Engineering Web Applications for Reuse

Daniel Schwabe and Luiselena Esmeraldo
Catholic University of Rio de Janeiro (PUC)

Gustavo Rossi and Fernando Lyardet
La Plata University

Web design frameworks combine generic conceptual, navigational, and context schemas. Such frameworks offer developers a conceptual approach to maximize design reuse, rather than code reuse, in Web applications. The authors apply the object-oriented hypermedia design extension, OOHDM-Frame, to determine key architectural components and design structures that lend themselves to reuse.

Building Web applications is a complex and time-consuming process. Such a task requires an understanding of the underlying domain—objects, behaviors, and application rules, for example. The task also requires that we carefully design the applications' navigational structure and user interface. User needs determine which navigation facilities to include, such as indexes, guided tours, and landmarks. The interface must guide the user through the sea of Web information by giving feedback on actions and by presenting the information clearly and meaningfully.

Another aspect to Web applications' complexity is that they often integrate distributed data or behavior repositories and usually support different user profiles. Applications typically evolve over time. As a result, they must be built modularly, taking into account the application's behavior, the need for easy navigation within an application, and the user interface. In addition, applications must be rapidly deployed with zero defects. To meet these concerns, software development should focus on reuse, combining the reuse of both design and components for efficient implementation of new applications.

We've designed several Web applications such as academic department Web sites, thematic portals, knowledge management portals, and online stores with our object-oriented hypermedia design method.[1] OOHDM considers Web applications as navigational views over an object model and provides basic constructs for navigation, such as contexts and indexes, and for user interface design. OOHDM lets us apply well-known object-oriented software engineering practices to build applications involving navigation. OOHDM supports several types of reuse directly; other types of reuse can be achieved through extensions of OOHDM such as OOHDM-Frame. We're seeking ways to maximize reuse in the development process, since we've observed a certain degree of commonality among navigation and interface solutions in similar application domains.

In this article, we introduce Web design frameworks—that is, a set of generic designs—as a way to reuse design in Web applications. We use OOHDM as a conceptual framework within which to discuss types of reuse. The most extensive type of reuse is achieved through complete application architectures, which are specified using OOHDM-Frame (an extension of OOHDM) in our examples. We also discuss the relation between Web design frameworks and Web application frameworks, arguing that although both must be integrated, the former can be specified independently of the latter.

The underlying concepts—reusing conceptual design, navigation design, abstract interface design, and implementation aspects of software development—also apply to other approaches, such as the WebML and HDM2000 design methodologies.[2,3]

Separation of concerns

Application domains grow ever more sophisticated, users need easier browsing to access more multimedia data, and new appliances require easy-to-use Web interfaces. To help designers manage the problem of the Web's complexity, we can separate three essential modeling concerns with the OOHDM: application behavior, navigational modeling, and interface design. Separation of concerns provides not only a solid ground for Web software evolution but also a basis for maximizing design and implementation reuse.

Application behavior

The core of every software application, Web based or not, is its domain or conceptual model. It must reflect which objects the application deals with, their relationships, and behaviors. In OOHDM, we specify the conceptual model with the Unified Modeling Language (UML) notation (http://www.rational.com/uml/resources/documentation/index.jsp).

1070-986X/01/$10.00 © 2001 IEEE

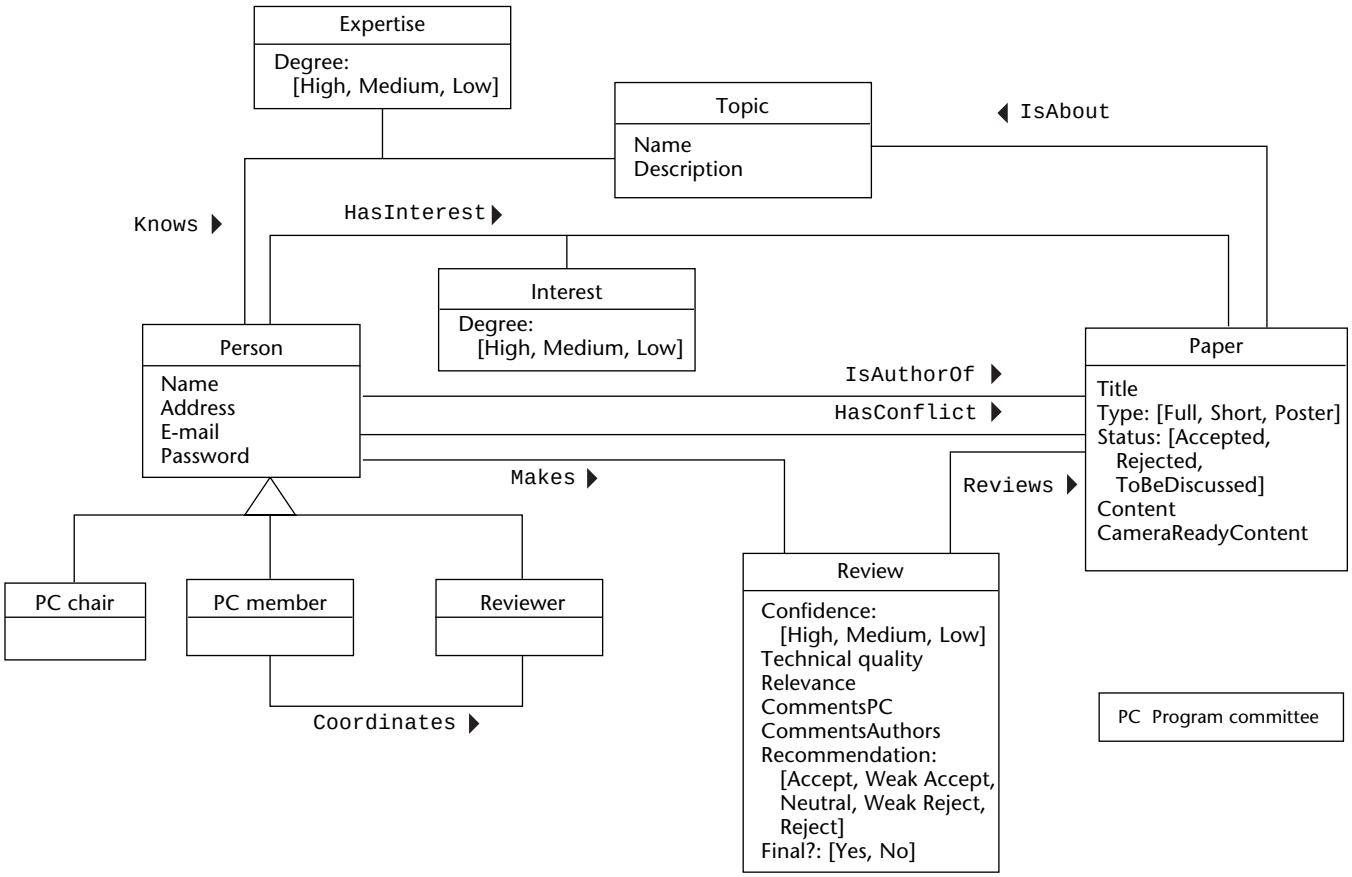


Figure 1. Conceptual model for the conference paper review system.

Web applications are distinguished by their navigational architecture—they're basically hypermedia applications. We must define which objects (nodes) users will perceive and how they will traverse the hyperspace (via links, indexes, and so on). We define nodes and links according to user profile and task.

In the rest of this article, we base our discussions and examples on conference paper review systems: Web-based applications that help conference program committees manage the process of receiving and evaluating conference papers. Figure 1 shows a possible conceptual model for this application. This model lets reviewers express their degree of interest in certain papers and indicate their degree of expertise concerning conference topics.

Navigational modeling

To support part of the conference paper review process, we defined a "view" over this schema in which nodes (Person, Paper, and Review) and links are observers—design patterns—of conceptual classes as Figure 2 (next page) shows.[4] Design patterns capture known good solutions to recurring design problems. Therefore, they represent design knowledge culled from the solutions given by experienced designers that have encountered these same problems before, possibly with slight variations. Applying a design patterns is a form of design reuse.

Each node class consists of attributes combined from different conceptual classes. Links indicate navigation paths and reflect conceptual relationships. For example, in Figure 2, the node class Person incorporates Expertise and Topic as attributes; similarly, Paper incorporates Topic as an attribute. Therefore, node class Person is an observer over the Person, Expertise, and Topic conceptual classes.

In OOHDM, we extended UML for defining node and link classes. We define node attributes with an object query language that lets us pick attributes from classes related to the observed class.[1] The navigational schema contains node classes and link relationships that indicate the basic navigation architecture. The navigational schema contains only links derived from "semantic" relationships (for example, the relationship `IsAuthorOf` between Person and Paper in Figure 2). Finer grained links that indicate, for example,

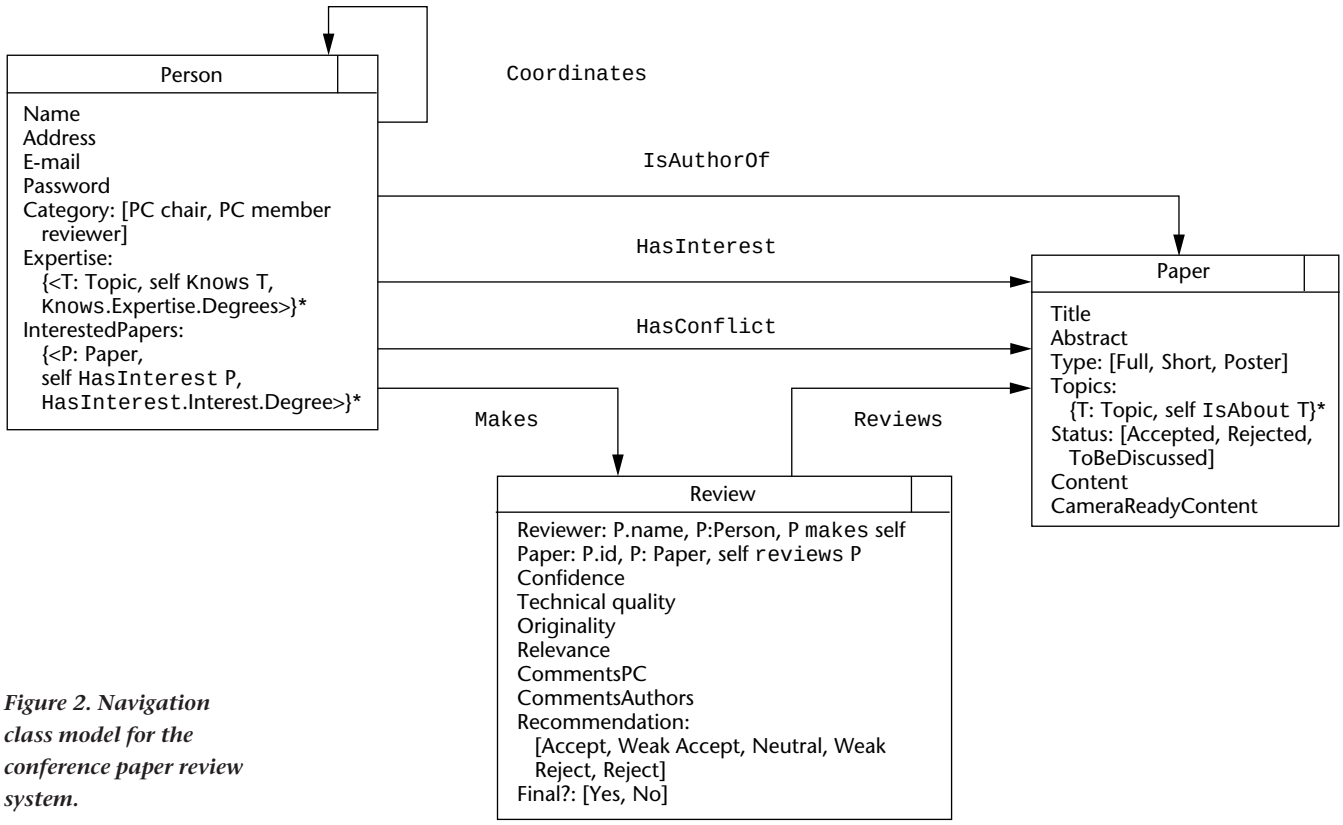


Figure 2. Navigation class model for the conference paper review system.

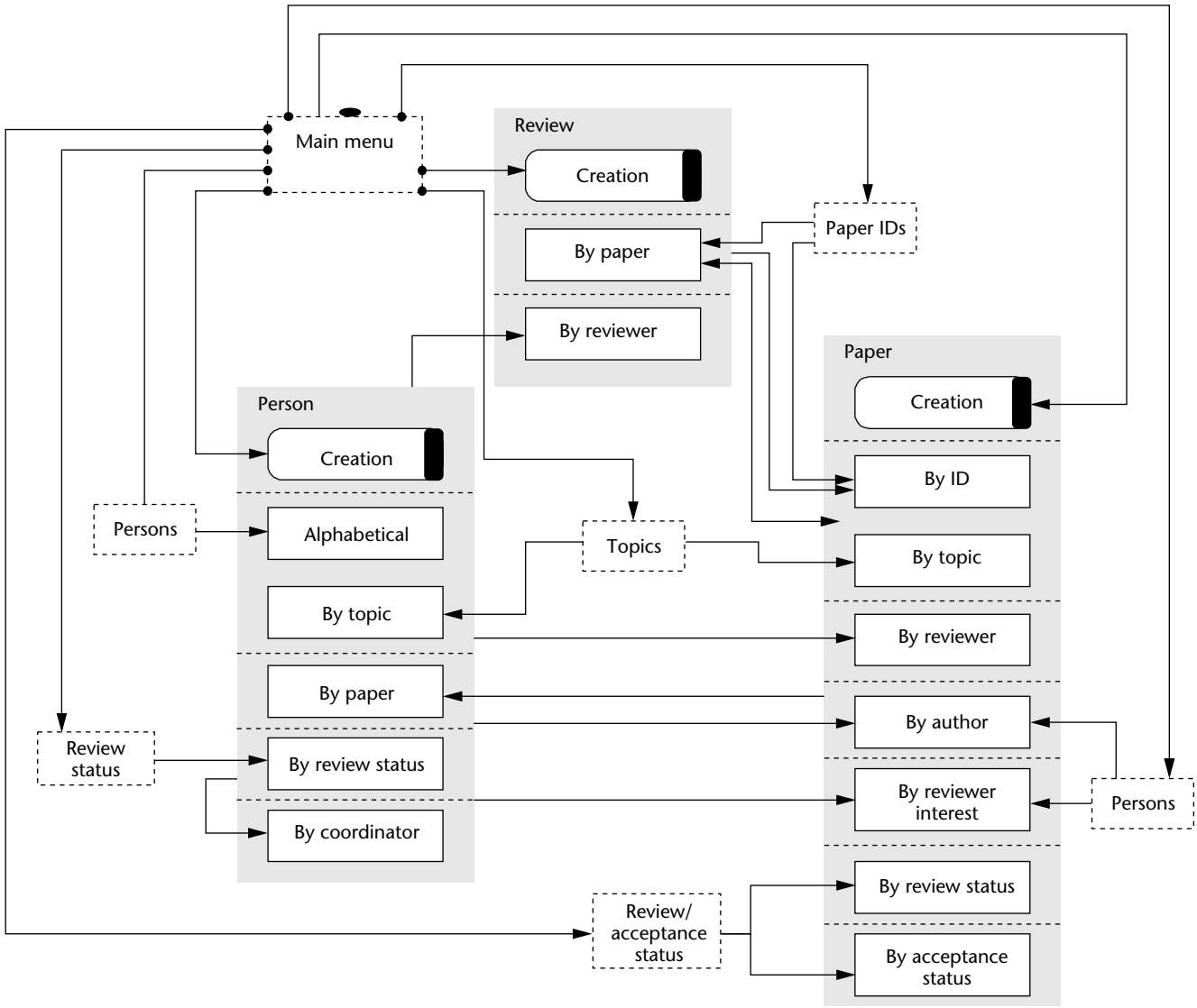
papers by the same author or papers reviewed by the same reviewer, are expressed as part of navigational contexts, explained next.

In OOHDM, we define navigational contexts as sets of nodes that can be accessed with indexes and traversed freely or in some order. Contexts can be defined by a query predicate, such as all instances of the node class Paper on the topic of hypermedia. We specify navigational contexts with a slight addition to UML by considering the contexts' nature as sets of nodes. The navigational context schema also shows indexes and notable entry points in the hyperspace (landmarks).

An index is a navigation object serving as an access structure containing references to other navigation objects. The same node can appear in more than one context—Paper by author and Paper by reviewer—with different attributes and outgoing links, for example, to reach the next/previous node in the context. These differences are expressed with `InContext` classes (decorators with the semantics defined by Gamma et al.[4] A decorator is a well-known design pattern that allows the adding of attributes to an existing class, under certain conditions.). Figure 3 shows a possible navigational context schema.

In Figure 3, rectangular boxes with solid borders denote contexts (sets of nodes); the label of each gray area indicates the class of the elements in the contexts within it. For example, Person alphabetical stands for an alphabetized list of persons (either reviewers or authors, or both). This information is actually contained in accompanying context specification cards—data structures that detail constraints and other aspects of instantiation.[1] Boxes with dashed borders indicate access structures (indexes), which also have corresponding access structure specification cards.

In the conference paper review system, we specify that papers can be navigated in several ways, such as by reviewer or by topic. This application allows submission of new papers, creation of reviews, and registration of new reviewers. Contexts whose elements (that is, objects that belong to it) might change during navigation are called dynamic contexts. Correspondingly, there are dynamic contexts (denoted by a black bar on the right), within which new instances of the appropriate navigation classes can be created. The application also permits the instantiation of relations, such as the assignment of a paper to a reviewer, which may occur during navigation of the Paper by Reviewer Interest context. This is also represented by the dynamic contexts.



Because operations in this domain depend on the reviewer's identity, the main menu has a protected access structure, indicated by the small black oval above it in Figure 3. This means that only authorized reviewers can access it and continue navigation. The access structure specification card lists the different user classes and access rights.

Interface design

Finally, the third modeling concern involves the user interface specification. For a given navigational model (a particular user profile), we might have to define different interface models according to the particular interface device to be used—for example, an Internet browser or a mobile phone. In OOHDM, we've used a formal model of interface objects that also considers the interface as a set of observers of navigation objects but space precludes further discussion.

A major advantage of using OOHDM for software design is that it lets us unambiguously identify these three design concerns independent of implementation. Even for a non-object-oriented implementation (such as combining relational stores with active server page or Java server page generation), however, it offers a clear understanding of the design aspects and their rationale, which simplifies system evolution and maintenance.

Design reuse

Classifying concerns as conceptual, navigational, and interface-related models gives us a rationale for design reuse. In the conceptual model, we could of course apply object-oriented reuse techniques such as defining abstract classes;

Figure 3. A navigational context schema for the conference paper review system. The dashed boxes indicate access structures (indexes).

we could also build application frameworks by systematically applying design patterns.[4,5] Our goal, however, is to push framework technology further into the Web domain through Web design frameworks.

Navigation patterns let us record, convey, and reuse design experience.[6] For instance, set-based navigation—which recommends organizing objects into meaningful sets (from the application point of view)—helps designers support tasks such as selecting items of interest in an electronic store, or picking a paper for reviewing. In object-oriented design, these patterns extend the basic Web navigation paradigm to solve recurrent problems. For example, the Landmark navigation pattern lets us model relevant nodes (points of reference) that must be reachable from every point in the application.[6] In Figure 3, the nodes are indicated by arrows with a black circle at the source.

Although the kind of reuse provided by patterns is valuable, complex Web applications need to maximize reusing larger design structures (the whole navigation topology or structure plus associated functionality). These design structures may arise at the application (conceptual level) or during navigational design. For example, the activities triggered when users order an item online is usually similar regardless of site. The way in which users navigate to locate and purchase items online is also usually similar.

Designers should find ways to express these commonalities by reusing generic designs such that only aspects unique to a particular site—for example, payment procedures or different ways of browsing—should be designed or programmed.

Similarly, although different conference paper review systems might vary in the individual procedures for selecting papers—for example, a two-tiered system versus a single-tiered system—they all have common functionality, such as reviewer registration, paper submission, assignment of paper to referees, and review submission (see http://www.cyberchair.org, http://dagwood.cs.byu.edu/PaperReview/, or http://witanweb.iit.nrc.ca/).

Object-oriented frameworks

An object-oriented application framework is a reusable design built from a set of abstract and concrete classes and a model of object collaborations in a particular domain. An application framework acts as the skeleton of a set of applications that can be specialized for a particular application. When many applications must be constructed in the same domain, frameworks provide templates for supporting their commonalities and accommodating variations. These templates usually take the form of abstract classes that must be subclassed with concrete classes. Alternatively, the classes are filled with "hook" methods implemented by the application's designer.[7] A key aspect of a framework is hot spots—places in the framework where the designer introduces an application's variations, or differences, in the same domain as the framework.

A hot spot identifies a place in the framework that the application designer can introduce an application's specificities; a hot spot delimits where reuse ends. For example, "paper assignment procedure" might be a hot spot in the paper review framework; actual applications will have different assignment procedures. Another example would be "payment method" in an electronic store—different stores accept different methods.

The basic philosophy behind frameworks—building reusable designs—can be applied to Web applications. However, we must carefully analyze and understand the variation points in Web application design models and the kind of abstractions involved in this kind of application. Unfortunately, current modeling languages (such as UML) don't support framework specifications well. Existing approaches emphasize a specific programming language. Instead, we use a simple notation, OOHDM-Frame, to develop generic Web designs that let us either generate an application framework (in a particular language) or specify a complete, running application in the intended domain.

Web design framework

Web design frameworks are environment- and language-independent whereas Web application frameworks are programmed in a specific language. A Web design framework contains a reusable application model in a particular domain that can later be instantiated into specific applications in that domain.

Our previous discussion on separation of concerns gives us a basis for characterizing a Web application's components, which we can use to design for reuse. Figure 4, which extends the OOHDM approach, shows an abstract diagram of a Web framework. It indicates what we call points of flexibility: design components where we design the framework's hot spots.

Figure 4 shows two main generic models—conceptual and navigational. A view mechanism describes the navigational model. (A view is meant here as a mapping between the two elements.)

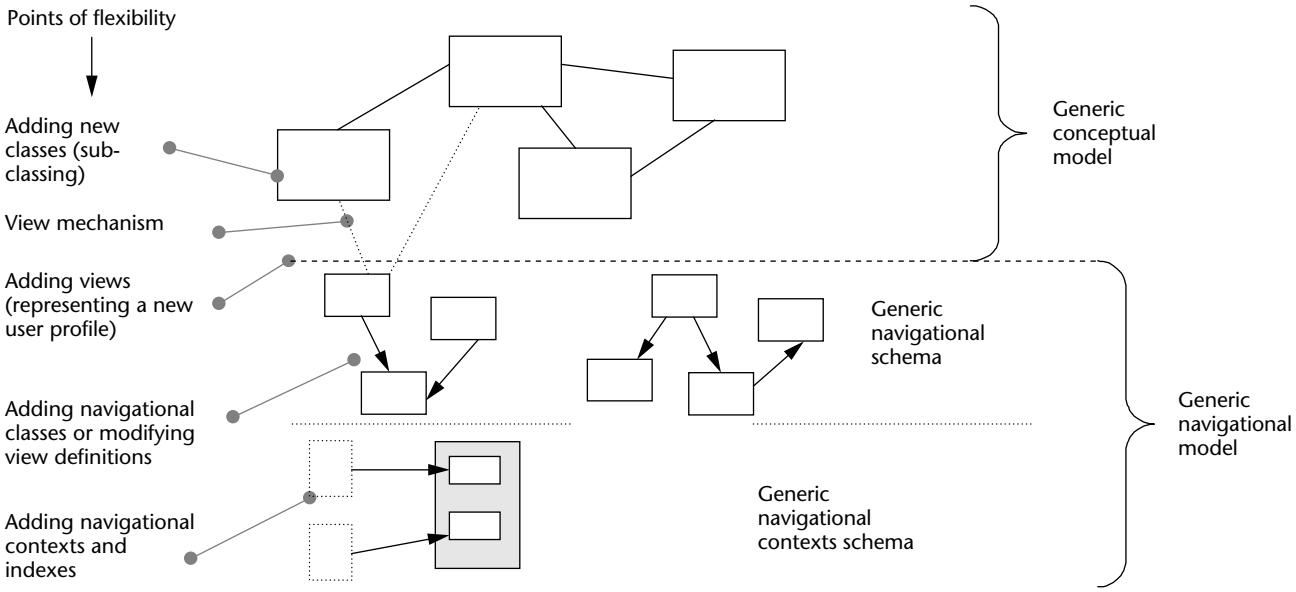


Figure 4. Components of a Web design framework.

The navigational model itself has two submodels—the generic navigation schema and the generic navigational context schema. Each model has points of flexibility that we can design for reuse using, for example, a subclassing mechanism.

Reuse in the conceptual model. As Figure 4 shows, we can specify hot spots in two places—the conceptual model and the navigational model. We can achieve genericity (genericness) in the conceptual model, through the particular hot spots that are allowed, with object-oriented techniques.[5] In particular, the hot spots of the conceptual model for the application domain can be defined according to Pree.[7] Briefly, this definition states that hot spots are framework components that will be replaced, either by class or subclass definition or by code rewriting, for a particular case.

For the conference paper review system, we can extend the conceptual model by defining subclasses. For instance, we could subclass paper as full paper, short paper, poster, and so forth. Because designing for reuse at the conceptual level employs well-known object-oriented techniques,[5] we focus instead on designing reusable navigational models.

Reuse in the navigational model. We can achieve variability (the degrees of freedom allowed in the framework) in the navigational model in different ways:

- By building completely new applications from the same conceptual model (such as defining a new user profile). For example, we could build an application permitting program committee members to delegate evaluation of papers to others, previously unknown (to the system), or we could include a view for the conference's general chair to evaluate the review process' status.
- By defining a generic navigational schema, which allows adding new node or link classes and refining the definition of attributes. We could redefine a review's Recommendation attribute, for instance, as numeric values rather than enumerated as specified in the schema in Figure 2 (Accepted, Rejected, ToBeDiscussed).
- By achieving a generic navigational context schema—that is, defining abstract access structures and navigational contexts. We could establish alternatives for grouping papers (into navigation contexts) by recommendation type, by review status, by referee preference, by author, and so on.

A Web design framework thus combines a generic conceptual schema and generic navigational and context schemas. This gives us a generic platform from which we can change the underlying application model, add or refine profiles or tasks, and define different navigation topologies for specific applications. We create a particular application by implementing the hot spots in each schema, developing a concrete design, and mapping this design into an implementation environment.

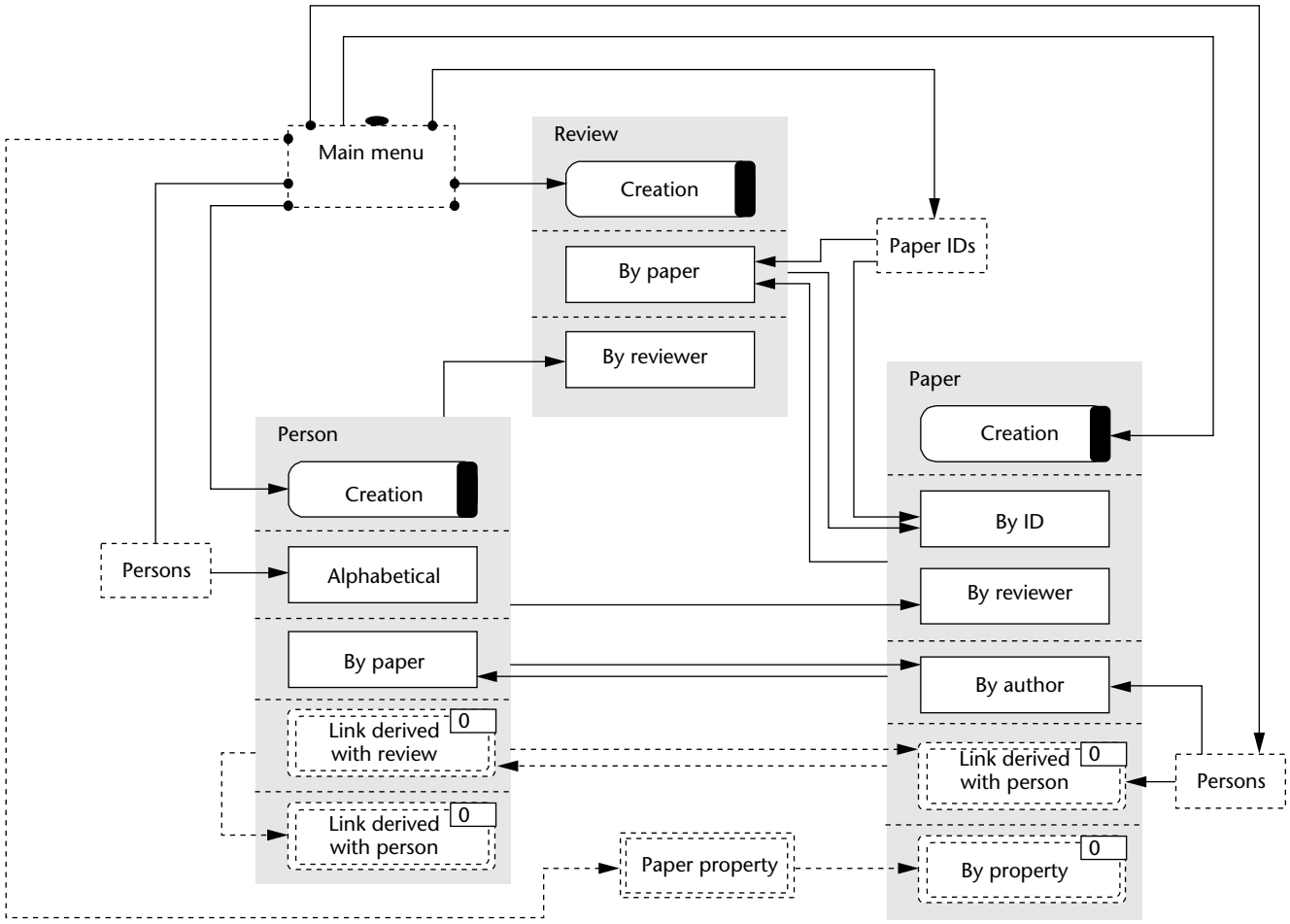


Figure 5. Generic context schema for the conference paper review system. Double-dashed border indicates generic elements (contexts or access structures). Here, "Paper property" is a generic index (access structure).

Specifying Web design frameworks

We've added simple primitives to OOHDM for specifying flexible design structures. This modified notation is called OOHDM-Frame. We focus here on using OOHDM to create generic Web design models; see elsewhere for notation details.[8]

As we'll show, we can achieve genericity in different ways. We might need to build completely different applications in the same domain (for example, different kinds of electronic stores). Or we might need to accommodate different user profiles (views) in the abstract model of one application and then customize it for each new user profile of the same store.

Genericity in the conceptual model

Creating a generic conceptual model requires abstracting the classes and behavior of different applications in the family. OOHDM-Frame uses UML to specify generic conceptual models. The only extension to UML is that dashed elements are optional.

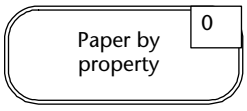
Specifying different user profiles

Accommodating variations in user profiles, and reusing what is common to all of them, lies midway between the conceptual and the navigational model. Each Web application is considered a view of the conceptual model. Given a particular conceptual model (generic in a domain or specific for an application), we can reuse it for different user profiles, as Figure 2 exemplified for Person in the navigation schema.

Specifying generic navigational schema

The navigational schema expresses which nodes the user will navigate, which attributes those objects will contain, and which links connect those objects. Therefore, node and link classes represent a Web application's basic navigational architecture. When designing a Web framework, we can specify abstract node classes (with fewer attributes) and link classes. These classes should be specialized for a particular application in the domain, much as they are in the conceptual model.

The viewing mechanism, which allows the definition of attributes inside a node, is more involved. For example, the Review class in the navigation schema in Figure 2 has imported the attributes Reviewer (the name of the person that created the Review) and Paper (the ID of the reviewed paper). Similarly, Reviewer has an expertise list of pairs `<topic, degree>`, derived from the conceptual relation class Expertise associated with the Knows relation between Person and Topic.



Paper by property generic context
Cardinality: 0 to *n*
Communicability: 0
Possible types: [static | session dynamic] + [index access]
Consistency/instantiation constraints: Definition predicate must be based only on paper attributes
User restrictions: Program Committee Chair or Person, where Person reviews Paper
Type: simple

Figure 6. A generic context specification card.

Expressing abstract navigational contexts

Different applications in the same domain can contain different navigational contexts. In some conferences, for example, reviewers might be allowed to "bid for" (request reviewing) papers of interest. In this situation, a context gathering all papers of interest for each reviewer would be helpful to the program committee chair to make the final assignments.

To express such generic contexts, the OOHDM primitives are extended within OODHM-Frame with the notion of a generic context, which stands for a possible set of contexts. We define these contexts parametrically by stating constraints over the properties that define possible instances. The double-dashed rectangles in Figure 5 denote such generic contexts. For example, the Paper by property generic context in Figure 5 stands for a set of contexts based on Paper properties. Some examples would be `Recommendation=Accepted`, `Status=ToBeDiscussed`, and so on. Generic contexts are further specified through specification cards (see Figure 6).

The same parameterization approach applies to link-derived contexts: those whose defining property is based on a 1 – *n* link. For example, the generic context Paper by link derived with person can be instantiated into any context based on any relation between Paper and Person, such as "has interest," "has conflict," or "reviews." Similarly, generic access structures allow specifying abstract indexes that can be later instantiated into concrete ones. The Paper property in Figure 5 is a generic index that can be instantiated into several indexes, according to the particular contexts instantiated for the Paper by property generic context.

When concrete contexts and access structures appear in the generic context schema, all applications derived from this framework must include them. In a sense, these contexts and structures are common to all applications built with the framework for the given domain. In our example, the context schema in Figure 5 specifies that all paper review systems must

- create new reviews,
- submit (create) papers,
- register (create) reviewers,
- navigate among reviewers in alphabetical order,
- navigate among all reviewers of a given paper,
- navigate among all papers by their IDs, and
- navigate among all papers assigned to a given reviewer.

In addition, Review can be navigated by paper ID or by reviewer.

One possible instantiation of the generic context schema in Figure 5 appears in Figure 3. The Paper by property generic context in Figure 5 has been instantiated in Figure 3 into two concrete contexts—Paper by acceptance status and Paper by review status. The Paper by link derived with person (Figure 5) has been instantiated in Figure 3 into a single context—Paper by reviewer interest.

Instantiating a Web design framework

We can implement the design in a Web environment in many ways. We briefly discuss two here: instantiating the design framework into an OOHDM model and implementing the resulting model in the Web, and implementing the design framework using a Web application framework.[5]

We implemented an object-oriented architecture that lets designers implement Web application frameworks for specific domains. This architecture contains classes that support the core OOHDM primitives—nodes, links, indexes, and contexts.

Web application model

We first instantiate the abstract model into a valid OOHDM model, then implement it with standard Web tools. The process for deriving a concrete OOHDM model from an OOHDM-Frame specification involves defining concrete classes (conceptual and navigational) and contexts from the generic diagrams.

To map an OOHDM model into a Web application, we used OOHDM-Web,[1,9] which is an implementation environment for OOHDM designs. A complete OOHDM design can be represented in OOHDM-Web with special-purpose data structures, which are basically nested lists of attribute-value pairs. These data structures contain class definitions, navigation context definitions, access structure definitions, and interface definitions. The description of database entries that store instance data is also included.

Context definitions comprise the query definition that selects the elements belonging to the context; the same is true for access structure definitions. Interface definitions are mixed HTML templates, one for each class in each context where it appears. The mixed HTML templates are pure HTML formatting instructions interspersed with function calls to a library of predefined functions, which are part of the OOHDM-Web environment. These functions allow retrieval of object attributes, or reference to other objects in specified contexts. We define reference functions to cause, when activated by users, the destination object to appear in the appropriate context, using the template defined for that context.

We've generalized this approach for dealing with OOHDM-Frame models by allowing substitution of generic definitions for concrete ones whenever appropriate. The resulting representation describes the framework's generic design. The instantiation process substitutes generic definitions in the framework by the definitions (using the OOHDM-Web representation) of their corresponding instantiated elements. For example, a generic class-derived context can be substituted by specific class-derived contexts in the instantiated framework. We achieve this by replacing, in the data structure that describes the framework, the generic context description by specific context descriptions.

Ultimately, when all hot spots have been plugged into the corresponding concrete application elements, the resulting data structure is a valid OOHDM-Web representation of the final instantiated application. Our current implementation doesn't automatically support all constraint verifications—the designer must do those manually while instantiating the framework.

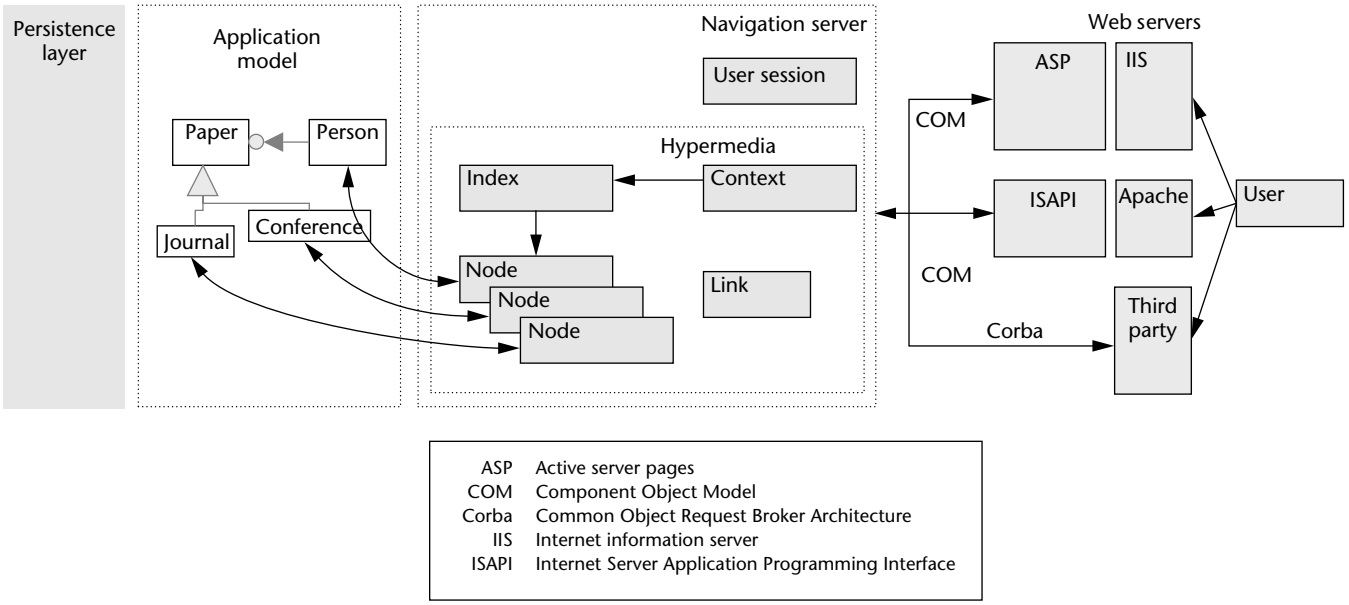
Web application framework

Web design frameworks for a particular application domain can also be mapped to an object-oriented framework. In a sense, the design frameworks thereby document the application frameworks' design.

We implemented an object-oriented architecture that lets designers implement Web application frameworks for specific domains. This architecture contains classes that support the core OOHDM primitives—nodes, links, indexes, and contexts. The abstract classes can be plugged into domain specific classes (such as Paper and Person) to extend the behavior of these application classes with navigation functionality.

The designer next implements the generic conceptual model with an object-oriented programming language such as Java. For each application, the designer either creates subclasses or instantiates the domain classes and connects them with the navigation classes and objects derived from the OOHDM-Frame generic navigational schema.

This architecture decouples the application and navigational model from the components that provide dynamic content generation on the Web and persistence. The components range from the Common Gateway Interface (CGI) and Internet Server Application Programming Interface (ISAPI) to active server pages (ASPs) and



Java server pages (JSPs). Decoupling lets a Web application framework be designed independent of particular commercial technologies so it can evolve seamlessly as Figure 7 shows.

In the architecture shown in Figure 7, the application model contains all application behavior (from the OOHDM-Frame generic conceptual model) expressed with an object-oriented language. Meanwhile, the navigation server contains classes that provide the ability to access nodes in different contexts, managing the navigation spaces and linking among nodes. The entire HTML rendering task is performed on the Web server side by either a custom third-party CGI/ISAPI module or through dynamic HTML. The other components manage persistence and dynamic page generation, typically page servers like ASP or JSP.

Problems of reuse

So far we've discussed some of the technical problems involved in reuse. Additionally, the investment needed to design and implement a Web (or an object-oriented) framework is important in coming up with the abstractions for later refinement and instantiation into an application. Moreover, using a framework is difficult, as we must understand those abstractions (generally poorly documented) to instantiate and extend the framework. OOHDM-Frame makes those design decisions explicit to the framework designer by providing a rich set of constructs to express them.

Reuse also involves complex economical and managerial aspects. The development life cycle must change with reuse-centric approaches. We must identify core abstractions and existing reusable components. New activities and roles must be defined, managed, and coordinated in the software team. Finally, it's usually difficult to estimate development costs (both for the framework developer and user), which adds risk to the development enterprise. These issues are addressed in detail elsewhere.[5]

Figure 7. Architecture for building Web application frameworks.

Web frameworks and architectures

The design frameworks we've discussed are largely independent of Web architecture in terms of implementation. For a complete solution, there's another level at which frameworks are also relevant, namely the implementation level. Since most Web-based applications exhibit common functionalities, it makes sense to speak of Web application frameworks. Such frameworks constitute predefined architectures that are reused in many actual implementations.

Web architectures define application framework components, their responsibilities, and the component relationships in a Web application environment. Typical components are the Web server, application server, the client, and a persistence server. From these, we can identify many architectural patterns.[10]

Finally, a Web application framework defines the architecture of a set of applications in a particular domain or conceptual model. This architecture is defined in terms of the relationships among domain components. While Web application frameworks deal with understanding the architecture of applications in the domain (and

part of this concern might not involve the Web at all), Web architectures address hardware and software implementation problems as well as communication components for building different Web applications (in different domains).

From a more conceptual point of view, the OOHDM approach also defines relationships among components (application objects, navigation nodes, and interfaces). The OOHDM design model induces an application architecture that may be itself implemented on top of different Web architectures. For example, it separates navigation information from content and from interface information.

From this point of view, since Web application frameworks are defined using the OOHDM approach, they can be implemented using different Web implementation architectures. (Fontoura describes an interesting approach for documenting traditional, non-Web-application, frameworks.[11])

Discussion and future work

Reuse of Web application design is already being employed in practice by the larger development organizations, albeit in a limited form. For example, it is beginning to appear in the form of "wizards" that are distributed together with many Web application servers or environments, such as ASP, Enterprise JavaBeans, Cold Fusion, StoryServer, Broadvision, WebSphere. Many are specific to e-commerce applications, but the principles behind them are the same: They provide an off-the-shelf solution that can be reused, with some adaptations.

We have observed that both within our own group, and within other design teams, designs rarely start from scratch, but rather from preexisting structures that are somewhat similar to what is needed in each particular case. The work presented here allows this practice to become a systematic, documented activity, which is independent of the implementation environment.

A distinguishing feature of a Web application is its navigation topology, which in great measure determines the application's success. Therefore, Web application frameworks must capture such aspects for families of applications in each domain. This is achieved through generic navigational contexts.

Generic navigational contexts are one of the most important architectural components in Web design frameworks. Contexts are recurrent patterns in Web applications since they usually deal with sets of similar objects. We're now incorporating other navigation patterns into OOHDM-Frame to enhance its expressive power; among these, we highlight personalization features. We are also developing XML specifications of frameworks, consistent with OOHDM XML specifications that already have been defined. Finally, we are investigating methods for framework design based on requirements that are common to applications in a particular domain.

Development, delivery, and maintenance times in the Web domain require reuse-centric approaches. The systematic reuse of semicomplete design structures, as described by Web design frameworks, is a key approach for maximizing reuse in Web application development. **MM**

References

1. D. Schwabe and G. Rossi, "An Object-Oriented Approach to Web-Based Application Design," *Theory and Practice of Object Systems (TAPOS)*, special issue on the Internet, vol. 4, no. 4, 1998, pp. 207-225.
2. S. Ceri, P. Fraternali, and A. Bongio, "Web Modeling Language (WebML): A Modeling Language for Designing Web Sites," *Proc. 9th Int'l Conf. the WWW (WWW9)*, Foretec Seminars, Reston, Va., 2000.
3. F. Garzotto et al., "Modeling-by-Patterns of Web Applications," *Proc. Int'l Workshop on the World Wide Web and Conceptual Modeling*, Lecture Notes in Computer Science 1727, Springer Verlag, Berlin, 1999, pp. 293-306.
4. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Reading, Mass., 1995.
5. M. Fayad, D. Schmidt, and R. Johnson, eds., *Building Application Frameworks*, Wiley & Sons, New York, 1999.
6. G. Rossi, F. Lyardet, and D. Schwabe: "Patterns for Designing Navigable Spaces," *Pattern Languages of Programs 4*, Addison Wesley Longman, Reading, Mass., 1999.
7. W. Pree, *Design Patterns for Object-Oriented Software*, Addison Wesley, Reading, Mass., 1994.
8. D. Schwabe et al., "Web Design Frameworks: An Approach to Improve Reuse in Web Applications," *Proc. WWW9 Web Eng. Workshop*, (Lecture Notes in Computer Science), Springer Verlag, Berlin, to be published in 2001.
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, 1999.
10. J. Conallen, *Building Web Applications with UML*, Addison Wesley Longman, Reading, Mass., 2000.
11. M. Fontoura, W. Pree, and B. Rumpe, "UML-F: A Modeling Language for Object-Oriented Frameworks," *Proc. ECOOP 2000 (LNCS 1850)*, Springer Verlag, Berlin, pp. 63-82.

Daniel Schwabe is an associate professor in the Department of Informatics at Catholic University in Rio de Janeiro (PUC), Brazil. He has been working on hypermedia design methods for the past 10 years. He is one of the authors of HDM, the first authoring method for hypermedia, and of OOHDM, one of the mature methods in use by academia and industry for Web applications design. He earned a PhD in computer science in 1981 at the University of California, Los Angeles.

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), a computer science research lab in Argentina. 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 in the Web field. He earned a PhD in Informatics in 1995 at the Catholic University of Rio de Janeiro (PUC), Brazil.

Fernando Lyardet is a research and teaching assistant at LIFIA, and an advanced undergraduate student at La Plata University, in Argentina. He works with patterns and pattern languages for hypermedia and Web applications, and with CASE tools for Web information systems.

Luiselena Luna Esmeraldo is an independent consultant working at Bayweb Consulting in Brazil. She has worked on hypermedia and Web frameworks for two years. She obtained a master of science in informatics in 1998, from the Catholic University in Rio de Janeiro (PUC), Brazil.

Readers may contact Schwabe at the Dept. Informática, PUC-Rio, Brazil, email schwabe@inf.puc-rio.br.

2001 Editorial Calendar

January-March
Web Engineering: Part 1

Leaders in the field discuss new approaches and tools for developing, deploying, and evaluating Web-based applications and systems.

April-June
Web Engineering: Part 2

Part 2 further explores Web-based systems and picks up where Part 1 leaves off. Read about lessons learned and the latest advances in creating applications and systems for the Web.

July-September
Intelligent Multimedia and Distance Education

Top researchers discuss next-generation applications in fields such as artificial intelligence, virtual environments, interactive multimedia, e-commerce, and distance education.

October-December
Multimedia and Security

Join the experts as they discuss the goals and problems in designing secure multimedia environments in the 21st century. Learn about the latest advances in proposed solutions such as digital watermarking and cryptographic protocols.

http://computer.org/multimedia