in Navigation models

Object-oriented modeling notation contains many different semantics for the concept of aggregation. UML, for instance, differentiates aggregation from composition in that a part of a composite object may not be part of another, while aggregated objects may share parts. In Web applications, it is usual, meanwhile, that nodes are formed out of

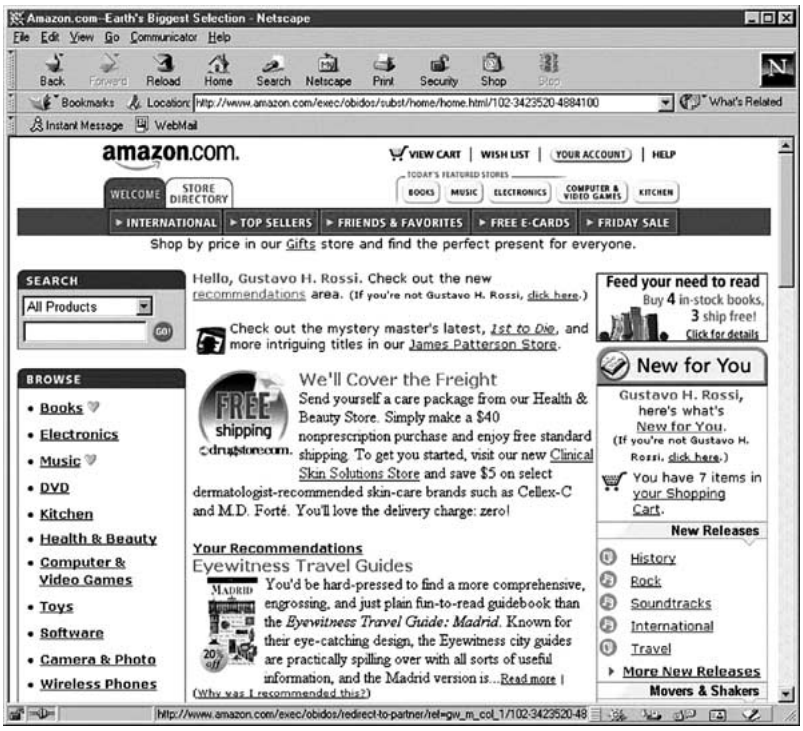

Figure 4. An example of aggregation: nesting of nodes in a home page.

nested parts (that themselves may appear in other nodes), as shown in figure 4. The Amazon home page, for example, is built out of different modules (New for you, New Releases, Search modules, Recommendations, etc.).

Notice that the semantics of this kind of nesting is similar to UML aggregation. In OOHDM, we have incorporated the idea of aggregated nodes to model those nodes that will act as "glue" for different parts. Figure 5 shows the specification that models the home page in figure 4.

Each part of an aggregated node may be itself a view on a conceptual object, thus allowing us to combine the concept of Observers with aggregated nodes. An interesting example of composition of user-selected parts can be found in `www.my.yahoo.com` (and in others `my.xx.com` sites), as we discuss in section 6.

5. Using Decorators in the navigational model

There are situations in which many users profiles may share all or part of the features of a particular design construct. For example, in the Conference Paper Reviewing application we have two specific kinds of users: PC Chair and PC Member; each will access different nodes (views) of the same object, the paper. In figure 6, we show the definition of Node

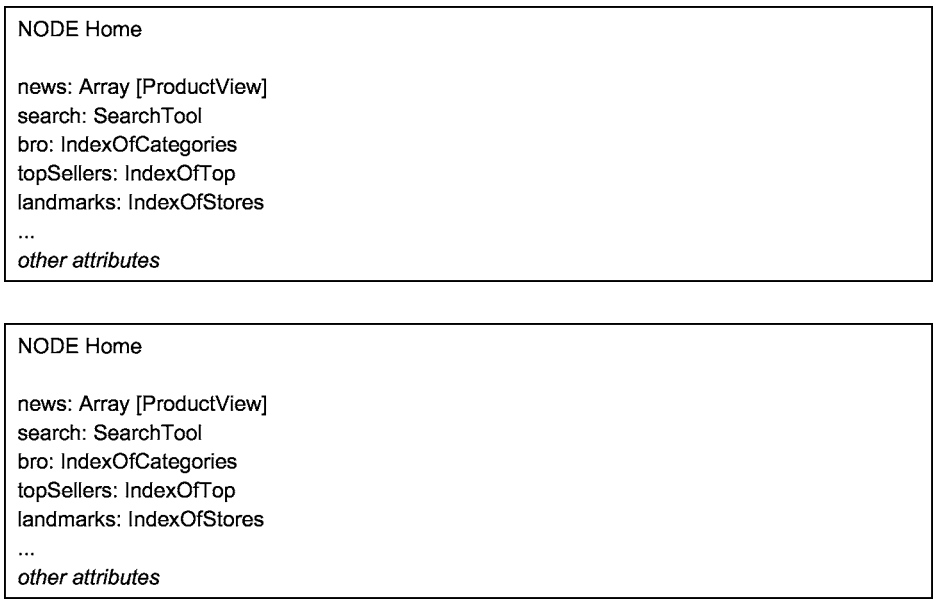

Figure 5. Using aggregation in node definition.

NODE PCMember.Paper **From** Paper Name:String Authors: String Abstract: Text MyEvaluation: text	**NODE** PCChair.Paper **From** Paper Name:String Authors: String Abstract: Text reviews: Anchor [ReviewedBy] additionalComments: Text

Figure 6. Node Paper for PC Chair and PC Member.

NODE Paper **From** Paper Name:String Authors: String Abstract: Text	**NODE** PCChair.Paper **EXTENDS** Paper reviews: Anchor [ReviewedBy] additionalComments: Text

Figure 7. Decorators in different user profiles.

paper for both profiles. Notice that part of the attributes (we do not include behaviors) is shared.

The best solution to this problem from an object-oriented point of view is to use the idea of Decorators in the definition of both nodes. We define a base Paper class and two decorators (we call them Extensions) as shown in figure 7. This solution is clearly better than using inheritance to specialize the paper class, as it does not preclude the definition of other sub-classes of paper in a broader context, such as Conference Paper and Journal Paper. This specialization would be impossible without polluting the hierarchy with the two classes corresponding to users' profiles.

Even in simple Web applications, the same node may appear in different contexts (defined in OOHDM as navigational contexts). For example, a paper may be browsed while navigating through the papers reviewed by a particular PC Member or while looking at papers on a particular subject. In both cases, the node (paper) will exhibit similar information: name, author, abstract and provide some anchors to navigate to the full version.

On the other hand, in each context the node will provide context-specific information and anchors; for example, in the PC members context we will find an anchor to navigate to information about that PC Member, and anchors for the next and previous paper by the same member. In the Subject Context, we may have anchors to the area's chair (if any), and again for next and previous papers in the context; some information about the area may also be included, such as the topics in the area. Again, we face the need for expressing some kind of variability that may appear in different occurrences of the same instance. In OOHDM, we use InContext classes that act as decorators for nodes, as shown in figure 8. When a node (for example, PC.Chair.Paper) is accessed in a particular context, such as the set of "Accepted" papers in figure 8, it exhibits additional information such as the schedule for presentation and anchors for accessing the next and previous papers in the set. The variable *T* in the second InContext definition stands for each member of a family of navigational contexts, one for each possible topic.

Notice that the anchors for traversing the set sequentially can be defined in an Abstract class for all InContext classes.

6. Design structures for personalized applications

There is a current trend in the Web industry about building Web applications that fit the needs of each individual to varying degrees. This kind of personalized software may involve simple features such as those shown before, i.e., different nodes for different user profiles, or may try to make finer grained personalization, e.g., a different node for each individual.

We have characterized some design problems involved in the development of customized Web applications [Rossi *et al*. 2001a,b], and identified which design structures are necessary to model different personalization strategies.

In the context of our design model, it is necessary to include the concept of User in both the navigational and conceptual models. The conceptual model should support objects standing for each individual user and storing information about him that may be useful during the personalization process. Meanwhile, the navigational model will connect the appropriate user object with its counterpart in the specific navigational ob-

InContext Accepted **Extends** PCChair.Paper	**InContext** Topic:T **Extends** PCChair.Paper
next: Anchor	next: Anchor
previous: Anchor	previous: Anchor
schedule: Date	relationToTopic: Text

Figure 8. InContext classes acting as decorators in different contexts.

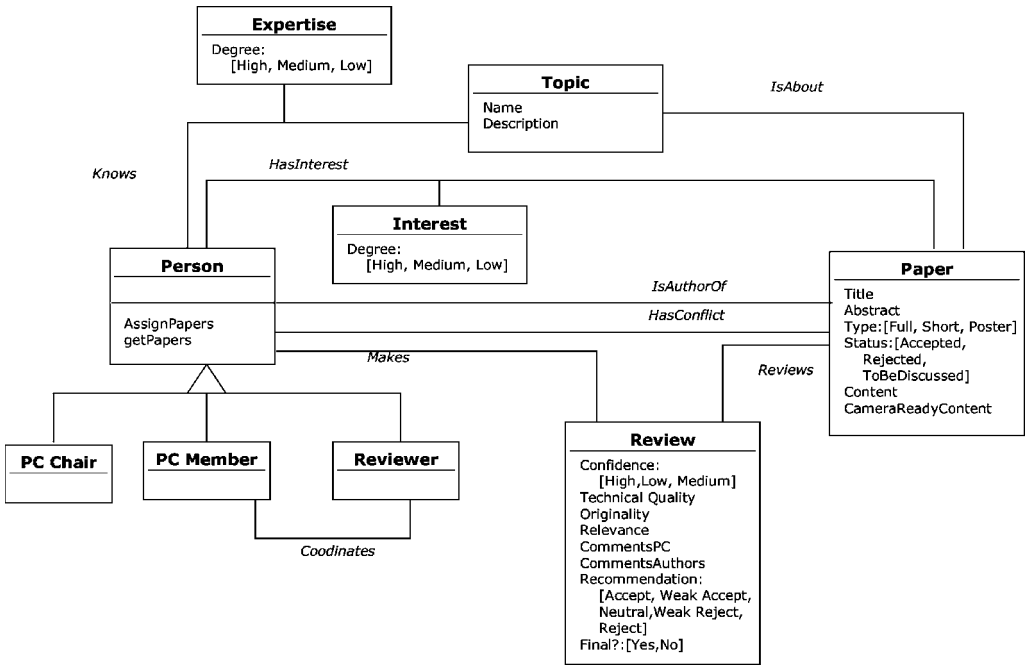

Figure 9. Modeling users in the conceptual model.

```
LINK PapersToEvaluate
SOURCE: Reviewer:r
TARGET: Paper: p
WHERE  p belongs to S.r getPapers
END
```

Figure 10. Specifying a personalized list of papers to evaluate.

ject. In figures 9 and 10, we show the specification of part of the conceptual model for the conference reviewing system and the definition of the link that connects the reviewer node to the papers he will evaluate. Paper assignment may be done manually by the PC Chair or using some algorithm that matches the person's interests with existing papers.

Notice that in this case personalization behavior is not provided separately from the domain model, i.e., there might not be specific personalization algorithms.

In figure 10, we bind the reviewer node to its counterpart in the conceptual model using the *S.r* expression.

It is interesting to notice that although personalization has received a lot of attention in the Web community, the need to use well-known object-oriented design structures for building this kind of software is not yet recognized, even though the same problems have already been faced in different circumstances.

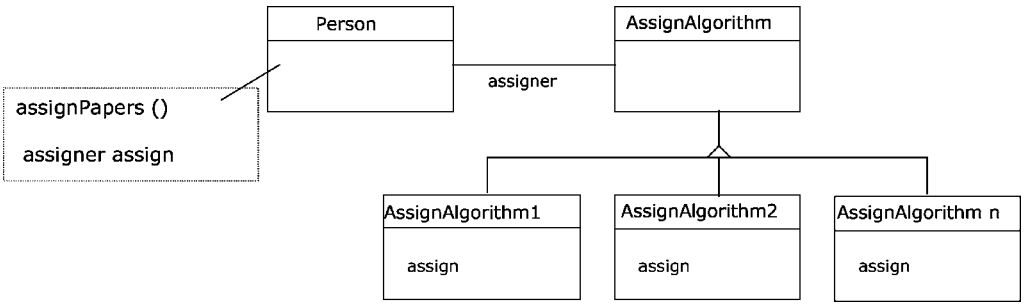

Figure 11. Using Strategy to deal with different algorithms.

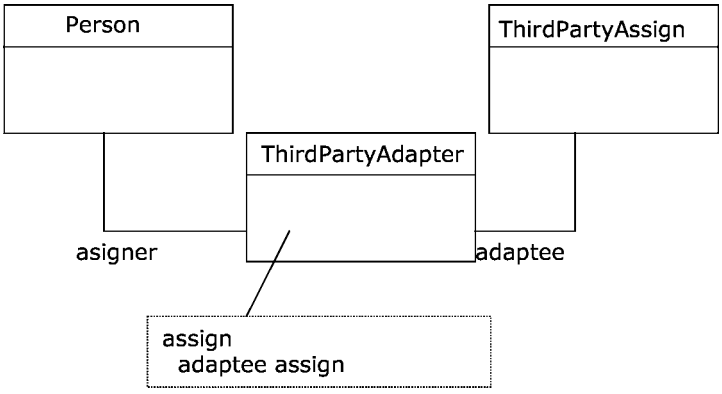

Figure 12. Using the Adapter design pattern for interfacing with third party software.

As an example, suppose that we want to use different algorithms to assign papers to reviewers, and that we want to change those algorithms for different conferences. In figure 11, we show how to use the Strategy design pattern [Gamma *et al*. 1995] to solve this problem elegantly. This solution permits decoupling the assignment algorithm from class Person thus allowing us to dynamically change the algorithm or even to assign different algorithms to different Persons.

A different problem arises when Web application designers use third-party personalization services (as those provided, for example, by `www.netperception.com`). Other design patterns such as the Adapter [Gamma *et al*. 1995] can be used to make different interfaces compatible (see figure 12).

As a last example, suppose that the same person may have different roles in the reviewing system (for example, PC member and Reviewer). Although the navigational appearance of the information object will not change, there is a clear need to separate the person from its role in the system, so that he doesn't have to enter information about him twice. He should also be able to switch roles in a reasonable way (from the point of view of consistency and design). In figure 13, we show how to apply the Roles design pattern [Baumer *et al*. 2000] to this situation.

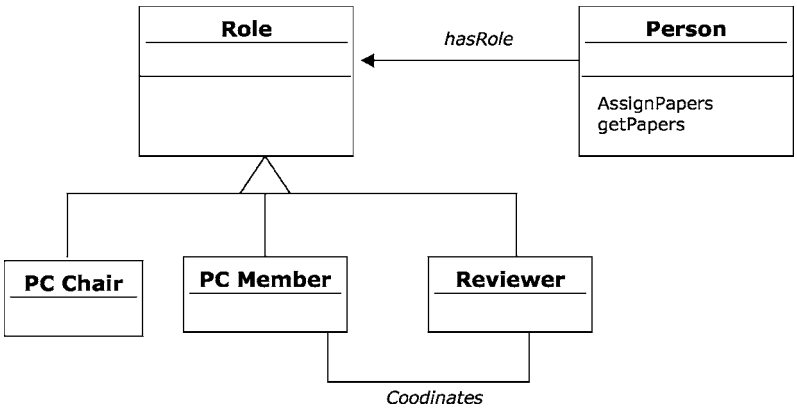

Figure 13. Decoupling roles from individuals.

7. One step further: From application models to frameworks

When many different Web applications must be constructed in the same domain, we can think about building an application framework in that domain. Object-oriented application frameworks are the state-of-the art solution for building high quality applications in a particular domain, by systematically reusing an abstract design for that domain [Fayad *et al*. 1999]. An object-oriented application framework is a reusable design built from a set of abstract and concrete classes, and a model of object collaborations. A framework provides a set of classes that, when instantiated, work together to accomplish certain tasks in the intended domain. An application framework is thus the skeleton of a set of applications that can be customized by an application developer. Application frameworks provide "templates" for supporting commonalties in the domain, and for accommodating individual variations. These "templates" usually have the form of abstract classes that must be sub-classified with concrete ones, or are filled with "hook" methods that must be implemented by the application's designer [Pree 1994].

It is evident that we can find many different examples in which frameworks may be useful in the Web domain, such as frameworks for electronic stores. In this case, the conceptual model may contain abstract classes for "Products" that will be classified according to each application; it will contain different classes and hook methods for different payment mechanisms, and so on. However, what makes Web applications different from other applications is that we have another axis of variability related to the navigational model. Variability in the navigational model can be achieved in different ways:

- By building completely new applications from the same conceptual model (e.g., defining a new user profile). In the conference review domain, we can define a new profile: General Chair, which may have different access restrictions compared to the PC chair.
- By defining a generic navigational schema (that allows adding new Node or Link classes and refining the definition of attributes). Notice that usually the addition of a new Node class is made each time we define a new class in the conceptual model.

- By defining generic contexts, i.e., defining abstract access structures and navigational contexts. In our example, one could allow several alternative ways to group papers into navigation contexts – by recommendation type, by review status, by referee preference, by author, etc. Each application (i.e., framework instantiation) will typically offer only a few of these.

A Web design framework is thus the combination of a generic conceptual schema and generic navigational and context schemas. It can be readily seen that with this combination we get a high degree of genericity in which we can change the underlying application model, add or refine profiles or tasks, and define different navigation topologies for specific applications. A particular application is then obtained first by exercising the hot spots in each schema, obtaining a concrete design, and then by mapping this design into an implementation environment.

We have been designing and implementing Web frameworks and studying design abstractions needed to support this type of generic designs. Most abstraction and reuse mechanisms simply mimic those existing in object-oriented framework technology, such as building abstract classes and template methods. However, we found a challenging concept when generalizing the idea of navigational contexts. While navigational contexts are sets whose objects fulfill some property, such as papers about a specific topic, or papers that have been accepted, generic navigation contexts add a level of variability in the specification. For example, we may say that our framework supports navigating through *Papers by property* and specify it leaving space for specific reviewing applications to implement the variability according to the specific need of the application. In other words, the actual paper property(ies) is defined when instantiating the framework. For example, in one application one may provide contexts for "Papers by rank", "Papers in alphabetical order" and "Papers by recommendation". Another example is "Paper by Relation to Reviewer", where the defining property is left open, but restricted to being based on a relation between Paper and Person. Examples of instantiations of this generic context would be "Paper Authored-by Person" and "Papers in Conflict-with Person" (see figure 9).

Although we have defined a notation for generic navigational contexts, it is clear that their very nature goes beyond of basic object-oriented concepts, and they have to be defined by combining several classes that interact to provide the expected functionality. For the sake of conciseness, we do not explain these issues further in the paper. A more detailed explanation of genericity in Web application frameworks can be found in [Schwabe *et al*. 2001a].

8. Concluding remarks

In this paper, we have discussed several object-oriented design constructs that are usually found in Web application models. We emphasized modeling instead of application constructs, since we believe that current implementation architectures for the Web are

still evolving and it is paramount to gain a complete understanding on the kind of abstractions we need to specify and design complex Web artifacts.

We have based our discussion on the OOHDM design framework; in particular, we have shown how to model and specify Web nodes and links as observers of conceptual objects and relationships. We have also shown how to use decorators both to build nodes extensions and to manage navigational contexts. We then addressed the construction of personalized applications and showed which design structures and patterns can be applied to model these kinds of situations. We finally discussed another level of genericity found while building application frameworks for a specific domain.

While object-oriented ideas are increasingly being used in the Web, they are generally relegated to the implementation phase of the development process. Modern architectures supporting components and objects still have to mature in order to be used effectively. Meanwhile, designers building complex applications face constant changes in both platforms and tools. We think that the investment in understanding design and modeling mechanisms is essential to cope with the changing nature of the Web environment.

References

Baumer, D., D. Riehle, W. Sibersky, and M. Wulf (2000), "Role Object," In *Pattern Languages of Program Design 4*, N. Harrison, B. Foote, and H. Rohnert, Eds., Addison-Wesley, Reading, MA, pp. 15–32.

Fayad, M., D. Schmidt, and R. Johnson, Eds. (1999), *Building Application Frameworks*, Wiley, New York.

Fowler, M. (1993), "Application Views: Another Technique in the Analysis and Design Armory," *Journal of Object Oriented Programming 7*, 1, 59–66.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995), *Design Patterns. Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA.

Jacyntho, M.D., D. Schwabe, and G. Rossi (2002), "A Software Architecture for Structuring Complex Web Applications," Technical Report, MCC-02/02, Department of Informatics, PUC-Rio, Rio de Janeiro, Brazil.

Johnson, R. and B. Foote (1988), "Designing Reusable Classes," *Journal of Object Oriented Programming 1*, 2, 22–35.

Johnson, R. and B. Woolf (1988), "Type-Object," In *Pattern Languages of Program Design 3*, R. Martin, D. Riehle, and F. Buschmann, Eds., Addison-Wesley, Reading, MA, 1998.

Kassem, N. and the Enterprise Team (2000), "Design Enterprise Applications with the Java 2 Platform, Enterprise Edition," available at `http://java.sun.com/blueprints/ guidelines/designing_enterprise_applications/apmTOC.html`.

Krasner, G. and S.A. Pope (1988), "Cookbook for Using Model-View-Controller Interface Paradigm in Smalltalk 80," *Journal of Object Oriented Programming 1*, 8, 26–49.

Nielsen, J. (1995), *Multimedia and Hypertext. The Internet and Beyond*, Academic Press, Boston.

Pree, W. (1994), *Design Patterns for Object-Oriented Software*, Addison-Wesley, Reading, MA.

Rational (1997), UML Document Set. Version 1.013 January 1997, Rational, available at `http://www.rational.com/uml/references/index.html`.

Rossi, G., D. Schwabe, C.J.P. de Lucena, and D.D. Cowan (1995), "An Object-Oriented Model for Designing the Human–Computer Interface of Hypermedia Applications," In *Proceedings of the International Workshop on Hypermedia Design* (*IWHD'95*), Springer Workshops in Computing Series, available at `ftp://ftp.inf.puc-rio.br/pub/docs/techreports/ 95_07_rossi.ps.gz`.

Rossi, G., D. Schwabe, and F. Lyardet (1999), "Web Application Models are More than Conceptual Models," In *Proceedings of the 1st International Workshop on Conceptual Modeling and the WWW*, Paris, France, November 1999, Lecture Notes in Computer Science, Vol. 1727, Springer, Berlin, pp. 239–253.

Rossi, G., F. Lyardet, and D. Schwabe (2000a), "Patterns for Designing Navigable Spaces," In *Pattern Languages of Program Design 4*, N. Harrison, B. Foote, and H. Rohnert, Eds., Addison-Wesley, Reading, MA.

Rossi, G., D. Schwabe, and F. Lyardet (2000b), "Patterns for E-commerce Applications," In *Proceedings of EuroPLoP 2000*, available at `http://www.coldewey.com/europlop2000`.

Rossi, G., D. Schwabe, and R. Guimarães (2001a), "Designing Personalized Web Applications," In *Proceedings of the 10th International Conference on the WWW*, ACM Press, New York, pp. 274–284.

Rossi, G., D. Schwabe, J. Danculovic, and L. Miaton (2001b), "Patterns for Personalized Web Applications," In *Proceedings of EuroPLoP 2001*, available at `http://hillside.net/patterns/EuroPLoP2001/papers.html`.

Schwabe, D. and G. Rossi (1998), "An Object-Oriented Approach to Web-Based Application Design," *Theory and Practice of Object Systems (TAPOS) 4*, 4, 207–225, Special Issue on the Internet.

Schwabe, D., G. Rossi, L. Esmeraldo, and F. Lyardet (2001a), "Web Design Frameworks: An Approach to Improve Reuse in Web Applications," In *Proceedings of the 2nd International Workshop on Web Engineering WWW9 Conference*, S. Murugesan and Y. Deshpande, Eds., Lecture Notes in Computer Science, Vol. 2016, Springer, pp. 335–352.

Schwabe, D., L. Esmeraldo, G. Rossi, and F. Lyardet (2001b), "Engineering Web Applications for Reuse," *IEEE Multimedia 8*, 1, 20–31.