Conceptual Modeling of Device-Independent Web Applications

Jaime Gómez and Cristina Cachero
University of Alicante

Oscar Pastor
Valencia University of Technology

Existing tools for building and deploying complex Web sites are inadequate for dealing with the software production process that involves connecting with underlying logic in a unified and systematic way. As a solution, we propose the OO-H method, an object-oriented software approach that captures relevant properties involved in modeling and implementing Web application interfaces.

The Web's characteristics—ubiquity, open standards, interlinking, easy access to information and services, and creating content easily—have created new business opportunities, such as e-commerce or online auction systems. Additionally, its characteristics have nurtured a need for both business-to-consumer and business-to-business applications inside the enterprise boundaries that take advantage of the latest techonologies. In light of ongoing technological developments, our platform strives for a lower software development and maintenance budget and a flatter learning curve.

We believe that existing software engineering approaches can successfully extend to new models that specifically gather Web-related characteristics. We based our decision on three assumptions:

1. Behavior in a Web application can no longer be restricted to simple information update operations, and so it must be addressed in a rigorous way.

2. Context information and domain behavior inside Web applications should be ideally addressed with already tested object-oriented software engineering methods, as in any other distributed application.

3. In spite of the same underlying logical layer, interface appearance and behavior greatly differ between Web and traditional applications.

Following the trend presented in Manolla[1] and Conallen,[2] our approach—known as the object-oriented-hypermedia (OO-H) method—looks at Web systems as unified software artifacts where structure, behavior, and presentation are all basic pieces that must be properly combined for a correct final software product. As its main contribution, the OO-H method provides a standard-based framework to capture all the relevant properties involved in the modeling and implementation of Web application interfaces. The OO-H method design process involves constructing two additional views, which complement those captured in traditional conceptual modeling approaches that comply with Unified Modeling Language (UML). The navigation view extends a class diagram with hypermedia navigation features, and the presentation view uses the different elements regarding interface appearance and behavior to model a series of interconnected template structures expressed in Extensible Markup Language (XML). An interface pattern catalog and a computer-aided software engineering (CASE) tool (see "CASE Tool" sidebar) complete our proposal. As a result, we obtain a device-independent, front-end specification. The software then automatically generates a Web interface that easily integrates with preexisting logic modules.

Overview

The OO-H method is a generic model providing designers with the semantics and notation necessary for developing Web-based interfaces and connecting them with previously existing application logic modules, thus facilitating applications migration. To achieve this goal, we based the OO-H method[3] on the information reflected in a UML-compliant[4] approach, known as the OO-method.[5] For our purposes (see Figure 1), the OO-method is an automated software production environment whose main constituents are

- a set of views to capture the system structure (statics) and behavior (dynamics), and
- a model compiler that generates the data

1070-986X/01/$10.00 © 2001 IEEE

sources and the logic modules in the desired implementation environment.

The OO-H method extends these views with two new complementary diagrams. The navigational access diagram (NAD) defines a navigation view, and the abstract presentation diagram (APD) gathers the concepts related to presentation. Both the NAD and the APD capture the interface-related design information with the aid of a set of patterns, defined in an interface pattern catalog integrated in the OO-H method proposal.

Following the OO-method philosophy, the OO-H method provides a model compiler that generates the Internet application front-end for the desired client platform and/or language (HTML, XML, and Wireless Markup Language, or WML). This extension provides a true three-tiered Internet solution, as observed in Figure 1.

OO-H method proposal

The OO-H method includes the following set of notations, techniques, and tools that make up a sound approach to the Web product modeling phase:

- a design process,
- a pattern catalog,
- a NAD,
- an APD, and
- a CASE tool that automates the development of Web applications modeled with the OO-H method.

CASE Tool

The CASE tool provides an operational environment that supports all the methodological aspects of the OO-H method. It simplifies the design and implementation of Web-based information systems from an object-oriented perspective, providing a comfortable and friendly interface for elaborating the OO-H method models. The most interesting contribution of this CASE environment is its ability to generate the Web application front-end in well-known industrial software development environments. Figure A shows a snapshot of the CASE tool. It captures a NAD instance for the sample discussion list application, which includes an abstraction of the participateInDList navigational target.

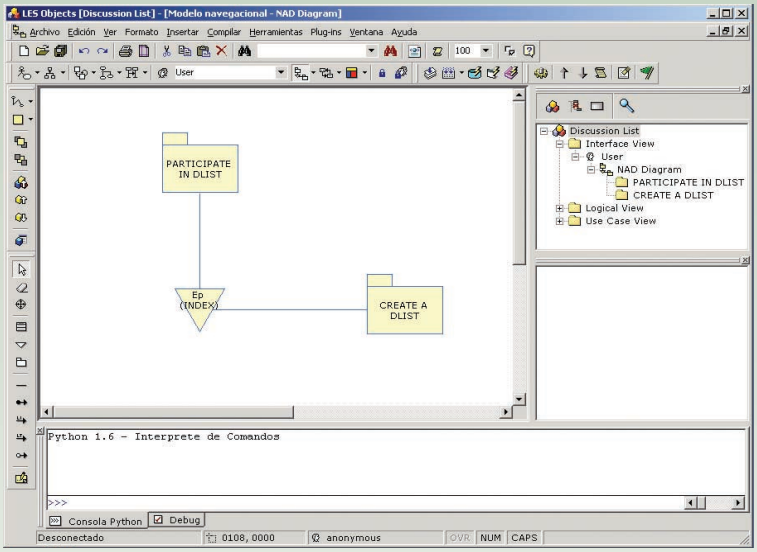
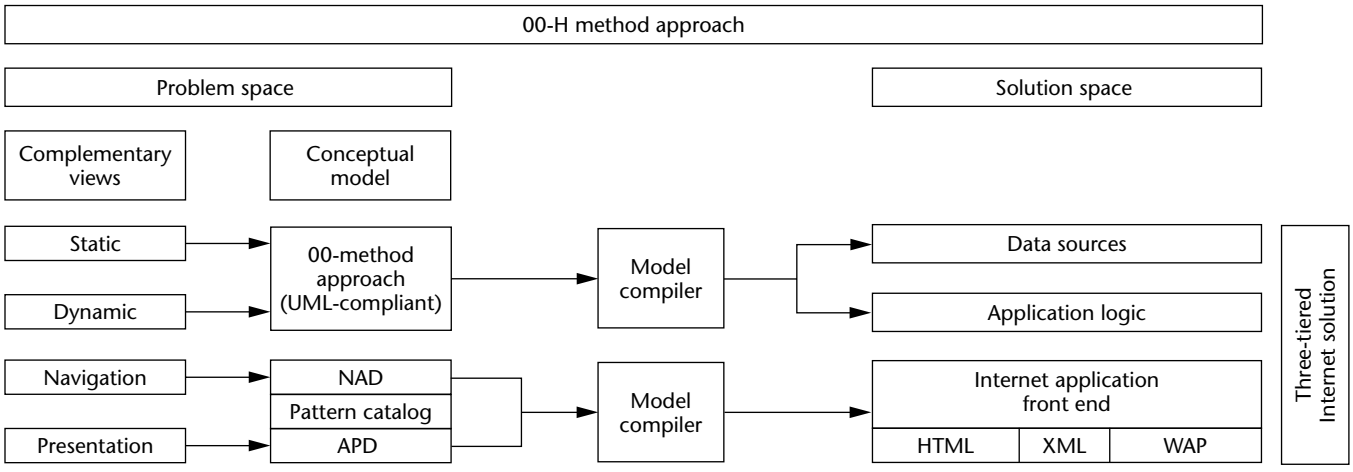


Figure A. The OO-H method CASE tool.

Design process

The design process defines the phases the designer has to cover for building a functional interface that fulfills the user requirements. As we

Figure 1. OO-H method: an overview.



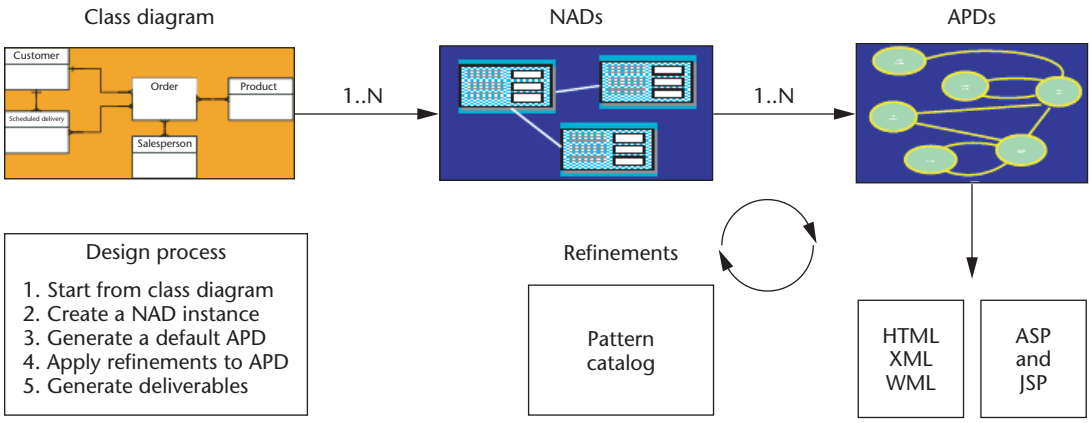


Figure 2. OO-H method: the design process.

see in Figure 2, the OO-H method design process departs from the domain information structure captured in a UML-compliant class diagram. From there, we model personalized (1..N) different NAD instances, one for each user type.

Each NAD instance reflects the information, services, and required navigation paths for the associated user's navigation requirements fulfillment. Once the NAD has been constructed, we can generate a default Web interface following a set of mapping steps. This automatic generation feature lets the designer shorten the time necessary to develop application prototypes. However, final implementations usually require a much higher level of sophistication, both from the visual and the usability point of view.

To improve the interface quality, the OO-H method introduces a second diagram—the APD, based on the concept of templates[6]—and directly derives its default structure from the NAD. To help the designer refine this structure while maintaining its quality, the pattern catalog contains a set of constructs that effectively solve problems identified within Web environments. This approach facilitates the reuse of design experiences and the consistency among the different interface modules and among application interfaces. Once we refine the APD, we generate a Web application front-end—either static or dynamic—for the desired environment, such as HTML, WML, active server pages (ASPs), and JavaServer pages (JSPs). Again, we can define different (1..N) APDs for the same NAD, reflecting different ways of visualizing the same navigation requirements. This independence from final implementation issues proves necessary in an environment where new appliances and languages for Internet access emerge constantly.

Pattern catalog

The OO-H method pattern catalog provides a hypermedia interface pattern language. We can see this language as a partially ordered collection of related patterns that work together in the context of hypermedia interfaces.[7] The patterns help capture the abstract interaction model between the user and the application.

We chose the pattern style defined in Buschmann et al.[8] for specifying the patterns. This style—largely based on the Alexandrian style[9]—suits abstract patterns with several possible ways of implementation. However, we enriched the implementation specification in order for them to drive the evolution of the diagrams: each pattern has one of its possible implementations set to "default" to help the designer obtain the desirable interface features with minimum effort.

Also, each implementation has an associated transformation rule, which is expressed in a syntax resembling Object Constraint Language (OCL).[10] These transformation rules can be instantiated and applied at different levels (from page level to schema level), and they drive the changes in the diagram where we apply the patterns. These changes include the creation of new pages, the redirection of links among pages, or the creation of new page dependencies. Furthermore, to improve its usability, the catalog contains a "Sample Usage" section that offers working Web examples where the catalog is successfully applied.

One of the main features of our catalog is that it is user-centered (as is the rest of the model). In other words, the granularity at which the patterns are described provides the designer with additional mechanisms to fulfill the user requirements. The patterns included in the catalog offer

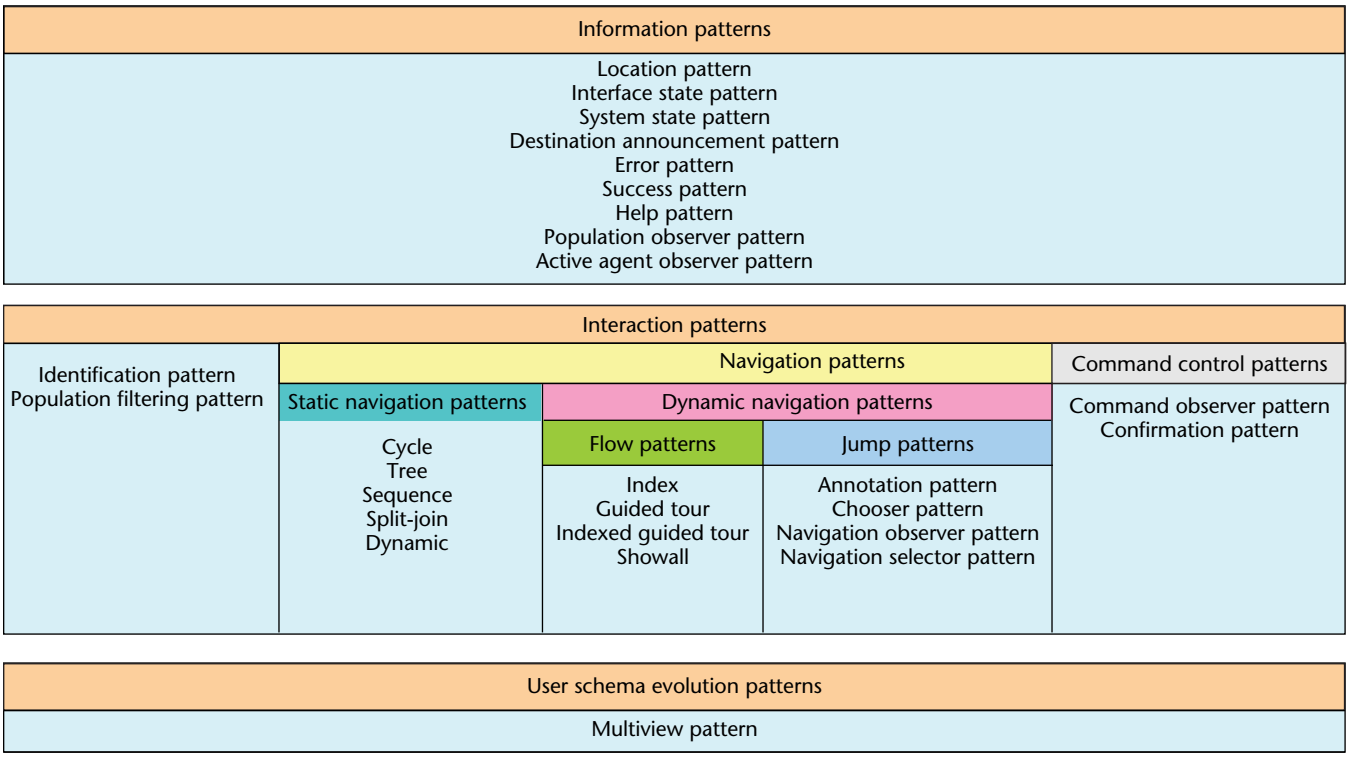


Figure 3. Pattern catalog.

alternative solutions to well-known hypermedia problems, considered from the user's point of view. Furthermore, its use lets the designer choose the most suitable among a set of alternative implementations, depending on the target application domain and on the designer's experience.

We show the structure of the catalog in Figure 3. The main categories are

1. *Information patterns*. Provide the user with useful application context information. One of the most relevant examples is the "location pattern" whose definition we present below.

2. *Interaction patterns*. Involve user-interface communication issues regarding both functionality and navigation.

3. *User schema evolution patterns*. Cover structural advanced features. As an example, we can cite the multiview pattern, which lets the designer present two or more simultaneous views of the information.

In the OO-H method, we can apply different sets of patterns to the two different diagrams of the modeling process—that is, the NAD and APD diagrams. At the NAD level, we can apply patterns related to user information selection and navigation behavior. At the APD level, however, we can apply patterns that provide the interface with nonmandatory additional features, which aim to improve its usability. As an example, a simplified definition of the location pattern follows:

1. Name: location pattern

2. Application level: APD

3. Context: A component gives information about the location context of the user inside the application.

4. Problem: The lack of a user mental schema when navigating through hypertext pages can cause the lost in the hyperspace syndrome. To avoid this problem and improve the usability of the interface, we need a mechanism to provide location information. We associate two forces with this problem:

- The path used to reach the information components isn't known a priori.

Acronyms

APD: abstract presentation diagram
ASP: active server page
DOM: document object model
DTD: document type definition
IDC: Internet database connector
JSP: Javaserver page
NAD: navigational access diagram
OCL: Object Constraint Language
OO: Object-Oriented
OO-H method: Object-Oriented-Hypermedia method
UML: Unified Modeling Language
UIML: User Interface Markup Language
WML: Wireless Markup Language
XML: Extensible Markup Language
XSL: Extensible Stylesheet Language

- The location component should be loosely coupled—the system view shouldn't depend on details of the location components.

5. Solution: Implement a mechanism in which a change on the view causes a change propagation that restores the consistency between the view and the location component.

6. Default implementation: Associate a meaningful header and footer to each group of related pages (see the "Abstract Presentation Diagram" section).

7. Other implementations: Adapt the observer pattern[11] to hypermedia environments.

Navigational access diagram

For a more general perspective of the approach, we'll use a discussion list management system as an example. As a basic explanation (for reasons of brevity), we assume the list manager system contains several discussion lists dealing with different Web technology topics, and the system forms each list by a set of hierarchically ordered messages relating to each other through a parent–child unary relationship. The discussion list user can read all the messages included inside any lists and reply to any of them.

We previously noted that one or more NADs capture the navigation model. Designers should construct as many NADs as different views of the system are required, and they should provide at least one different NAD for each user type (agent type) allowed to navigate through the system. A NAD is based on four types of constructs: navigational classes, navigational targets, navigational links, and collections. Also, when defining the navigation structure, designers must consider some orthogonal aspects, such as the desired navigation behavior, the object population selection, the order in which objects should be navigated, or the cardinality of the access. We capture these features by different kinds of navigation patterns and filters associated with links and collections. We further develop these concepts below.

- *Navigational classes*. Enriched domain classes whose attributes and method visibility have been restricted according to the user access permissions and navigation requirements. A sample enrichment is the differentiation among three types of attributes: V-attributes (visible attributes), R-attributes (referenced attributes, which are displayed after a user demand), and H-attributes (hidden attributes, which are only displayed when an exhaustive system population view is required—for example, for code refinement reasons).

  In our example (see Figure 4), we model two domain classes. The discussion list class defines the discussion topics available in the discussion list manager system, while the messages class contains the different messages and replies sent by the users. The discussion list class has a single attribute, the nameDList attribute, modeled as a V-attribute. This name unambiguously identifies each discussion topic inside the interface, and so it must always be present. Besides its identifying attribute titleMsg, the messages class has a field named textMsg, modeled as an R-attribute. This fact implies that the user has to explicitly click on its reference to read the message.

- *Navigational targets*. Group the elements of the model that collaborate in the coverage of each user navigational requirement.

  In the example (see Figure 4), we observe how there's a single navigational requirement, participateInDList, that contains both the information and the services needed for the user to effectively navigate the system.

- *Navigational links*. Define the navigation paths the user can follow through the system. They may have both a navigation pattern and a set of navigation filters associated, which provide additional information to construct the user

navigation model. We can distinguish among four link types: I-links (internal links) define the navigation path inside the boundaries of a given navigational target; T-links (traversal links) are defined between navigation classes belonging to different navigational targets; R-links (requirement links) point at the starting navigation point inside each navigational target; and S-links (service links) show the services available to the user type associated to that NAD.

In Figure 4, we observe how the structural underlying relationship between the *discussion list* and the *messages* navigational classes give a semantic meaning to the I-link defined between them. This link has a *showall-dest* navigation pattern associated to it, which causes all message instances contained in a given discussion list to appear together. The destination modifier (*dest*) reflects the fact that traversing that link implies a step further in the navigation path, which in our implementation environment means that the name of the discussion list and its related messages appear on different pages. The R-link labeled *enter* points at the navigation class from which the user will start the navigation inside that navigational target. Finally, an example of S-link associates with the *replymessage* service inside the class messages, which lets the user answer any previous message. At execution time, the user must introduce all service parameters (in our example, the answer title and text) not fed to the service by means of filters associated to the S-links.

- *Collections*. Possibly hierarchical structures defined on navigation classes or navigation targets. They provide the user with new ways of accessing the information. The OO-H method defines several types of collections, including C-collections (classifier collections), where the object population criteria is predefined, and S-collections (selector collections), where the user should introduce the criteria at execution time, thus requiring a new form to be generated on the fly from the parameters specified for such a collection.

  In Figure 4, we see a single C-collection, drawn as an inverted triangle and associated to the NT participateInDList. This C-collection determines the entry point to the Web application.

For more information on the diagram, see Gómez, Cachero, and Pastor.[3]

Commercial interfaces tend to require a greater level of sophistication than that provided by the NAD diagram, regarding both appearance and usability features. To refine the interface, the OO-H method defines another diagram, the APD.

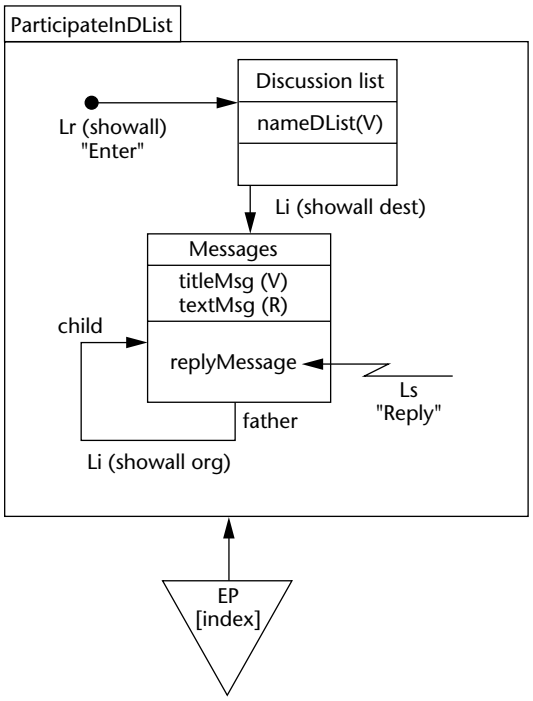


Figure 4. NAD diagram of the discussion list system.

Abstract presentation diagram

We adopted a template approach for the specification of the visual appearance and page structure of the Web. To separate the different aspects contributing to the final interface appearance and behavior, we defined five types of templates, expressed as XML documents.[12] To define the tags and the structure of the document, we associate a document type definition (DTD) with each type of template. (Note that a new proposal, called XML-Schema, is being discussed at http://www.w3.org/XML/ as an alternative to the DTD language.)

Template types. The five main template types defined in the OO-H method are

1. tStruct: its instances define the information that has to appear on the abstract page.
2. tStyle: its instances define features such as physical placement of elements, typography, or color palette.
3. tForm: its instances define the data items required from the user to interact with the system.

4. tFunction: its instances capture client functionality. They're based on the document object model (DOM) specification (see http://www.w3.org/XML/), which aims to give a platform and language-independent interface to the structure and content of XML and HTML documents. It also seeks to standardize an interface for these objects for navigation and document processing. The functions defined in this kind of template are language-independent, so they must be mapped to a target language (either JavaScript, which captures HTML-specific components of the DOM, or another) to become operative.

5. tWindow: its instances define a set of simultaneous views available to the user.

For reasons of brevity, we only show the tStruct DTD. A DTD specifies the set of rules defining a valid XML document. It thus defines the tags that appear in any template belonging to that type, the elements it may contain, and its attribute assignments. It also contains other information, such as processing instructions, a document type declaration, comments, and so on.

```
<!ELEMENT collection (object+|link*)>
<!ATTLIST collection
  format (ulist|olist) "ulist"
  style CDATA #IMPLIED
>
<!ELEMENT link (call)*>
<!ATTLIST link
  name ID #REQUIRED
  type (string|button|image|menuitem
       |automatic) "string"
  show (here|new) "new"
  pointsTo
(tForm|tWindow|tFunction|tStyle
       |tStruct|command) "tStruct"
  dest CDATA #REQUIRED
>
<!ELEMENT object (attrib+|call*)>
 <!ATTLIST object
  type CDATA #REQUIRED
  style CDATA #IMPLIED
>
<!ELEMENT attrib (call)*>
<!ATTLIST attrib
  name ID #REQUIRED
  type
(string|date|text|integer|anchor|…)
       #REQUIRED
  order (yes|no) "no"
>
<!ELEMENT call EMPTY>
<!ATTLIST call
  event (onChange|onClick|onLoad|…)
       #REQUIRED
  function CDATA #REQUIRED
>
<!ELEMENT label EMPTY>
<!ATTLIST label
  text CDATA
>
```

Looking at this DTD, we can infer the following:

1. A tStruct consists of a set of collections (which must contain at least one object) and any number of links. Collections have a default format associated (unordered list) and could also have a user-defined style (captured in the model inside a tStyle template).

2. A call, which can associate with every element of the page, defines the trigger condition for the interface to perform an action.

3. A link can have zero or more calls associated to it. A name, a type, and a set of attributes related to its behavior define a link (where it's going to show the destination page, which one is the destination page, and its type). In the APD, we show only links with the show attribute set to new (meaning that they point to a new abstract page).

4. An object contains one or more attributes and zero or more calls. It also has both a type (class to which it belongs) and a style (possibly missing) associated to it.

5. An attribute can have any number of calls associated to it. It also has a name (required), a type, and a Boolean order value that indicates whether the order in which the objects appear on the screen depends on the value of that attribute.

6. A tStruct page can have any number of labels that capture the static text appearing in the interface.

We can derive the default template structure from the information captured in the NAD com-

bined with a set of defaults defined for the undefined values.

Default APD. The OO-H method defines a set of steps for the mapping of the different elements contained in the NAD diagram into the equivalent elements in the APD.

The default APD gives a functional but rather simple interface, which will probably need further refinements to become useful for its inclusion in the final application. It can, however, serve as a prototype for validating that user requirements were captured correctly. In our example (see Figure 5), the automatically generated pages are

1. the homepage, derived from the entry point collection,
2. a DList page, of type tStruct, that gathers the different discussion list objects managed by our system,
3. a message view page that includes all the messages related to each discussion topic, and
4. a reply message page, of type tForm, for the user to introduce the required parameters for the insertion of a reply.

Also, the different links defined in the NAD (see Figure 4) provide the information necessary for the connection of these APD pages.

We describe the main mapping steps that drive this default APD generation in the sidebar "Default APD Mapping Rules."

APD refinement. The refinement process modifies the default APD structure. We simplify this process with the application of a series of APD-related patterns captured in the catalog. A transformation rule describes the changes introduced in the APD when the designer decides to apply a pattern (expressed in an OCL-like syntax[10]) associated to any of its possible implementations. As an example, Figures 6 and 7 (next page) illustrate the definition and instantiation of the location pattern transformation rule.

Every APD has an identifying name that acts as a root element and derives every other element from the diagram. That root element is of type APDSchema and, in our example (Figure 7), corresponds to the DList element (name of the APD diagram). Pages are added to the schema by means of an addAPDPage service associated to the APDSchema object. Once we add a new page to the schema, we must define its name and its type. In addition, these new pages can have any number of functions associated to them (for example, as a *mail to Webmaster* service). We must previously define

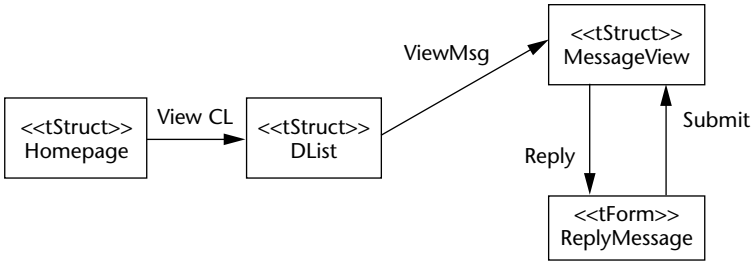


Figure 5. Default APD for the discussion list system.

Default APD Mapping Rules

To speed up the Web interface design process, the OO-H method defines a set of NAD2APD mapping rules that—when applied—cause a fully operative default APD diagram to be generated automatically from the information gathered at the NAD level. Namely, the mapping rules are

- V-attributes, I-links, T-links, and R-links appear as elements of the tStruct page.
- C-collections and S-collections are static trees. A tStruct single abstract page defines them—the page contains a tree-like structure made up of link elements pointing to other tStruct elements. An S-collection also implies the possibility of creating a new tForm template, if the filter associated with it involves any user-dependent value.
- S-links may generate a previous tForm abstract page that contains the parameters for the user to introduce all the commands involved in the service. They must contain a link element that points to a command (considered as either a method or a transaction in the OO-method). If the command returns anything, the software generates another tStruct page with the structure that the return value defines (either as an object or a set of objects).
- R-attributes cause a new tStruct abstract page to appear on the diagram and a new link element pointing to it on the original one. The new template contains all the R-attributes to be shown plus a meaningful reference to the original object.
- NAD navigation patterns are a set of link elements defined on a single tStruct page. They capture all indexes, guided tours, indexed guided tours, and showall patterns in the APD.
- The software encloses all objects to be shown in collections. The software defines default user-defined data validations for the main attribute types. Call elements with onChange event functions implement these validations. The elements associate with validation functions that depend on the attribute type.

```
//Create the new pages
[<APDSchema>->addAPDPage(varpage);
     varpage.name=<pagename>
varpage.type='Tstruct';

//associated tfunctionlibfunctions
     (nonmandatory)
[fvarpage= <APDSchema>->
     select(type='tFunctionLib');
<APDSchema>->select
     (name=varpage.name)->
     AddFunction
     (fvarpage.function);]

//associated tstyle page
     (nonmandatory)
[svarpage=<APDSchema>->
     select(name=<stylename>);
varpage.IncludeStyle(svarpage);]?

//create links to other pages
     (nonmandatory)
[otherpage=<APDSchema>->
     select(name=<otherpagename>);
varpage->AddLink(<othervarpage>);]*

]+

//Link new pages to existing APD
     structure
[<pagecontext>->select
     (type='Tstruct' or
     type='Tform')->Include
       StructPage (varpage,
     position) ||
[<pagecontext>->select
     (type='Tstruct' or
     type='Tform')->Include
       StructPage (varpage,
     position)]+
```

Figure 6. Location pattern transformation rule.

```
//Create the new pages
Dlist->addAPDPage(h); h.name'head'
     h.type='Tstruct'
Dlist->addAPDPage(h); f.name'foot'
     f.type='Tstruct'

//Add foot functions (they must be
     previously defined in the
     tFunctionlib Page)
fl=Dlist->select
(type='tFunctionLib');
Dlist->select(name='foot')->
     AddFunction
     (fl.mail);
Dlist->select(name='foot')->
     AddFunction
     (fl.back);

//add a link from head to home page
home=Dlist->select(name='homepage');
Dlist->select(name='head')->
     AddLink(home);

//Link new pages to existing APD
     structure
Dlist->select(type='Tstruct' or
     type='Tform')
     ->IncludeStructPage
(h,Beginning); Dlist->select(type='Tstruct' or
     type='Tform')->IncludeStructPage (f,End);
```

Figure 7. Transformation rule instantiation for the location pattern.

these functions in a function repository made available to the APD by a tFunctionLib abstract page. Also, we could decide to associate a style to these newly created pages and/or some links to other pages, such as the site's homepage.

In Figure 8, the application of the transformation rule instantiation (Figure 7) to the DList default APD shown in Figure 5 causes two new pages to appear: a header and a footer page. These pages relate to every tStruct and tFunction page in the diagram, which essentially means that every page generated from that APD shares the same layout context. The foot page has two related functions: mail and back, whose behavior the abstract tFunctionLib page describes. Also, the transformation rule adds a link from the head construct to the application homepage.

In our example (Figure 8), the application of an error pattern similarly causes a new abstract tStruct page to appear on the schema, as well as a set of dependency arrows to specify the pages where an error might occur. Furthermore, the designer has applied the multiview pattern, which creates a tWindow construct.

Generally speaking, patterns might cause any kind of abstract page to appear or be modified. As a further example, applying the confirmation pattern would cause a confirm function to be included before requesting any change of the type specified to the system. Therefore, the content of the function page—where we include the client logic—would change.

In the OO-H method, you can also choose whether to make the chosen patterns visible. For example, the physical pages and the set of arrows

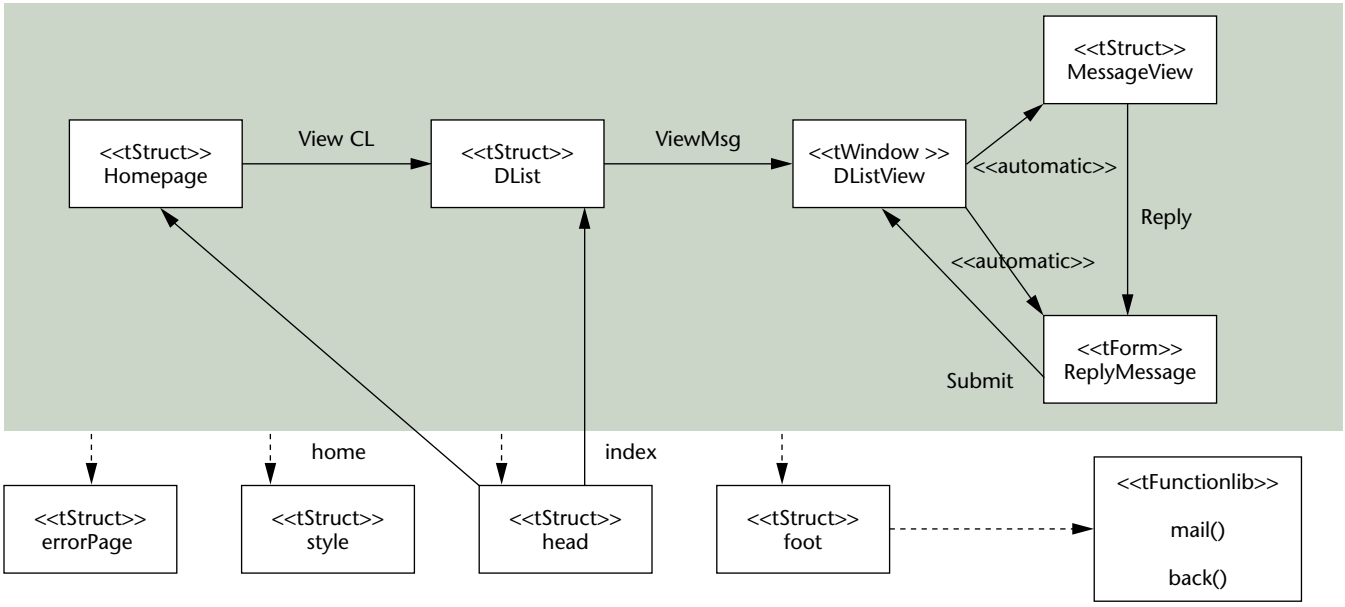


Figure 8. APD refinement of the discussion list system.

implied by the location pattern don't provide much meaningful information, so you could decide not to show them on the diagram. The same would happen with the error pattern. Using implicit patterns clarifies the schema and avoids overwhelming the designer with excessive data.

The generated page template for the DList abstract page after applying the refinements is

```
<?XML version="1.0"?>
<!DOCTYPE tStruct SYSTEM "tStruct.dtd"
     encoding="UTF-8">
<tStruct>
  <label style="" text="Available
       chats" />
  <link name="error" type="automatic"
       show="new" pointsTo="tStruct"
       dest="errorPage">
  </link>
  <link name="head" type="automatic"
       show="here" pointsTo="tStruct"
       dest="head">
  </link>
  <collection format="ulist" style="">
       <object type="discussion list">
            <attrib name="nameDList"
                 type="STRING">
            </attrib>
            <call event="onClick"
                 function="validate">
       </object>
  </collection>
  <link name="foot" type="automatic"
       show="here" pointsTo="tStruct"
                 dest="foot">
  </link>

</tStruct>
```

After refining the APD, we feed the device-independent modeled interface features to a model compiler (not discussed here) that has the target-environment knowledge that lets it generate an operational Web interface.

The generated front-end. We generated the interface from the APD in Figure 8, which corresponds to the discussion list. We show the list system in Figure 9 (next page).

The process is as follows. First, the model compiler tool looks for the page template derived from the application entry point (see Figure 9a). Note that every page of the diagram has the same header/footer associated, which provides the interface with a common visual context.

When the user clicks on the enter link (defined with the *show new* navigation pattern associated to it), the software performs an object population of the DList tStruct and shows the materialized HTML page (see Figure 9b). Again, when the user clicks on the link associated to the desired discussion list (also defined as a link with the attribute show set to new) the tWindow abstract page appears. This template captures a behavior different from tStruct's—instead of populating the template with objects, it defines the characteristics of the two different and simultaneously available views of the system.

Figure 9. (a) Discussion list entry point. (b) Available discussion lists. (c and d) Adding an opinion to the list.

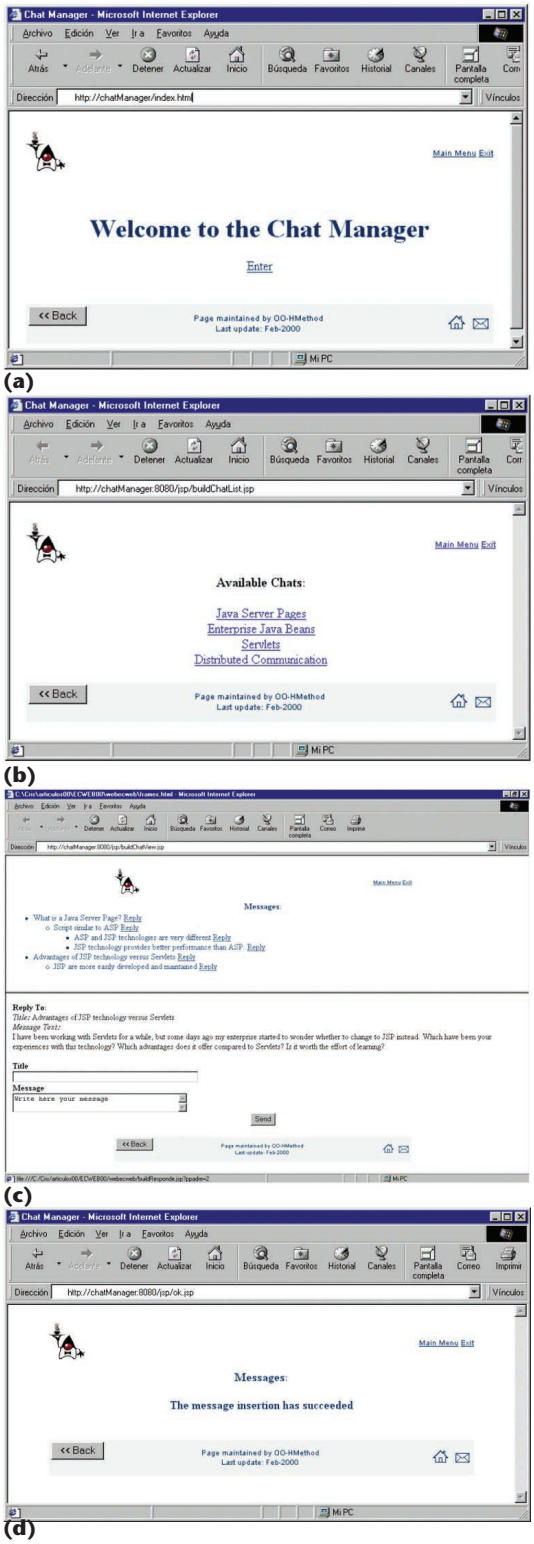


Figure 10. Discussion list system in a WAP device.

The tWindow therefore defines two links whose type is set to automatic (because they don't require user interaction). These links are in charge of loading the two current views: the messages kept on the system (again a tStruct abstract page) and a form with fields that the user must fill out to add a new message to the application (see Figure 9c).

The addMessage function returns a Boolean value, which makes the final confirmation message appear once the operation is successfully completed (see Figure 9d).

We developed the sample application using JavaServer pages and Java Bean components (see http://java.sun.com) as the server technology and HTML as the client technology.

The model compiler design lets code be generated for other environments. Figure 10 shows a snapshot of the interface generated from the same specification for a different appliance (in this case, a WAP device).

Comparison with related work

Interface designers have traditionally tackled their tasks using appliance-specific languages directly. That is the case of many commercial applications and technologys, such as Internet Database Connection (IDCs), ASPs (Microsoft), or Cold Fusion (Allaire). This approach has many disadvantages (see http://www.uiml.org), including lack of support for several target environments (mobile devices, voice, speech), need for the designer to perform error-prone activities (direct access to databases, explicit page linking), high cost of prototyping, difficulty in reasoning and extracting knowledge about the most suitable solutions to detected usability problems, or the difficulty of evaluating interfaces.

We identified several proposals that provide partial solutions to these problems and use template-based Web engineering tools. Of particular interest are those that use XML instead of being HTML-oriented. This fact provides them with a greater level of flexibility. For example, MyXML (see http://www.infosys.tuwien.ac.at/myxml/) is a template engine based on XML and Extensible

Stylesheet Language (XSL) for generating either static or dynamic sites. User Interface Markup Language (UIML—see http://www.uiml.org) is another XML-compliant markup language that specifically tackles the problem of a device-independent description of user interfaces. It provides an event mechanism to communicate with underlying logic modules. It also proposes some new orthogonal system views for functionality and interface widgets, and the whole definition of the interface (from structure to styles to presentation to rendition for a given device) is encapsulated in a single document. Other systems, such as Strudel,[13] have gone one step further and integrated heterogeneous information sources as a previous step from which to build the structure of the Web application and to generate the HTML representation of pages. This aspect isn't considered in our proposal, as our point of departure is a structured object-oriented information model.

These solutions clearly separate logic, layout, and structure, thereby facilitating code reuse and site maintenance. Also, they explicitly give support to dynamically generated Web content. Furthermore, UIML and MyXML provide mechanisms for connecting with underlying logic. However, they're still closer to the solution space than to the conceptual (domain problem) space, even if they adopt a more abstract approach than commercial applications.

We also base the OO-H method on a set of XML-compliant specifications. The main contribution of the OO-H method at this point is our extensible taxonomy of templates, which introduces a broader separation of the different concepts involved in the construction of a Web interface. For example, the tFunction view of the interface, adds a client–logic abstract definition component to the other views of the system, and therefore facilitates the maintenance and/or change of target language for this client logic. As a further advantage, these specifications try to keep as close as possible to some well-known XML standard proposals. This limits the designer's need to learn new languages and specifications to understand the semantics underlying the templates. Following our example, the OO-H bases the abstract definition of the client logic in the OO-H method on the DOM interface model.

The OO-H method also increases the level of abstraction at which it defines a Web interface and is, therefore, much closer to another group of proposals that tackle the modeling problem from a conceptual point of view. Within these approaches, the UML community has presented its proposal for modeling Web architectures.[2] Use of a standard UML notation and the distinction between server-side aspects and client-side aspects are two of its main features. Although using prototypes adds flexibility to the way the designer can approach the modeling process, this approach has more to do with the physical software module structure than with the logical analysis of the application structure. Consequently, it greatly differs from our approach.

Other well-known proposals include Araneus,[6] AutoWeb,[9] HDM2000,[14] or OO-HDM,[15] which clearly separate structure, navigation, and presentation features. They also share many of the concepts identified in the classical hypertext theory—such as that of collections, navigational classes, or perspectives—with the OO-H method. Furthermore, our approach is—like OO-HDM[15]—user centered (as it relies on user requirements) and object oriented. This allowed us to use the knowledge domain implicit in object-oriented models to improve the interface usability. The inclusion of a pattern catalog and the way these patterns can apply to the different diagrams to modify both the model and the final implementation are closely related to this usability concept and are one of the main contributions of our method.

Another characteristic of other approaches is that they're based on hypermedia systems, which target to the information navigation and visualization.[16] As a result, they don't deal with complex functionality, either on the client or on the server, which current Web environments need and we explicitly tackle in the OO-H method.

The appearance of new technologies and standards such as XML—which were present from the very beginning of the OO-H method—caused many of those proposals to evolve. This proved true for Araneus-XML[17] (Araneus model translation to XML), WebComposition[18] (an extension to the OO-HDM model), and WebML[19] (AutoWeb evolution).

With regard to business logic interaction, all of these new proposals already integrate some simple functionality (such as update operations). Developers are now working on the specification of complex business logic and rules. In contrast, the OO-H method centers on defining and integrating Web interfaces with existing business modules. As a result, the OO-H method specifically provides mechanisms for invoking services, selecting the possibly complex parameters to be passed to a given method, dealing with invocation errors, and so on. Finally, starting from a UML-compliant conceptual modeling approach

facilitates the OO-H method's integration with other proposals.

Discussions and future work

Web applications development methods are facing the problems traditionally associated with definition of a robust and sound software production method—how to go properly from specification to implementation. Most of the current tools focus on the solution space, forgetting the widely accepted weight of modeling in the process of producing quality software.

Web sites have evolved from merely hypermedia information repositories to hypermedia distributed applications (generally known as Web applications). We must introduce new conceptual features in the modeling step, especially those related to navigation and presentation that are basic in Web environments. Consequently, accepting that we need conceptual modeling for developing correct Web applications, we require new tools to design and implement a software production process according to these ideas.

We call these new scenarios e-modeling software production environments, meaning that a process for applying conceptual modeling techniques to the development of Web applications is defined. To properly face this problem, two activities traditionally performed in isolation must integrate—modeling the operations and modeling the hypermedia.

Moreover, this integration must take into account the existence of fully tested applications that need to migrate to this new development environment. Using a conventional (UML-like) OO conceptual modeling approach, the needed expressiveness can be introduced in the model to properly specify navigation and presentation features. All this information must be complemented properly with a precise methodological guidance to go from the problem space (represented by the conceptual modeling step) to the solution space (represented by the final software product). We must provide integration mechanisms to connect with preexisting business modules. Furthermore, as Web technology is in continuous evolution, providing device-independent capabilities will be a constant requirement, supporting the idea of implementing the same conceptual schema in different target devices, depending on the customer choice.

During the next few years, a significant number of CASE tools will appear to provide users with practical solutions for all the previous ideas. Furthermore, the tools will require model-based code generation techniques to translate conceptual schemas into their corresponding software representations, which will be built using the selected Web development technology. One of the most relevant problems we'll have to solve in this context is how to properly connect navigational and presentation features with a correct functionality. More and more, to have a pleasant Web site (following the Web aesthetic standards) won't be enough—the system functionality will be required by customers, and derived from their requirements. If we have a look on the software engineering history, we can conclude that—once again—defining a rigorous software production method (including the Web particularities from the conceptual modeling point of view) is a basic necessity for assuring a quality final product. **MM**

Acknowledgements

We'd like to thank Antonio Párraga for his help and support in implementing important parts of our tool. Dr. Gómez would also like to thank Dr. Isidro Ramos (leader of the PLIS group of UPV) for giving him the opportunity to join his research group in 1997.

References

1. F. Manola, "Technologies for a Web Object Model," *IEEE Internet Computing*, Jan. 1999, pp. 38-47.
2. J. Conallen, "Modeling Web Application Architectures with UML," *Comm. ACM.*, vol. 42, no. 10, Oct. 1999, pp. 63-70.
3. J. Gómez, C. Cachero, and O. Pastor, "Extending a Conceptual Modelling Approach to Web Application Design," *Proc. 12th Int'l Conf. Advanced Information Systems* (CAISE 00), Lecture Notes in Computer Science 1789, Springer-Verlag, Berlin, June 2000, pp. 79-93.
4. *OMG Unified Modeling Language Specification*, http://www.rational.com/uml/, June 1999.
5. O. Pastor et al., "From Object-oriented Conceptual Modeling to Automated Programming in Java," *Proc. Int'l Conf. Entity Relationship Approach* (ER 98), Lecture Notes in Computer Science 1507, Springer Verlag, Berlin, 1998, pp. 183-196.
6. G. Mecca, et al., *The Araneus Guide to Web-Site Development*, tech. report, Univ. of Rome, Rome, Mar. 1999.
7. G. Rossi, D. Schwabe, and A. Garrido, "Design Reuse in Hypermedia Applications Development," *Proc. 8th ACM Conf. Hypertext*, ACM Press, New York, 1997, pp. 57-66.

8. F. Buschmann et al., *A System of Patterns*, Wiley & Sons, New York, 1996.
9. C. Alexander, *The Timeless Way of Building*, Oxford Univ. Press, Oxford, 1979.
10. J. Warmer and A. Kleppe, "The Object Constraint Language," *Precise Modeling with UML,* Addison-Wesley, Reading, Mass., 1998.
11. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, Reading, Mass., 1995.
12. S. McGrath, *XML by Example: Building E-Commerce Applications*, Prentice Hall, Upper Saddle River, N.J.,1998.
13. M. Fernández, D. Suciu, and I. Tatarinov, "Declarative Specification of Data-Intensive Web Sites," *Proc. Usenix Conf. Domain Specific Languages*, ACM Sigplan Notices, Boone, March 1999, pp. 135-148.
14. L. Baresi, F. Garzotto, and P. Paolini, "Web Sites to Web Applications: New Issues for Conceptual Modeling," *Conceptual Modeling for E-business and the Web*, Lecture Notes in Computer Science 1921, Springer-Verlag, Berlin, 2000, pp. 89-100.
15. D. Schwabe and R. Almeida Pontes, "A Method-Based Web Application Development Environment," *Proc. 8th Int'l World Wide Web Conf.* (WWW8), position paper, Web Engineering Workshop, 1999, http://www. budhi.uow.edu.au/web-engineering99/accepted_papers/schwabe.pdf//.
16. M. Bieber and C. Kacmar, "Designing Hypertext Support for Computational Applications," *Comm. ACM*, vol. 38, no. 8, 1998, pp. 99-107.
17. G. Mecca, P. Merialdo, and P. Atzeni, "Araneus in the Era of XML," *IEEE Data Engineering Bulletin*, vol. 22, no. 3, Sept. 1999, pp. 19-26.
18. H. Gellersen and M. Gaedke, "Object-Oriented Web Application Development," *IEEE Internet Computing*, vol. 3, no. 1, Jan./Feb. 1999, pp. 60-68.
19. S. Ceri, P. Fraternali, and A. Bongio, "Web Modeling Language (WebML): A Modeling Language for Designing Web Sites," *Proc. 9th Int'l World Wide Web Conf.* (WWW9), Foretec Seminars, Weston, Va., May 2000, http:www9.org/w9cdrom/index.html.

Jaime Gómez is a professor of software engineering at the Computer Science School at the University of Alicante. He received a BS in computer science from the Technical University of Valencia in 1992 and a PhD in computer science from the University of Alicante in 2000. His current research interests include object-oriented conceptual modeling, Web engineering, model-based code generation and component-based development. For more than five years, his research group has developed methods and tools to facilitate software development in the industry. He is the author of two journal papers and more than 25 papers presented in international conferences like the Conference on Advanced Information System Engineering (CaiSE) and the International Conference on Conceptual Modeling (ER).

Cristina Cachero is a research student at the Computer Science School at the University of Alicante. He received a BS in computer science from the University of Alicante in 1997. She currently performs research at the University of Milano, under the supervision of Dr. Stefano Ceri. Her research interests include object-oriented conceptual modeling, human-interaction computing, and Web engineering. She is the coauthor of five articles presented at various international conferences.

Oscar Pastor is a member of the Computation and Information Systems Department, Valencia University of Technology (Spain). He received his PhD degree from the Valencia University of Technology in 1992, after a research stay in HP Labs, Bristol, UK. He is currently a professor of software engineering at the Valencia University of Technology. His research activities involve object-oriented conceptual modeling, requirements engineering, information systems, Web-oriented software technology, and model-based code generation. He devotes considerable effort to issues of technology transfer from academia to industry. The author of more than 100 research papers in conference proceedings, journals and books, he has received numerous research grants from public institutions and private industries.

Readers may contact Jaime Gómez and Cristina Cachero at the Department of Information Systems and Languages, University of Alicante, c/ San Vicente s/n 03690 Alicante (Spain), e-mail jgomez@dlsi.ua.es or ccachero@dlsi.ua.es. Contact Oscar Pastor at the Department of Information and Computing Systems, Valencia University of Technology, Camino de Vera s/n Apdo. 22.012 E-46071 Valencia, Spain, e-mail Oscar Pastor at opastor@dsic.upv.es.