Cohesive Design of Personalized Web Applications

Good software engineering practices, such as separating concerns and identifying patterns, simplify the critical design decisions in building personalized Web applications.

Daniel Schwabe and Robson Mattos Guimarães
Pontifícia Universidade Católica do Rio de Janeiro

Gustavo Rossi
La Plata University, Argentina

Although writing software that adapts itself to its users is not a new problem in software engineering,[1-2] the wide reach of the Web, combined with the myriad platforms that support browsing, have reshaped it. Today's challenge is building customizable Internet applications. Almost from the Web's beginning, browsers let users personalize presentation features. With the rise of e-commerce and increasing competition among sophisticated Web sites, other kinds of personalization have become even more important. For example, a personalizable Internet application might include multiple interfaces, each customized to a specific device. It might provide different navigation topologies to different users, recommend specific products according to a user's preferences, or implement multiple pricing policies. All these kinds of personalization require an application to model the user and user preferences, build profiles, find algorithms for best linking options, and so on.

Researchers have addressed each of these problems individually, but they have paid little attention to integrating all the personalization features into a single design. How can the designer comprehend the design abstractions occurring and the interplays between them, anticipate interaction patterns, and so on? Approaching personalization from the design perspective can give us an abstract understanding — independent of a specific application's characteristics — of the kinds of interactions personalized applications entail. It can also provide a framework for reusing designs and design experience. Meanwhile, a solid software engineering approach can produce complex Web applications that evolve seamlessly as their personalization requirements evolve.

This article describes our work in this area: using the Object-Oriented Hypermedia Design Method (OOHDM) for constructing customized Web applications.[3-4] Incorporating well-known object-oriented design structures and techniques, OOHDM produces flexible Web application models. Designers can add personalized behavior to these models with minimal code manipulation, and reasoning over design objects yields better insight on the personalization process. Although this article casts the discussion in terms of the OOHDM primitives, the ideas presented here can be easily applied to other design approaches, such as WebML.[5]

The Software Engineering Approach

Web-based software evolves continually. Applications that affect the core of a company's business, such as those for e-commerce, require tight schedules for new releases. In this context, shortening development cycles through design reuse becomes extremely important. Requirements related to personalization, however, are usually difficult to foresee: legacy Web applications — older Web applications written before personalization (and other issues such as dynamic generation) that became so important — are already a reality, possibly heralding a Web software crisis.

The only way to avoid the problem of dealing with older Web applications whose requirements and premises have become obsolete is to understand that designing complex Web applications requires rigorous software engineering techniques. Their design must be clearly documented, using appropriate notations.

This challenge is all the more difficult with Web applications, however, because they exhibit features that make them different from conventional software. For example, they use a navigational metaphor typical of hypermedia applications — most Web applications provide browsing facilities uncommon in conventional software. In addition, the exploding popularity of mobile, wireless, and other Internet-enabled devices forces us to face new interface design problems.

State-of-the-art software design notations and methods, such as UML (www.rational.com/uml) and associated process approaches, fail to consider these characteristics. Ultimately, we need new methods that address Web applications' particular design dimensions. These methods must be easy to integrate with existing methods so that they conform to existing standards. OOHDM is our answer to this challenge.

OOHDM Design Framework

The key concept behind OOHDM is that Web application models involve a conceptual model, a navigational model, and an interface model.[3] In OOHDM, we build these models using object-oriented primitives with a syntax close to that of UML.

Conceptual Model

The conceptual model represents domain objects, relationships, and the intended applications' functionality.

Suppose we're building a Web-based conference paper review system to help conference organizers assign reviewers and track reviewer recommendations. (Such a system could also work for selecting other types of items — software systems, services or equipment, suppliers, and so on.)

Our review system's conceptual model would contain classes such as "paper," "reviewer," and "review" along with appropriate behaviors — "assignPaper" and so on. In some Web applications, these behaviors might be quite complex — for example, we might need optimal algorithms for assigning papers according to reviewers' preferences. So specifying the behaviors with an object-oriented notation is valuable even though the implementation may involve different, possibly non–object-oriented, languages or databases.

Most personalization mechanisms involve objects and algorithms that are expressed as part of the conceptual model. In Figure 1 (next page), we show a simplified conceptual model of our conference paper review system. Objects belonging to subclasses of the `Person` class, such as "reviewer" and "committee member," are responsible for processing requests related to individual customizations.

Navigational Model

In the OOHDM approach, the user does not navigate through conceptual objects, but through navigation objects, or nodes. Using a language similar to object database view-definition approaches,[3] OOHDM defines nodes as *views* on conceptual objects.

Nodes contain perceivable attributes and anchors; they are complemented with links that are themselves specified as views on conceptual relationships. The navigational class schema shows the node and link classes that make up the navigational structure of our conference paper review application. Different user profiles (reviewer and program committee chair, for example) might have different interests and access rights, influencing the actual node structure. One easy way to use OOHDM to build customized Web applications is to reuse the conceptual model and build a different navigational model (view) for each user profile.

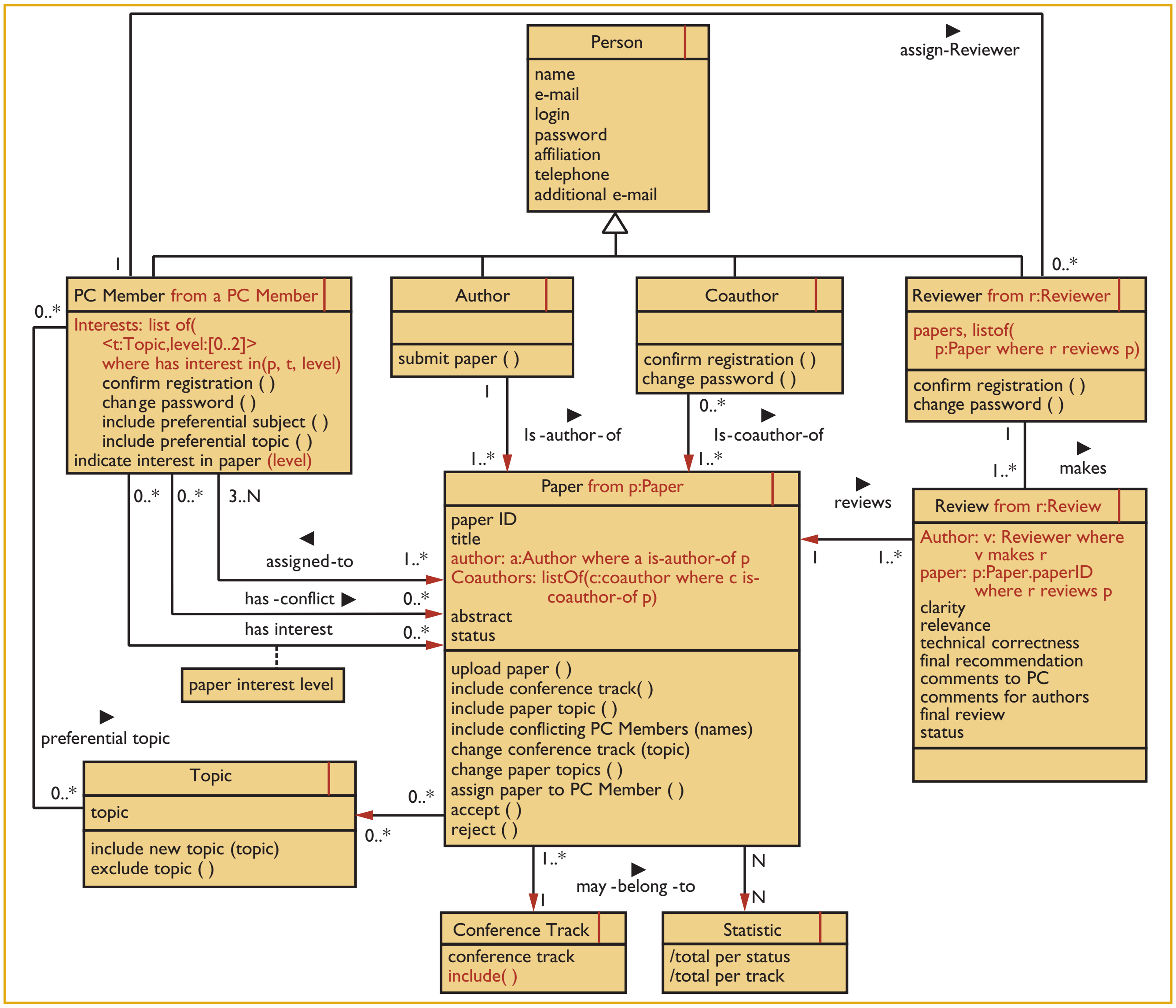

Figure 1. Conceptual model of the conference paper review system. Object classes represent papers, reviews, and people involved in the review process with their properties (attributes). This model also includes important relations between object classes, such as "PC Member is-assigned-to Paper." Text in red indicate the mappings over conceptual classes that define the navigational classes; this figure shows only a few to illustrate how attributes are mapped from the conceptual model (for example, the "author" of a "Paper"). The links in the navigational class schema are directional.

The red text in Figure 1 shows some of the navigation classes for the PC chair user profile.

In OOHDM, we specify the structure of the navigation space through a context schema, which shows the application's navigational contexts and access structures. A navigational context is a set of objects (papers written by an author, reviewers for a paper, and so on) that can be explored in a particular order — sequentially, for example.[3] Access structures act as indexes (lists of pointers, or links) to groups of related objects; each of its elements is specified by characterizing the target object (usually showing some of its attributes), and the anchor containing the link. Figure 2 shows an example context schema for the review system. For simplicity, we have omitted details, but this example should convey how to represent an application's entire navigation space compactly.

In Figure 2, rectangles indicate contexts (sets) of objects — for example, "paper by PC member." Dashed rectangles indicate access structures. Context cards fully detail the contexts, which include the actual condition defining the context members, access restrictions, and internal navigation structure. A context or an index preceded by a black ellipse (as is the main menu in Figure 2) indicates

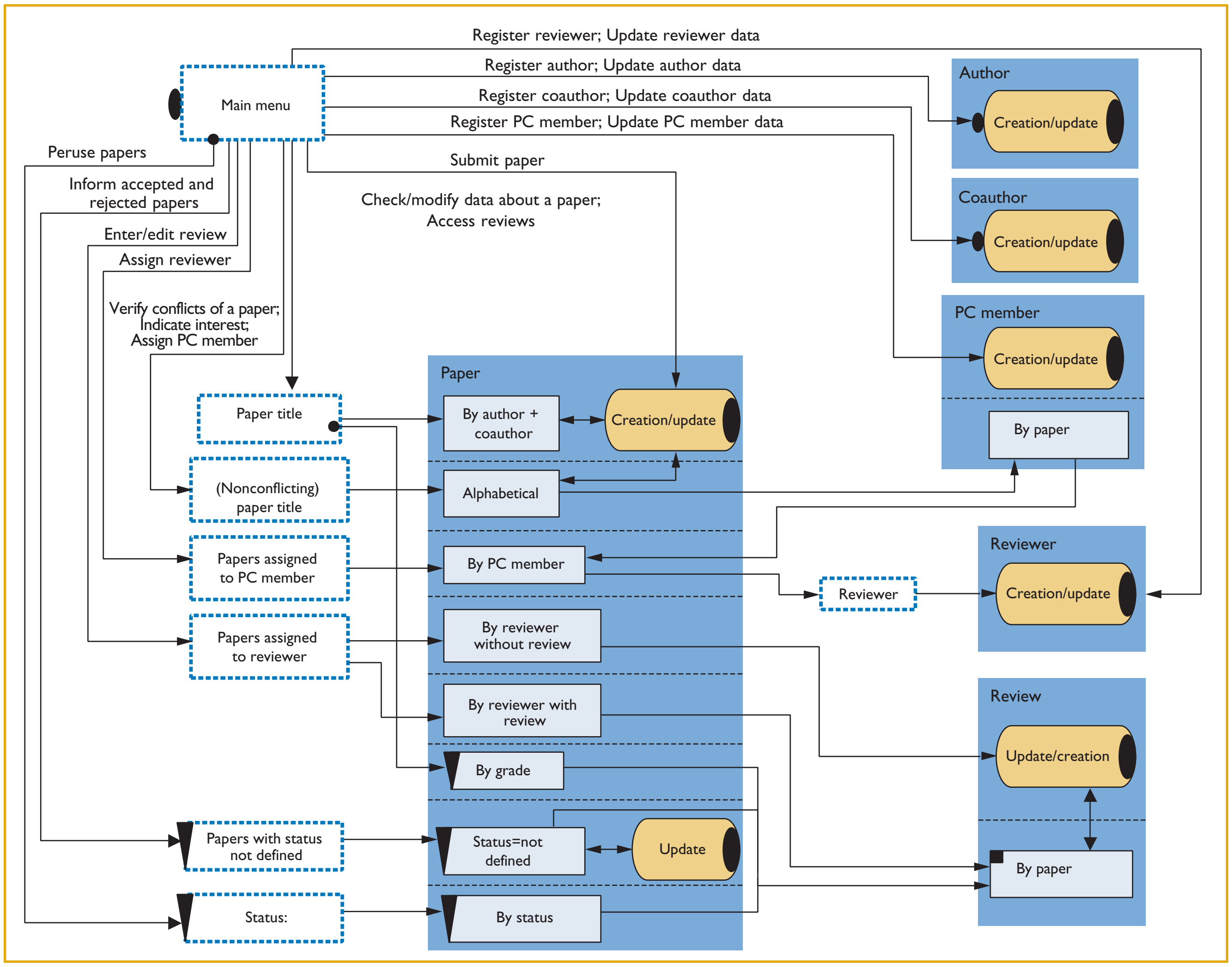
Register reviewer; Update reviewer data
Register author; Update author data
Register coauthor; Update coauthor data
Register PC member; Update PC member data
Main menu
Submit paper
Check/modify data about a paper; Access reviews
Peruse papers
Inform accepted and rejected papers
Enter/edit review
Assign reviewer
Verify conflicts of a paper; Indicate interest; Assign PC member
Author
Creation/update
Coauthor
Creation/update
PC member
Creation/update
By paper
Paper
Paper title
By author + coauthor
Creation/update
(Nonconflicting) paper title
Alphabetical
Papers assigned to PC member
By PC member
Reviewer
Reviewer
Creation/update
Papers assigned to reviewer
By reviewer without review
By reviewer with review
Review
Update/creation
By grade
Papers with status not defined
Status=not defined
Update
By paper
Status:
By status

Figure 2. Context diagram of a conference paper review system. Square solid boxes are sets of related objects, such as "Paper by PC member," gathering the papers assigned to that committee member. Lines indicate navigation possibilities, such as going from "paper by grade" to "reviews by paper." Dashed boxes are indexes.

an access restriction – typically, the user must log in to access its elements.

An arrow leaving a context indicates that from an element in that context, you can follow a link to another element in the destination context. For example, from the "paper by reviewer with review" context, you can navigate to the "reviews by paper" context, which is the set of reviews for that paper. Arrows with a black dot at the origin indicate landmarks, elements that can be accessed from anywhere in the application.

Because a node, such as a paper, can appear in different navigational contexts, we must define the features – the attributes and anchors – that apply for each context. For example, an anchor indicating the next paper in a context will activate different links according to the context in which it appears ("next paper by an author," "next paper about a topic," and so on). We specify those features using `InContext` classes that "decorate" the node when it is viewed in a context.[6]

Interface Model

Finally, the abstract interface model indicates the user-perceptible manifestation of navigation objects. Separating the interface from the navigation specification lets us cope with varying interface technologies modularly. For example, given a particular navigation model, we can specify different inter-faces – for a browser or for a variety of mobile devices such as phones and handheld organizers. We also use an object-oriented formalism – abstract data views,[7] which act as observers of nodes.[6]

For conciseness, we do not address interface personalization in this article, although our model handles it with the approach it uses for other

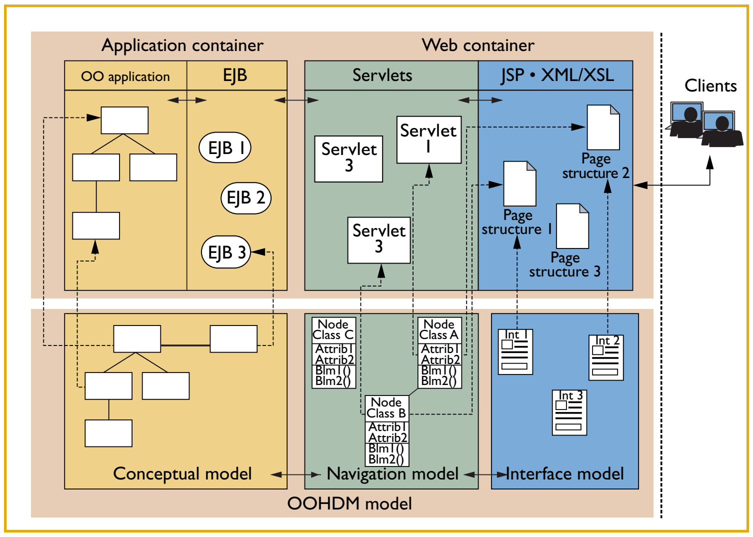

Figure 3. Mapping OOHDM schemas to a Web architecture. Each schema is mapped onto a different part of the runtime architecture: conceptual classes onto Java classes or EJBs (if concurrent access is needed); navigational classes onto servlets; and interface objects onto JSP pages.

kinds of personalization.

Hot Spots

In summary, the OOHDM framework provides hot spots (an idea from the application frameworks domain[8]) that let us specify customized structures and behaviors. It does this

- *in the conceptual model*, by explicitly representing users, roles, and groups, and by defining algorithms that implement different business rules for different users;
- *in the navigational model*, by defining completely different applications for each profile, by customizing node contents and structure, and by personalizing links and indexes; and
- *in the interface model*, by defining different layouts according to user preferences or selected devices.

Mapping OOHDM Designs

The process of building a running Web application from a set of OOHDM schemas is straightforward; the heuristics to perform this mapping depend on the implementation architecture (data model, page generation strategy, and so on). An automatic tool supports synchronization of design documents with implementation code.[9]

Now let's look at how we can map the schema we've already discussed — the conference paper review system — to an implementation using Java, Enterprise JavaBeans (EJB), Java Server Pages (JSP), and servlets together with XML/XSL. Figure 3 presents an overall architectural description. This architecture is in fact a simplification of a more sophisticated implementation framework we have been developing.[10]

We map classes in the conceptual model into Java classes or EJBs to provide concurrent access to shared resources. For each user profile, we define node classes as views over one or more conceptual class; we can readily implement these using JSPs that generate a portable XML (or HTML) document. Because nodes also act as façades for conceptual objects — for example, when providing some particular behavior — we can conveniently define servlets that trigger methods in conceptual objects and then activate corresponding server pages to generate a possible output. We derive XSL specifications from interface specifications statically when possible or dynamically when interface customization depends on profile information that might change during the application's execution. Figure 4 show portions of the XML and servlet specifications for a paper node in the review system.

We can cast most personalization patterns into object-oriented structures, in both the conceptual and navigational models. Doing this provides a road map to customized implementations that are closely related to the corresponding OOHDM specifications — and thus improves documentation and stakeholders' understanding. Expressing design decisions at a high level of abstraction, such as schema specifications, simplifies the overall development process; it also gives us forward and backward traceability information.

Personalization Patterns

During the past few years, we've been mining Web applications for patterns.[11] Patterns, which identify recurrent design problems along with their high-quality solutions, were devised by the architect Christopher Alexander[12] and later adapted to the software systems field.[6] In software, identifying patterns is useful for recording design experience and reasoning about the design process. We can describe a pattern by stating its intent, the problem it addresses, and the abstract solution to that problem. When a group of patterns covers all design problems in a particular domain, and we have a set of rules that lets us apply them in a certain order, we call this group a *pattern language*.

We've discovered a set of patterns related to personalized Web applications, which together show, in coarse grain, what we can personalize in such an application.[13] Let's turn now to some abstract patterns that illustrate our design strategies for describing personalized software. We'll present these patterns along with their corresponding OOHDM specifications.

```
<Paper title="" paperId="" status="">
   <author name="" email="" webpageUrl="">
     <organization name=""webpageUrl=""></organization>
   </author>
   <coauthorsList>
     <author name="" email="" webpageUrl="">
       <organization name="" webpageUrl=""></organization>
     </author>
     <author name="" email="" webpageUrl="">
       <organization name="" webpageUrl=""></organization>
     </author>
   </coauthorsList>
   <abstract></abstract>
</Paper>
```

Figure 4. XML specification of the paper node in the conference paper review system.

Role-based Personalization

Different kinds of users perform different tasks or have different roles. Each role might have different access rights and might be interested in different aspects of the same domain data. Roles access basically the same information objects, although they might view them at different abstraction levels.

For example, in the conference paper review system, the roles are PC chair, PC member, and reviewer; the information objects are papers and reviews. In a double-blind review, a PC chair can see a paper's authors, but the reviewers cannot. In addition, a reviewer can access other reviews of a paper assigned to him or her, but the authors cannot (at least until the final result is decided).

Solution. Define the application's nodes and links as different observations on the information model, decoupling objects from their consumers. Consider each possible view as a different application over the same domain.

OOHDM approach. We define nodes, links, and indexes by specifying a query on conceptual objects. This lets us, for example, define each attribute of a node either as identical to a conceptual object attribute, such as the title attribute of the paper node in the conceptual schema (Figure 1). We obtain other attributes — for instance, the authors' names — from the `person` conceptual class through a query. In the specification below, the variable *p* stands for the conceptual object acting as the "subject" of the node specification.

```
NODE Paper FROM Paper: p
Name:String
Authors: Set Select name From Person:A
               Where A IsAuthorOf p
Topics: Set [topic1, topic2,…]
```

Moreover, we can define completely different applications by just customizing node classes to each user role or profile, as shown in Figure 5 (next page). When defining the public customized view of papers, we basically use attributes from the paper class, adding the paper's schedule (the date and time of the session) by querying the session object. Meanwhile, in the PC chair view, we include the list of reviewers by querying corresponding objects in the conceptual model.

Many applications, such as the paper review system, define personalization according to the role the user is playing. When a user logs on, the application enables the roles assigned to that user.

In OOHDM, two primitives allow the design of link and content personalization for role-based personalization: views and contexts. Navigation in OOHDM designs always occurs within navigation contexts, which are sets of nodes.[3] The entire navigation space is organized into meaningful sets (contexts), according to the tasks the users must perform. Because a node is always accessed within a context, we can define additional attributes to be available to the node only when it is being accessed within a given context. We achieve this through `InContext` classes, which function as decorators[6] for nodes.

In the conference paper review system example, "paper by reviewer" and "paper by title" are useful contexts. The former gathers all papers assigned to a given reviewer; the latter gathers all papers, ordered alphabetically. When accessed within the paper by reviewer context, a paper could include the acceptance recommendation that the reviewer gave it; this attribute would not be available in other contexts. The linking topology is also customized — there could be *next* and *previous* links in the paper by reviewer context but not necessarily in the paper by title context.

A judicious design will define the navigation topology to identify the meaningful contexts for

```
NODE Public.Paper FROM Paper: p
        Name:String
        Authors: Set Set….
        Abstract: String
        Schedule
                Select date From Session:s
                Where p isScheduledIn s
```

```
NODE PC.ChairPaper FROM Paper: p
        Name:String
        Authors: Set Set….
        …..
        Reviewers:
                Set Select name From Reviewer:r
                Where p isReviewedBy r
```

Figure 5. Profile specifications. Different specifications show different information depending on the profile.

each role, letting the user see customized content according to the navigation path chosen (corresponding to different roles). The key need here is to identify individuals and their corresponding roles, so that the application enables the appropriate roles.

The following text shows a specification of the papers by reviewer context customized not only to the reviewer role but also to each individual reviewer.

```
Context: Papers BY Reviewer, user: Reviewer
Elements: p:Paper where p IsReviewed By user
InContext Class:PaperByReviewer
Navigation: sequential, order by authorName
```

We do this by applying a filter stating that the context's elements are only those papers that have been reviewed by the current user. The `InContext Class` declaration binds the context to the specific decorator, adding anchors for sequential navigation to other papers.

Link Personalization

Web applications deal with a great number of objects; how we reach them depends on many factors. Different users — individuals or roles — should have different linking topologies. For example, certain users should have more direct access to certain information objects. In the conference paper review system, we might want each reviewer to have direct access only to the papers he or she will evaluate and no others. An electronic store, such as Amazon, might want to give customers personal recommendations of products they might like. The products that are directly accessible for one user might be different than those accessed by a different user.

Solution. Personalize links by calculating the link's end point with user-related information. With personalized links, all users access the same information objects. Although anchors may look similar — see, for example, Amazon's link to Recommendations — each individual has a different, customized node topology.

Algorithms for calculating links' end points. Link personalization, the scenario we find in recommender applications,[14] is by far the most widespread kind of individual customization on the Web. In OOHDM, we specify links by indicating the source and the target; in the target we can write expressions involving conceptual objects, as shown here:

```
LINK Recommendations, user:Customer
SOURCE HomePage
TARGET CD: C  WHERE C belongsTo
user.reccomendations.
```

The variable "user" in this expression refers to the actual individual using the application; in this example it refers to a customer object. The application sends the message "recommendation" to the object standing for the user.

Not all recommendation strategies involve personalization, however. For example, sites such as Amazon provide other kinds of recommendations that depend on the product rather than the individual user. ("Users who bought this product also bought…" is a product-based recommendation.)

Our conference paper review application includes another interesting example of link personalization, shown below.

```
NODE Paper FROM Paper
Name:String
related: Anchor (relatedPapers)

Link relatedPaper, user: Reviewer
SOURCE  Paper
TARGET  Paper: P WHERE P belongsTo
user.assignedPapers
```

When a reviewer accesses a paper node, he or she can navigate to related papers that the reviewer is evaluating. This link is personalized.

Structure Personalization

Many applications must handle not only thousands of objects but also a great variety of subjects and services. Because the number of possible options can overwhelm users, we might want to circumscribe the navigation space to the aspects that interest the user. For example, most portals such as Yahoo or CNN.com deal with dozens of topics — from weather to news to stock values. Although we might want to organize subjects taxonomically to reduce complexity at each level, users might be reluctant to navigate to lower-level objects. We want to give the user some freedom of navigation without causing cognitive overhead.

Solution. Personalize the Web application's structure, or let the user do it. Consider the information space as a set of aggregated objects (or modules) and select only those objects that the user might want to consume. For each module, select only those parts that the user might wish to read. For example, one user might see sports and entertainment, another, financial news and politics. This solution is typically found in "my x" applications (such as My Yahoo, my.yahoo.com).

Modeling structure personalization

Structure personalization involves choosing a set of services — such as news, weather, business, and sports — and further selecting finer-level information from each module: which sports, which kinds of news, and so on. From an object-oriented view, this situation's navigational model is a recursive aggregation of objects. The application selects corresponding modules from the user profile as in the following text, which describes the actual screen one sees when accessing my.netscape.com.

```
NODE MyNetscape, user: User
       Advertisement: Text
       Welcome: Text  [user.name]
       Modules: Set [user.modules]

NODE Weather From Weather, user: User
       cityWeather: Set [user.weather]
       forecast: Forecast
[user.weatherForecast]
```

As in the link personalization example, the user variable binds the navigational specification with the conceptual object, which in turn manages the profile (but doesn't necessarily store it). The expressions in brackets indicate the values of attributes that come from the user object in response to messages such as "module," "weather," and so on. For simplicity, we didn't define a profile class.

Content Personalization

In some Web applications, we might want to provide individual users with slightly different content for a particular information item. For example, a virtual store might want to give customers special discounts according to their buying history. (We could solve this problem by splitting an object and using link personalization — for example, linking to "my price." But this solution introduces artificial objects — prices — that are really just attributes of an object.)

Solution. Define personalized contents in nodes by letting node attributes vary according to the user. From the standpoint of object orientation, this means that we partially couple the value of an attribute to the user consuming it. Many stores, such as Half.com, use this solution.

Content personalization in OOHDM. OOHDM handles content personalization similarly, by sending messages to users in the node specification, as shown here:

```
NODE Customer.CD FROM CD: c,
user:Customer
Name:String
Price: Real [subject.price -
user.CDdiscount]
Description: Image
     ....
     ....
Comments: Anchor [Comments]
```

In this case, user refers to an instance of customer, and the price attribute value is personalized by applying the corresponding discount. A more seamless design can be obtained by moving the discounting algorithm to the conceptual model, as a responsibility of the "user" object.

We can regard structure personalization as a particularization of content personalization. However, although the two specifications in OOHDM are similar, their intent is significantly different, so we discuss them separately.

We can achieve content personalization with role-based personalization, making certain attributes accessible only to certain roles and varying their values depending on the individual user. When a product's price doesn't depend on the particular user profile, we can delegate the discount calculation to the corresponding product object and further optimize the specification.

Behavior Personalization

Web applications combine hypertext navigation with other functions such as bidding, adding products to a shopping cart, and so on. Although these

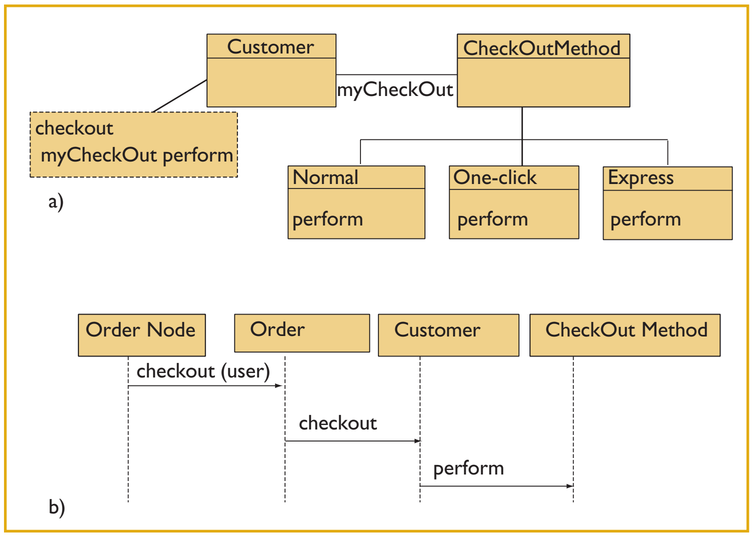

Figure 6. The OOHDM approach to behavior personalization: a) class diagram for personalized checkout and b) sequence diagram for the checkout behavior.

operations may appear as links in the browser's interface, in fact they trigger behaviors far more complex than just navigation. Suppose we want to provide individualized responses to a particular operation: for example, in an electronic store we might want to simplify the checkout process for some users, as the "one-click buy" function does at Amazon.com.

Solution. Personalize the application behavior by making the behavior dependent on the user triggering it. The result of this process might or might not imply that users perceive different pages upon activating the personalized operations. If navigation is involved, behavior personalization could be similar to link personalization. However, the main difference from link personalization is that what we are customizing is not the target of a link but the operation executed.

OOHDM approach. OOHDM offers various ways of obtaining personalized behaviors. Polymorphic behavior is implemented as a method in the corresponding node (for example, checkout in node order). Then, the message is sent to the corresponding conceptual class with user as a parameter, which in turn delegates the message to the object standing for the user (customer), as shown in the class and sequence diagrams, Figure 6. The customer in turn uses a strategy for different implementations of checkout.[6]

Using Patterns to Simplify Design

We specified all these pattern microarchitectures using "pure" object-oriented modeling concepts, with minor syntactic additions such as the user variable in navigational objects. Decoupling concerns in Web applications, as we've described here, is a key approach for obtaining modular, evolvable designs. These design specifications are useful even when using non–object-oriented implementation tools (Jim Conallen's book, *Building Web Applications with UML,* for example, includes an interesting discussion regarding ASP implementations[15]). Personalized applications pose new modeling challenges, such as representing the user, the personalization rules, and so forth.

We derive many personalized structures simply from well-designed conceptual and navigational models, and they might not require specific algorithms at all. For example, in the conference paper review system, each instance of PC committee member contains explicit references to the papers that member will evaluate, and in this way the corresponding Web page shows links to those papers.

Understanding the personalization patterns to apply simplifies the design enterprise, because we can reuse design experience when dealing with previously solved situations.

We have purposely avoided discussing aspects of personalization related to in-the-small algorithms (such as collaborative filtering, classification, and so on) or to the process of extracting valuable profile information from different data sources such as log files. In some specific personalization domains (such as recommender applications), we can find such finer-grained patterns.[14]

Conclusion

We strongly believe that the key aspect for obtaining evolvable personalized Web applications is to separate design concerns by focusing on architectural and design issues before building the actual application. Most of the design decisions we discussed here can be easily applied in other Web design methods.

To complete our pattern language, we are currently mining finer-grained patterns related to personalization. For example, the *advising* pattern focuses on helping a user find what he or she wants in a complex site.[16] In e-commerce applications, a typical use of *advising* consists of including recommendations on the user's home page. We can implement *advising* as a refinement of link personalization. *Push communication* provides another way of helping users identify new products in a site, by proactively informing them about new things they might be interested in.[16] This pattern can be implemented simply by sending emails to the user or as a refinement of structure or content personalization.

We are also investigating the problem of reengineering legacy Web applications to incorporate personalization features; we are closely following the OOHDM design framework while reverse and forward-engineering these applications. In this process, we have identified specific design problems that occur in personalizing Web applications and have found appropriate solutions for each of those problems. For example, by applying well-known design patterns like wrappers,[6] we can seamlessly extend existing applications by adding a personalization layer where we design the personalization rules.

We are currently working on applying similar strategies for separating business (personalization) rules from wrappers so that we can engineer those rules using lighter weight objects. The final purpose is to build a software infrastructure that lets us personalize legacy Web applications just by plugging in new code that implements the personalization logic.

References

1. P. Oreizy et al., "An Architecture-Based Approach to Self-Adaptive Software," *IEEE Intelligent Systems,* vol. 14, no. 3, May-June 1999, pp. 54-62.
2. P. Brusilovsky, "Methods and Techniques of Adaptive Hypermedia," *User Modeling and User Adaptive Interaction,* vol. 6, nos. 2-3, 1996, pp. 87-129.
3. D. Schwabe and G. Rossi, "An Object-Oriented Approach to Web-Based Application Design," *Theory and Practice of Object Systems (TAPOS),* vol. 4, no. 4, Oct. 1998, pp. 207-225.
4. G. Rossi, D. Schwabe, and R. Guimarães, "Designing Personalized Web Applications," *Proc. 10 Int'l Conf. WWW (WWW10),* ACM Press, New York, 2001, pp. 275-284.
5. S. Ceri, P. Fraternali, and S. Paraboschi: "Web Modeling Language (WebML): A Modeling Language for Designing Web Sites," *Proc. 9 Int'l World Wide Web Conference,* Elsevier, New York, 2000, pp. 137-157.
6. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, Reading, Mass., 1995.
7. D.D. Cowan and C.J.P. Lucena, "Abstract Data Views, An Interface Specification Concept to Enhance Design for Reuse," *IEEE Trans. Software Eng.,* vol. 21, no.3, Mar. 1995, pp. 229-243.
8. M. Fayad, D. Schmid, and R. Johnson, eds., *Building Object-Oriented Application Frameworks,* John Wiley & Sons, New York, 2000.
9. D. Schwabe, R. Pontes, and I. Moura, "OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW," *ACM SigWEB Newsletter,* vol. 8, no. 2, June 1999.
10. M.D. Jacyntho, "A Java Framework for Implementation of Hypermedia Applications in the WWW," MSc thesis, Dept. of Informatics, Pontifícia Universidade Católica do Rio de Janeiro, 2001 (in Portuguese).
11. G. Rossi, D. Schwabe, and F. Lyardet, "Improving Web Applications with Navigational Patterns," *Int'l J. Computer and Telecomm. Networking,* May 1999, pp. 589-600.
12. S. Alexander et al., *A Pattern Language,* Oxford Univ. Press, New York, 1977.
13. G. Rossi et al., "Patterns for Personalized Web Applications," *6th European Conf. Pattern Languages of Program – EuroPLoP 2001,* Hillside Group, Kloster Irsee, 2001, also available at www.hillside.net/patterns/EuroPLoP2001/papers/Rossi.zip.
14. J.B. Schafer, J. Konstan, and J. Riedl, "Recommender Systems in E-Commerce," *Proc. E-Commerce 99,* ACM Press, New York, 1999, pp. 158-166.
15. J. Conallen, *Building Web Applications with UML,* Addison-Wesley, Reading, Mass., 2000.
16. G. Rossi, F. Lyardet, and D. Schwabe, "Patterns for E-Commerce Applications," *Proc. 5 European Conf. Pattern Languages of Program – EuroPLoP 2000*, Hillside Group, Kloster Irsee, 2000, also available at www.coldewey.com/europlop2000/papers/rossi+lyardet+schwabe.zip.

Daniel Schwabe is an associate professor in the Department of Informatics at Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). He researches hypermedia and knowledge-based application design, from both academic and industrial perspectives. He holds a bachelor's degree in mathematics, an MSc in informatics from PUC-Rio, and a PhD in computer science from UCLA. He is a member of the ACM and the Brazilian Computing Society.

Robson Mattos Guimarães currently works at Oi PCS, a new mobile telephone company, where he is responsible for wireless Internet technologies at its research laboratory. His interests include pervasive computing, mobile applications, and the Semantic Web. He is finishing his master's degree in the Informatics Department at PUC-Rio, Brazil.

Gustavo Rossi is a full professor at La Plata University in Argentina and is the head of LIFIA, Laboratory for Education and Research in Advanced Informatics. His research interests include Web design patterns and frameworks. He is one of the OOHDM methodology authors and is working on the application of design patterns to the Web field. He holds a PhD in informatics from PUC-Rio.

Readers can contact the authors at {schwabe, robson}@inf.puc-rio.br or gustavo@sol.info.unlp.edu.ar.