Chapter VIII

OO-H Method: Extending UML to Model Web Interfaces

Jaime Gómez
The Web Engineering Research Group, University of Alicante, Spain

Cristina Cachero
The Web Engineering Research Group, University of Alicante, Spain

ABSTRACT

The mostly "creative" authoring process used to develop many Web applications during the last years has already proven unsuccessful to tackle, with its increasing complexity, both in terms of user and technical requirements. This fact has nurtured a mushrooming of proposals, most based on conceptual models, that aim at facilitating the development, maintenance and assessment of Web applications, thus improving the reliability of the Web development process.

In this chapter, we will show how traditional software engineering approaches can be extended to deal with the Web idiosyncrasy, taking advantage of proven successful notation and techniques for common tasks, while adding models and constructs needed to capture the nuances of the Web environment. In this context, our proposal, the Object-Oriented Hypermedia (OO-H) Method, developed at University of Alicante, provides a set of new views that extend UML to provide a Web interface model. A code generation process is able to, departing from such diagrams

and their associated tagged values, generate a Web interface capable of connecting to underlying business modules.

INTRODUCTION

In the last few years, we have witnessed how existing and to come Web technologies have induced much more flexible distributed environments where new business opportunities have appeared but also new risks related to software development (Murugesan et al., 1999). Although the scientific community agrees that in order to keep the possibility of failure to a minimum, the development process for enterprise applications should evolve in a Web engineering manner, there is no agreement at how the core activities behind a sound Web application development process should be addressed. Some approaches, most coming from the hypermedia community, consider Web applications as information delivery systems, where only trivial functionality is offered (Mecca, Merialdo, & Atzeni, 1999; Schwabe & Almeida, 1999; Ceri et al., 2000). Others, mainly coming from the traditional Software Engineering field, regard Web applications as traditional distributed applications, and propose modeling approaches that make exclusive use of standard models and notation to capture the idiosyncrasy of this new platform (Conallen, 1999). Still other approaches consider the Web as a dynamic business Web (Gartner Group, Inc., 2001), where the application development consists of a process of communication and integration of Web services disseminated over the net and offered via (often) collaborating technologies such as UDDI (Universal Description, Discovery and Integration, 2001), DSML (Directory Services Markup Language, 2001), SOAP (Simple Object Access Protocol, 2001) or WSDL (Web Services Description Language, 2001).

We agree with Manola (1999), in that each of these trends partially addresses the nuances Web applications involve, and so a fusion of their respective points of view is needed in order to provide a cohesive solution to the Web application modeling process. On one hand, hypermedia modeling methods contribute a deep reflection on Web navigation and interaction issues, which are basic for a Web application to succeed. However, these sound navigation features are not enough, as Web applications must also provide the user with the (often complex) functionality they need, far beyond navigation. The modeling of functional requirements has already been partially addressed in a number of Advanced Software Production Environments, many based on the UML (UML Specification. V1.3., 1999) notation, that use Model Based Code Generation techniques (Bell, 1998) and have the support of "traditional" software engineering methods. The problem is that those traditional models,

methods and notations do not provide the mechanisms, models and constructs needed to capture the specific hypermedia semantics, and so are too cumbersome when used in isolation. Furthermore, functionality may be centered on a single server or be distributed over the Web, and it is here where the concept of dynamic business Web plays an important role: proposed models should be capable of abstracting the implementation issues far beyond the concept of client–server applications: the integration of services is a must in the to-come Semantic Web (Semantic Web, 2001), but existing proposals (User Interface Modeling Language, 2001; Gellersen & Gaedke, 1999) are still, from our point of view, too close to the solution space.

Our approach, known as the OO-H (Object-Oriented Hypermedia) method (Gómez, Cachero, & Pastor, 2001), centers on the authoring process (product development phase). It follows this "integrating" philosophy and extends traditional software engineering approaches (based on UML) with two new models, namely, (1) a navigation model and (2) a presentation model (further divided into an abstract presentation layer and a layout composition layer). The navigation model is built up departing from a UML-compliant use-case and a business class diagram. Use cases provide OO-H with the user-centered perspective which we think is necessary in Web environments, while the class diagram is a suitable mechanism not only to show the domain conceptual model but also to expose the interfaces that permit the connection with preexistent business logic modules, no matter where they are located or its internal nature (Web services, dll's, javaBeans, etc.). In the implementation phase, tagged values associated with these interfaces determine their nature and the most suitable access protocol to actually invoke the services and recover the return values. The process is iterative and incremental (as suggests the idiosyncrasy of Web applications[1]), and in each iteration, a refined version of the application is generated.

Special features, particular to any kind of Web application (including *e-commerce* applications), are addressed by means of frameworks and patterns that are gathered in an OO-H pattern catalog. Patterns capture practices proven successful to solve a given problem under different circumstances. The pattern mechanism allows the encapsulation of Web design knowledge and facilitates reuse. The OO-H case tool allows the addition of such patterns and frameworks in a dynamic way. Its use speeds up the development process while isolating the designer from implementation issues, likely to change in such a dynamic environment as the Web. Therefore, the designer is allowed to focus on more abstract features such as service chains, user profiles, usability features, and so on. The use of patterns is further discussed in section "View Refinement: Presentation Modeling".

Departing from the navigation model, a default abstract presentation model can be obtained in an automatic way. This abstract presentation model is composed of a set of XML templates that gathers the orthogonal views OO-H takes into account when dealing with Web interfaces, and which are, namely, structure, user interaction, navigation, client logic, page composition, layout, external references and connection details with underlying business modules. The designer can perform further refinements (editing the XML files or applying patterns from the pattern catalogue) on this default specification to get the desired final interface appearance and behavior.

The remaining of the chapter is structured as follows: Section 2 gives a brief overview of the OO-H models and constructs that provide the needed hypermedia semantics and describes in detail, by means of a comprehensive example, the modeling process. Section 3 presents a discussion about related work in the field. Last, conclusions and further work are sketched in Section 4.

THE OBJECT-ORIENTED HYPERMEDIA METHOD

A Brief Introduction

The OO-H (Object-Oriented Hypermedia) method is a generic model, based on the object-oriented paradigm, that provides the designer with the semantics and notation necessary for the development of Web-based interfaces and its connection with previously existing application logic modules.

OO-H defines a set of diagrams, techniques and tools that shape a sound approach to the modeling of Web interfaces. The OO-H proposal includes:

- Design process
- Pattern catalog
- Navigation access diagram (NAD)
- Two-fold presentation layer (abstract presentation diagram and composite layout diagram)
- CASE tool that supports and automates to a certain extent the development process

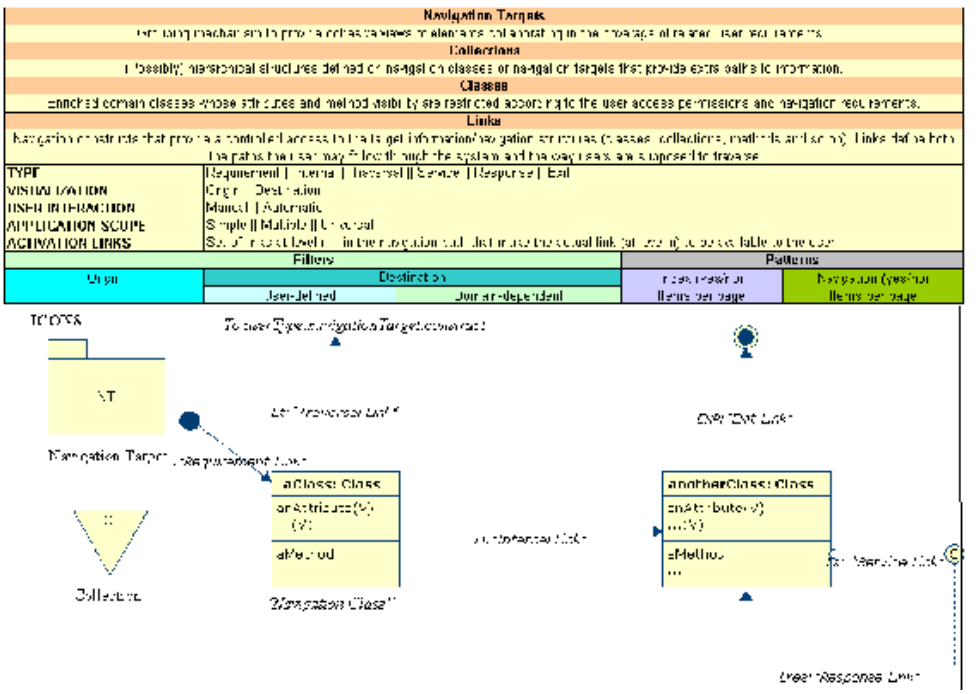
The extension to "traditional software" production environments is achieved by means of two complementary views: (1) the navigational access diagram (NAD) that defines a navigation view, and (2) the abstract presentation diagram (APD) and composite layout diagram (CLD) that gather the concepts

related to abstract structure of the site and specific presentation details, respectively. The NAD diagram enriches the domain view provided by the UML use case and class diagrams (UML Specification. V1.3., 1999) with navigation and interaction features. Also, to define navigation and visualization constraints, OO-H uses the object constraint language (Warmer & Kleppe, 1998), a subset of the standard UML that allows software developers to write constraints over object models augmenting the model precision. OO-H associates such constraints to the navigation model by means of filters defined upon links. On the other hand, the definition of abstract pages in the APD is based on a set of XML DTDs. 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 that is integrated in the OO-H proposal.

The Navigation Access Diagram

The navigation model is captured by means of one or more NADs. The designer should construct as many NADs as different views of the system are required and should provide at least a different NAD for each identified user profile. This diagram is based on four types of constructs: (1) navigation classes, (2) navigation targets, (3) navigation links and (4) collections. Also, when defining the navigation structure, the designer must take into account

Table 1



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. These features are captured by means of different kinds of navigation patterns and filters associated with links and collections. Table 1 shows an overview of the main NAD constructs.

- **Navigation classes** (NC): These are enriched domain classes with attribute and method visibility that has 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, only displayed when an exhaustive system population view is required, e.g., for code refinement reasons).
- **Navigation targets** (NT): They group the elements of the model that collaborate in the coverage of each user navigation requirement.
- **Navigation links** (NL): They define the navigation paths the user is able to follow through the system. They may have a navigation pattern and a set of navigation filters associated, which together provide the required additional information to construct the user navigation model. OO-H defines six link types:
  - **I-links** (internal links) define the navigation path inside the boundaries of a given NT.
  - **T-links** (traversal links) are defined between navigation classes belonging to different NT.
  - **R-links** (requirement links) point at the starting navigation point inside each NT.
  - **X-links** (exit links) point at places outside the boundaries of the application. They are also used as an auxiliary mechanism to represent the feeding of parameters to methods.
  - **S-links** (service links) and their corresponding **R-links** (response links) show the services available to the user type associated with that NAD and the view the user accesses when the interface recovers the control of the application. Service links also gather the way the user is required to introduce the parameters needed for the invocation of any method. Regarding such parameter introduction, OO-H defines five possibilities: (1) *hidden* and (2) *constant* parameters imply no user introduction of values. By default, the introduction mode is set to (3) *immediate*, which means that the interface shows a text field where the user must type the required value. When the user is allowed to choose among a predefined set of possibilities, the

introduction mode is set to (4) *selection*. Last, when the parameter selection requires navigation, the (5) *navigation* mode (with a start navigation link and an end navigation link, chosen among those defined in any NAD) is established for that parameter.

- **Collections**: They are (possibly) hierarchical structures defined on navigation classes or navigation targets. They provide the user with new ways of accessing the information. The most common type of collection, and the one we will use along this paper, is the C-collection (classifier collection) that acts as an abstraction mechanism for the menu concept.

Regarding **navigation filters**, we have already mentioned that they are captured in OO-H by means of OCL expressions. We can distinguish between filters applied to objects in origin (Fo) and filters applied to target population (Fd). Fo's capture origin navigation constraints, that is, conditions applied to the origin object (or set of objects) that prevent the user from following the navigation if not fulfilled. The main difference between Fo's and Fd's is that, while Fo's inhibit navigation (the appearance of links in the interface), Fd's restrict the target population being visualized, but do not refrain the link from being shown in the interface.

On the other hand, **navigation patterns** are characterized by two properties: indexing (yes/no and, if yes, number of items per page, in order to allow index pagination) and navigation (yes/no and, if yes, number of items per page, in order to diminish guided tours size).

OO-H defines other navigation-related metamodel attributes associated to links as follows:

- **Visualization** (show in origin/show in destination): any link implicitly connects an origin (implicit or explicit) and a target information set. When the visualization attribute is set to origin, the target information set is visualized together with the origin, that is, in the same abstract page. However, when it is set to destination, a new abstract page is generated, and a navigation action (such as clicking on an anchor) is required to follow the navigation path.
- **User interaction** (manual/automatic): Sometimes it is useful for the user not to be obliged to click on a link in order to get the target information set. This characteristic is captured in the user interaction metamodel attribute, which in this case, will be set to *automatic* as opposed to the traditional (default) *manual* mode.[2]
- **Application scope** (simple/multiple/universal): This concept stands for the number of objects a given link involves in origin when it is traversed. The origin of a given link can be defined to be a single object (simple), a

set of objects chosen by the user (multiple) or the set of objects present in the actual view (universal).

Last but not least, OO-H introduces the concept of **activation link**. Several times the information the user needs to access slightly varies depending on the contextual navigation (where the user comes from). OO-H abstracts such a situation by means of an activation-link mechanism. Each link defines its set of activation links, that is, links that, when coming through them, make the actual link available for further navigation.

All of these concepts[3] will be used later in this chapter, when we present the navigation access diagrams corresponding to the case of study.

In the rest of this chapter, we illustrate OO-H by means of an example: a conference paper review system, that is, a Web-based application that helps conference program committees manage the process of receiving and evaluating conference papers.[4]

The Case of Study: A Conference Review System

The purpose of the system is to support the submission, evaluation and selection process for papers sent to a given conference. We can identify the following actors interacting with the application:

- PC Chair — Responsible for creating and managing the conference
- PC Member — Responsible for evaluating a set of papers assigned
- Reviewer — Responsible for reviewing a paper
- Author — Responsible for submitting a paper for acceptance at the conference

The following functions (processes) must be supported by the system:

- Paper submission (any registered author may submit a paper)
- Assignment of papers to PC Members
- Assignment of papers to Reviewers (a PC Member may re-assign a paper to a Reviewer)
- Input of reviews
- Acceptance or rejection of papers

The OO-H method tackles the development of Web applications following a user-oriented approach. The process starts by defining a use case diagram and a business class diagram (both are part of UML). These two models provide the necessary input to accurately model the navigation layer of the application. The construction process also includes a first level of personalization: different user profiles may have associated different navigation diagrams.

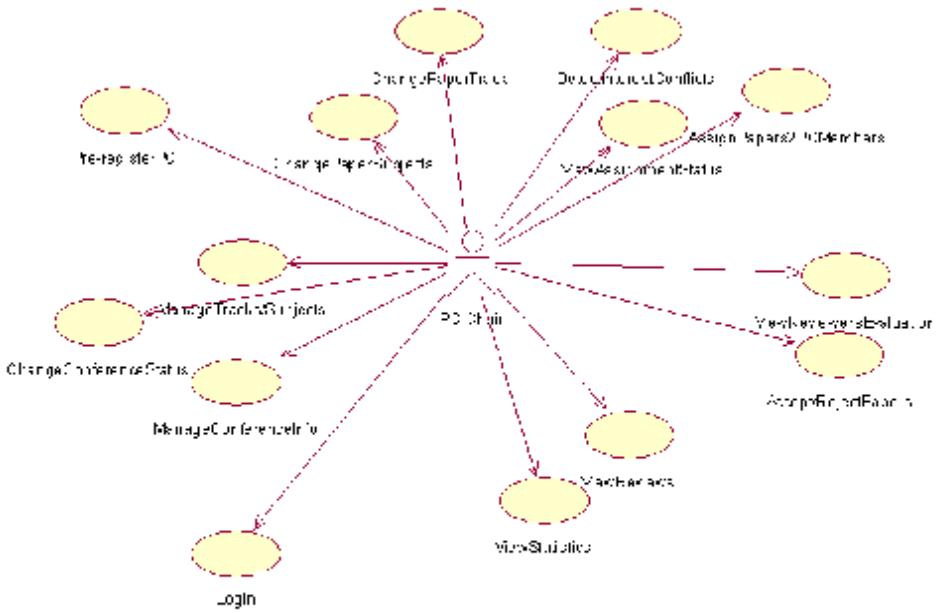
Use Case Diagrams

The use case diagram is one of the key mechanisms of UML. It captures the system functional requirements for each user type (actor) and drives the remaining phases of the software construction process. OO-H uses it as a basis on which the navigation requirements are structured.

To illustrate this step, we will model the use case diagram corresponding to the PC Chair (see Figure 1). As stated in the description of the system, "*a PC Chair is responsible for creating the conference, determining the conference tracks and subjects, establishing the Program Committee and, advised by the PC Members, defining the final list of accepted and rejected papers. The conference is supposed to have a set of tracks and, optionally, a set of subjects. The PC Chair also defines the conference deadlines: submission, review and notification*." At this stage, the association of a storyboard (mockup of the interface) to the different use cases is advisable to better illustrate user–system interaction. It is also important to note that the use cases represented in OO-H are business use cases, and so they may (and usually will) have different implementations depending on the target technology, architecture and target platform.

Together with the use case diagram, OO-H needs a UML business class diagram. Both diagrams provide the input to design the NAD diagrams that reflect the navigation paths through the system.

Figure 1: UC PCChair

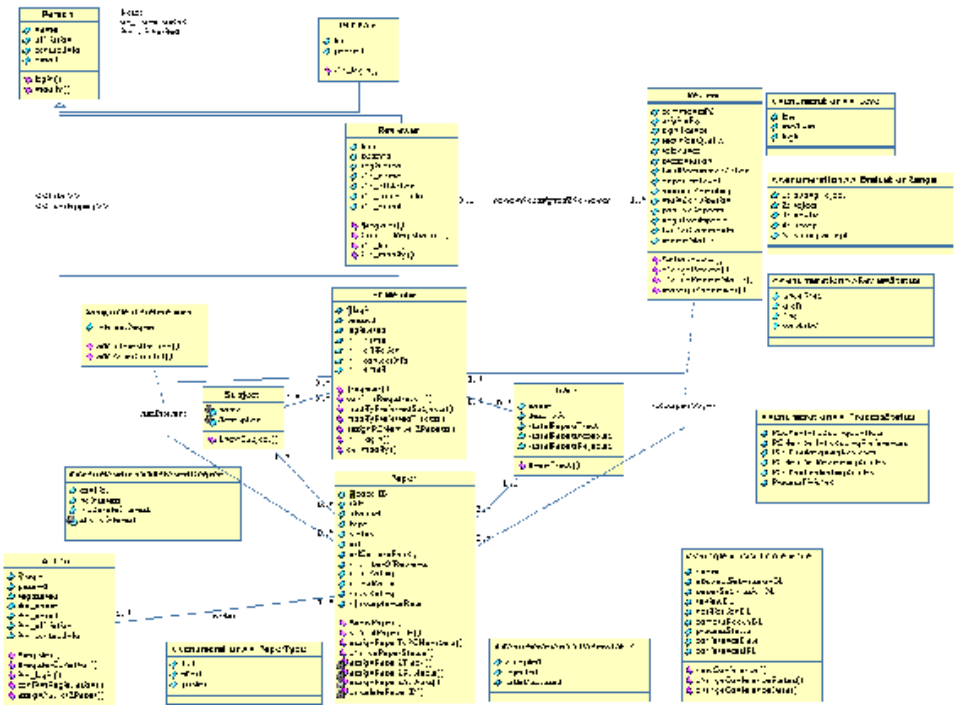


Business Class Diagram

The class diagram modeled for the conference review system can be seen in Figure 2. Just to clarify, note that a slash (/) next to an attribute/method stands for *derived*. A dollar symbol (*$*) next to an attribute/method name stands for *class-scope*[5] attribute/method. Also, the <<*enumeration*>> stereotype defined on classes is the mechanism UML uses to define enumerated types. The attributes of those stereotyped classes represent the possible values of the enumerated type.[6]

Our class diagram depicts the following domain concepts (business classes): users (categorized into Authors, Reviewers, PC Members and PC Chairs), articles, revisions, tracks and subjects. Moreover, we detected two association classes: reviews (evaluation and comments a PC Member introduces in the system regarding a given paper) and revision preferences (interest degree and interest conflicts a PC Member has with reference to a given paper). Some attributes and methods have been directly derived from the description of the system. Others have been inferred from the application domain. The responsibility assignment has been realized taking into account which class was responsible for the most of the data involved in each one of the methods.

Figure 2: CRS class diagram



Although we tried to avoid method duplication in order to simplify the diagram, sometimes (mainly when we are dealing with methods that involve creation or deletion of relationships between objects) it might be convenient to provide the user with access modes from each of the classes involved.

Based on the system description, we identified six stages for the conference revision process, all controlled by the PC Chair. Each stage implicitly determines different function sets available to each profile.

- *AuthorSendingPapers*: In order to begin the revision process, the PC Chair must introduce the conference parameters (PC Chair data, conference dates, URLs, tracks, subjects, PC Members involved in the revision process, and so on) and, as the last step, open the period for the authors to submit papers. The system transition to this status might imply the sending of a *Call for Papers* to a set of selected distribution lists (DBWORLD, ISWORLD, etc).
- *PCChairIntroducingConflicts*: Once the paper submission period has expired (*conference.paperSubmissionDL*), the PC Chair changes the status of the system. This status change grants the access of the different profiles to a new set of tasks. For example, in this new state, the PC Chair will be able to revise the submitted papers, change their track and subjects, if necessary, or look for revision conflicts (e.g., PC Members that are authors of a submitted paper).
- *PCMemberIntroducingPreferences*: The following step is to open the system for the PC Members to introduce their preferences regarding the submitted papers, as well as revision conflicts not detected by the PC Chair (if any).
- *PCChairAssigningReviews*: Once the PC Chair closes the period to register paper preferences, and taking into account the preferences and conflicts registered in the system, it is time for the PC Chair to assign papers to the different PC Members for revision.
- *PCMemberReviewingPapers*: Then, the period for each PC Member to introduce its revisions is opened. Again, the system transition to this state implies the sending of an e-mail to each PC Member with information regarding the papers the member has been assigned and the period of time available to perform the revision. This phase ends when the review deadline (*conference.reviewDL*) is reached.
- *PCChairEvaluatingPapers*: Once the PC Chair sets the state of the system to this value, and taking into account the reviews introduced by the PC Members (or the corresponding external reviewers), the PC Chair must decide which papers are accepted and which ones are rejected.

- *ProcessFinished*: This last step implies the sending of an e-mail to all paper authors, informing them of the revision process result regarding their papers.

OO-H considers that the e-mails the system must send on some conference state transitions are isolated inside the body of the c*onference.changeProcessStatus* method and, therefore, out of the scope of our models.

Modeling Assumptions

In order to simplify the diagrams, we applied the <<*singleton*>> pattern (Cachero et al., 2001) to the *Conference* class. This pattern implies that the generated system deals with a single conference (i.e., there may exist just one object of type *conference)*. All papers, tracks, subjects and people in the database are implicitly related to that conference. Extending such a system to deal with several conferences at a time is trivial.

Once the user requirements have been clearly identified and the business domain modeled, we can go one step further and focus on navigation paths through the information space. The class diagram helps decide which paths are semantically relevant, while the user-case diagram structures such paths according to the user expectations of the system.

Navigation Modeling

In OO-H, the use-case diagram and the storyboard help the designer to decompose the system interface into subsystems and pages through which the user can navigate in a meaningful way, taking into account the user need for fulfillment of a set of functional and navigation requirements.

The construction process of the navigation access diagrams (NAD) is divided into the following steps:

(1) Grouping process on the use-case diagram: OO-H packages the use cases attending to a set of criteria.[7] For the sake of simplicity, we depicted these grouping decisions by means of transparent package symbols around the use cases involved, although the actual notation (used in the OO-H case) is UML-compliant. Each package links the underlying use cases to a single navigation target, where their inner navigation paths are modeled.
(2) Automatic derivation of the top level of the NAD diagram
(3) Construction of the different NAD diagrams, driving the navigation decisions by the corresponding storyboard

Next, we illustrate these steps by means of an example: the NAD diagram construct process for the PC Chair profile.

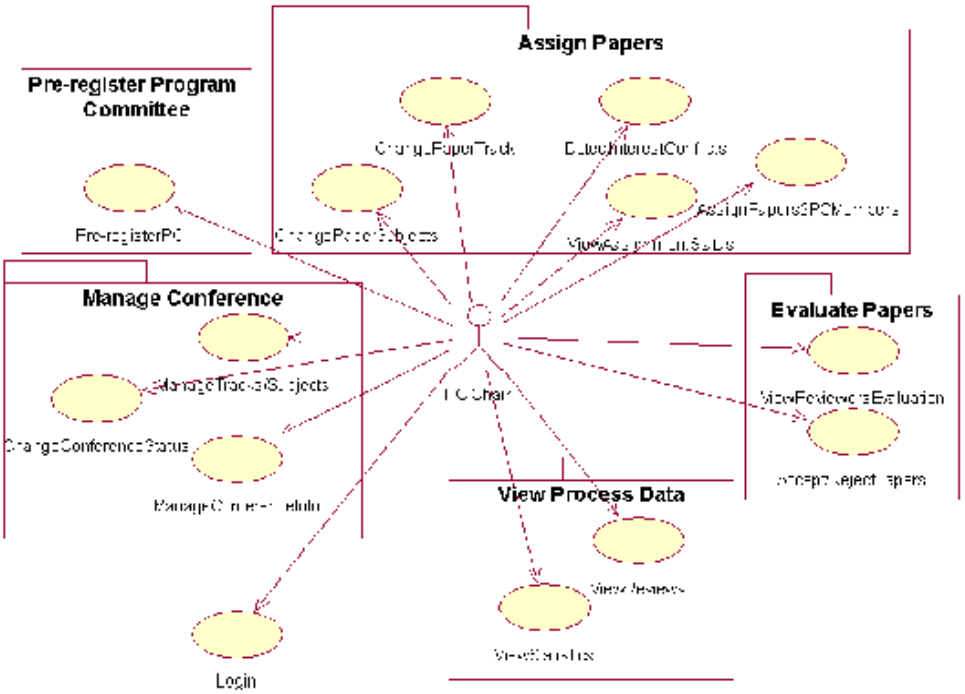
PC Chair Navigation Profile

In this section, we will present, step by step, the construction process of the NAD diagram corresponding to the PC Chair. The process followed to construct the diagrams corresponding to the other actors (PC Members, Reviewers and Authors) is analogous.

If we look again at the PC Chair use case diagram (see Figure 1), we will observe the set of functional requirements the interface must fulfill.

In Section 2.4, we commented how this diagram was the base on which to show decisions regarding how to group those requirements into navigation targets (NT), attending at semantic, functional dependency and data criteria. We say we are using semantic criteria when we group use cases that have a similar aim. As an example, in Figure 3, we can observe how the use-cases *view Reviews* and *view Statistics* have been grouped under the NT *View Process Data*, due to the fact that both provide reports on the review information (one aggregated, the other one detailed) contained in the system.

Figure 3: UC grouping process



On the other hand, in order to gather *view Reviewers Evaluation* and *Accept/ Reject Papers*, we applied what we call a functional dependency criterion, that is, we departed from the premise that in order to be able to accept and reject papers, we must have a synthesized view of every reviewer evaluation in order to help the PC Chair to make a sound decision. Last, the use cases *ChangeConferenceStatus* and *ManageConferenceInfo* have been grouped under the NT *Manage Conference*, following a data criterion, that is, due to the fact that both access and manipulate data of the *Conference* class.

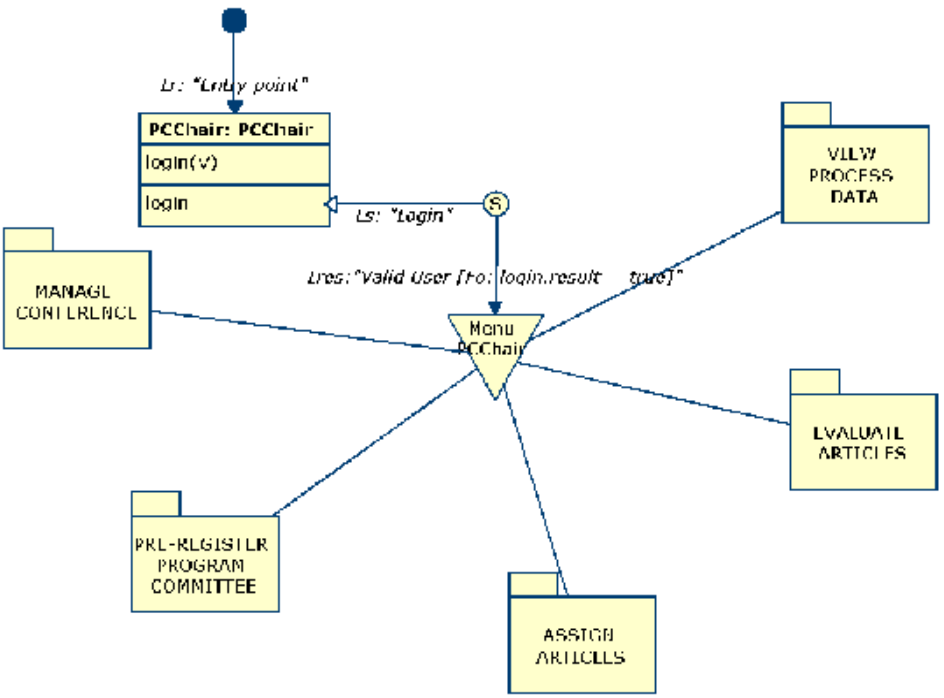
The grouping process entails an interface structure. Consequently, the more careful this process is performed, the higher the quality of the final interface structure will be.

Once this process is finished, the next step is to dive into each grouping and define the inner navigation paths. In order to illustrate the navigation design decisions, we are showing the *storyboard* corresponding to the first NT, *Manage Conference.*

PC Chair Profile Entry Point

In Figure 4, we observe the modeling of the entry point to the application (represented by the requirement link *Entry Point*). One possible set of

Figure 4: PCChair NAD Level 0



storyboard pages corresponding to this diagram is showed in Figures 5 and 6. The first abstract page corresponds to a form for the user to log in to the system. This process involves a user login, password and profile, which correspond to

Figure 5: Login

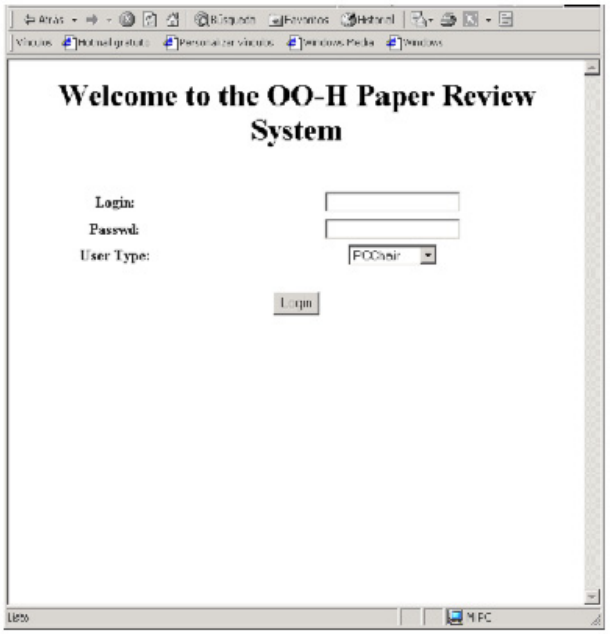
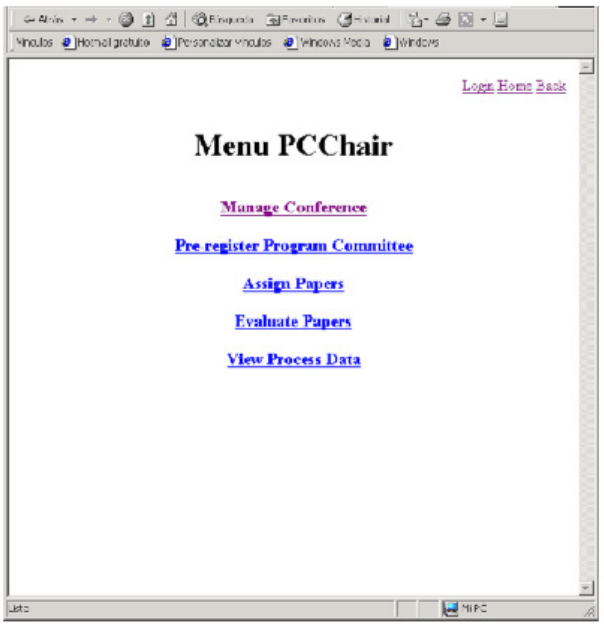


Figure 6: Menu PCChair



the parameters (all mandatory) of the method *PCChair.login*. If the user exists (condition that is reflected in the *Fo Valid User*), the user will be shown a menu, where a link to each of the five NT identified (see Figure 7) is presented. Such a menu is represented by the collection construct *Menu PCChair*.

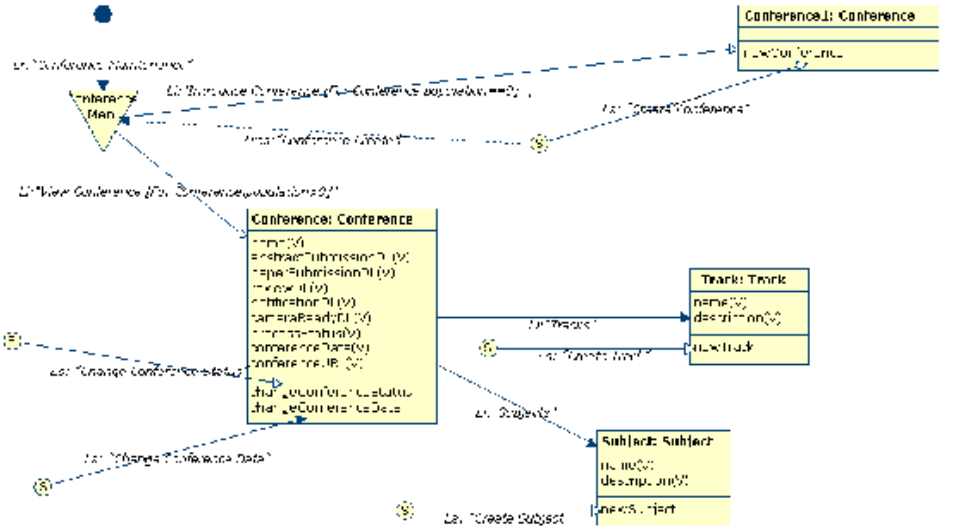
Each NT can be further "exploded" to show detailed internal navigation. To illustrate this, next we will show the internal structure of the NT *Manage Conference*.

Navigation Target "Manage Conference"

When the PC Chair selects the *Manage Conference* option,[8] the interface subsystem modeled inside the corresponding NT is accessed. The entry point to this NT (requirement link *Conference Maintenance*, see Figure 7) points at a new collection, called *Conference Menu*, that differs from the previous one in the type of links departing from it; while in the *PCChair Menu* collection, the links had the visualization attribute set to *show in destination,* this time they are of type *show in origin* (hollow arrowheads of *Introduce Conference* and *View Conference* links). This represents the fact that, in this case, no link to a new page must be generated, but that the information corresponding to the destination classes must be directly presented to the user, provided that the corresponding Fo is evaluated to be true.

Furthermore, as both filters (*conference.population)==0* / *conference.population > 0*) are disjoint, only one view will be available at a time: if the conference has not yet been created (it is the first time the PC Chair enters the system), the screen corresponding to the *createConference* method will appear (see Figure 8). This page gathers the set of parameters the method *New Conference* requires. Once the method has been invoked and the control returned to the interface, the response link *Conference Created* drives the user again to the *Conference Menu*. This time, however, when the filters are checked again, it is the *View Conference* link that is evaluated to be true, and so the system will automatically generate the page shown in Figure 9. This page provides a view of the conference data, together with three buttons (*Tracks*, *Subjects* and *Change Conference Data*), corresponding to the three links set to *show in destination* that depart from the *Conference* navigation class. Activating each one of these links, we will navigate to the views *Change Conference Data* (see Figure 10), *Tracks* (see Figure 11) and *Subjects* (see Figure 12), respectively. Also, there is another button (*Change Conference Status*) that actually executes the homonym service, with the value of its input parameter (*newStatus*) set to the string selected in the selection list shown in Figure 9.

Figure 7: Manage conference



This selection list is the result of associating a selection introduction mode to the change service status parameter.[9] At this point, the reader may wonder how OO-H generates the mediator for the interconnection with preexistent logic modules. In fact, the answer is in the class diagram. We already noted how the class diagram can show the interface to underlying business libraries. In order to identify those libraries, OO-H defines the <<legacy>> stereotype that, associated to domain classes, reflects the fact that those classes are part of a preexistent library. Also, services in those <<legacy>> classes have their input/output parameters defined at domain level. At navigation level (in the NAD diagram), we thus just have to center on the way the user is going to introduce those parameters, and whether introduction is mandatory or not: only mandatory attributes that have no default value in the class diagram or in the navigation diagram appear as mandatory in the interface. Before the generation process is executed, the CASE tool detects <<legacy>> classes and requires the designer to enter the necessary solution-space data (physical location of the library, desired protocol to actually connect to the library interface, etc.) to generate the mediator. Note how this fact obliges the <<legacy>> classes to capture the exact interface to the desired library. All this information is kept in an XML template structure (Tlogic: see section "View Refinement"), which is later used by the tool to perform the generation process.

In Figure 10, we can also observe how the different method parameters may have a default value associated (in this case, the actual value of the corresponding class attributes). Also note that, when not otherwise stated, a default response link is supposed to go back to the view from which the service was invoked.

Figure 8: New conference

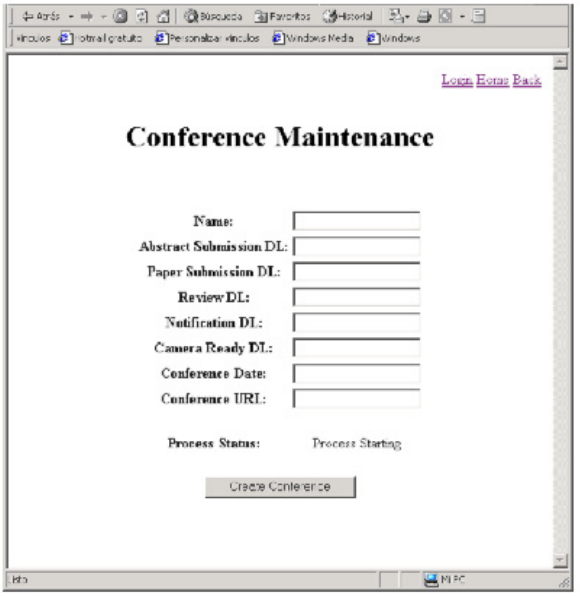
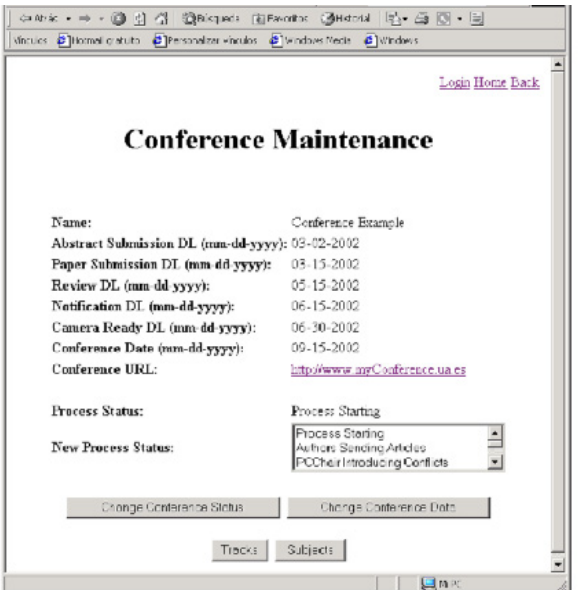


Figure 9: Conference view/change status



As the reader will have already inferred, it is the visualization attribute (explained in Section 2.2) that characterizes the final abstract page structure of the interface. We call this page structure abstract, because there is nothing that prevents those pages from being further composed into a frame structure or any other mechanism that allows the coexistence of different views of the system on the same physical screen.

Figure 10: Change conference data

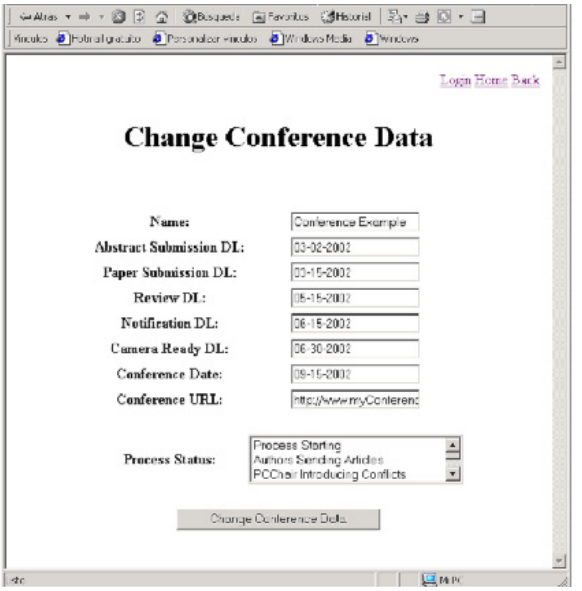


Figure 11: New track

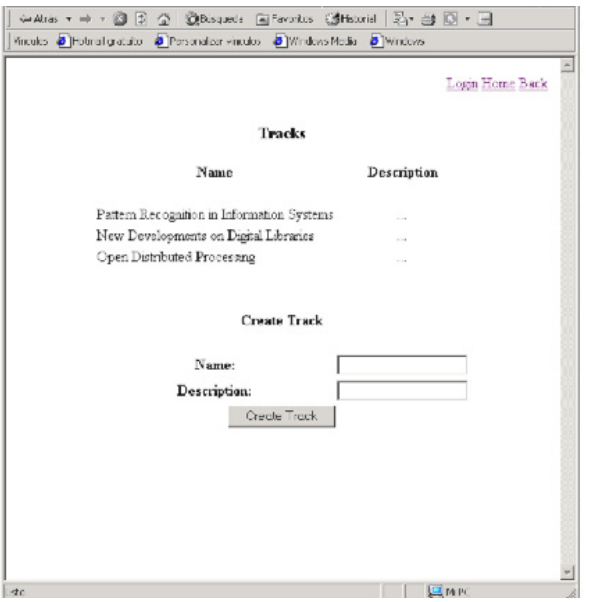
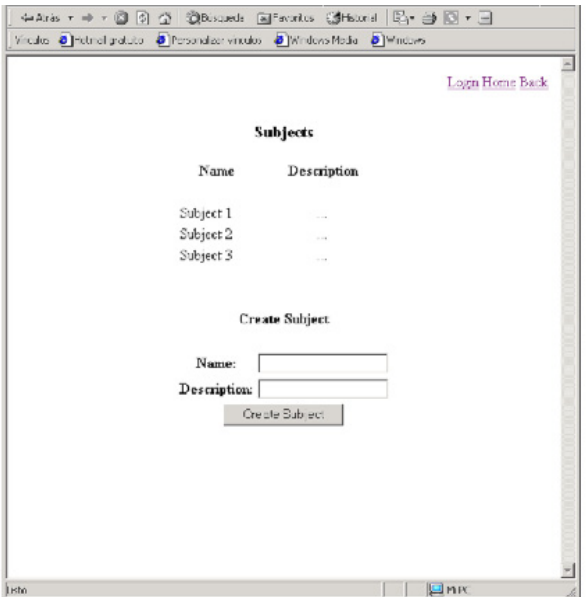
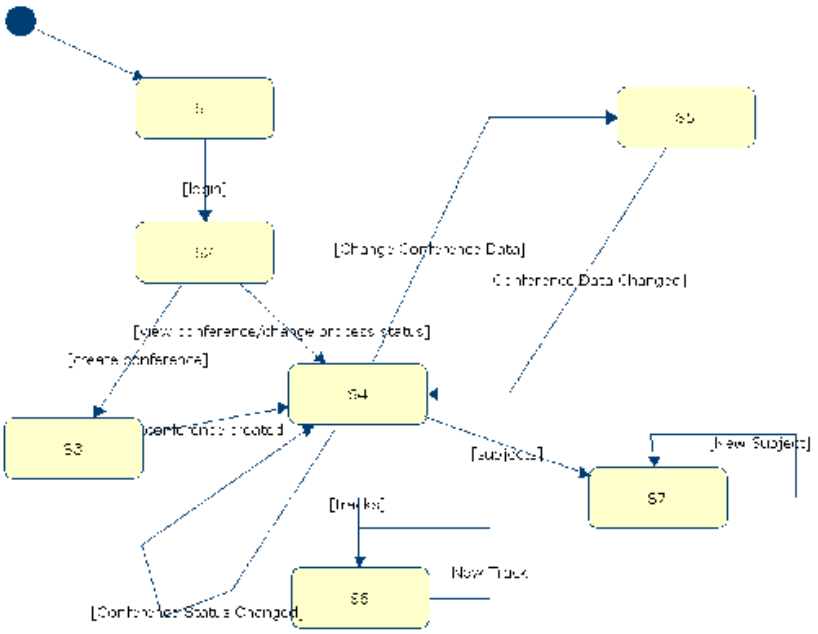


Figure 12: New subject



In Figure 13, we can observe the interconnection of the screenshots captured in Figures 5 to 12. In the next section, we will explain how this site view interconnection perfectly matches with the APD diagram, automatically

Figure 13: Site view



generated by the OO-H CASE tool departing from the corresponding NAD diagram. Also, note how the storyboard presents a link from every abstract page to the NT origin, another one to the application entry point and another one pointing at the previous abstract page in the navigation path. OO-H automatically generates those links (from each page to the home page, to the NT root page and to the previous page in the navigation path) if not otherwise stated.

Although at this stage of the process we already have all the information needed to automatically generate a functional prototype, OO-H recognizes the need for a greater level of interface sophistication than that provided by the information gathered at NAD level, regarding appearance and usability features. The abstract presentation diagram (APD) and the composite layout diagram (CLD) provide a set of mechanisms to refine the interface at a lower level of abstraction.

View Refinement: Presentation Modeling

A default APD, reflecting the abstract page structure of the interface, can be automatically derived from the NAD diagram. This default APD gives a functional but rather simple interface (with default location and styles for each information or interaction item, and only simple patterns applied), which will probably need further refinements in order to become useful for its inclusion in the final application. It can, however, serve as a prototype on which to validate that the user requirements have been correctly captured. Furthermore, it separates the different features that contribute to the final interface appearance and behavior by using a page taxonomy, based on the concept of templates and expressed as XML documents, which are, namely:

(1) Tstruct — Used to capture the information that needs to be shown
(2) Tform — Used when the page, apart from information, includes calls to underlying logic
(3) Tlink — Captures the interconnection and dependencies among pages
(4) Tfunction — Gathers client functionality used in the different pages
(5) Texternal — Used to gather type, location and behavior of external elements (such as images, applets, etc.) that may refine the initial interface
(6) Tlayout — Where the location of elements and the definition of simultaneous views and synchronization is captured
(7) Tstyle — Where OO-H maintains features such as typography or colour palette for each element of the interface
(8) Twidget — Where implementation constructs are related to the different information and interaction items depending on the final implementation platform and language

(9) Tlogic — Where the system keeps implementation details regarding interaction with underlying business logic (kind of service, parameters, connection protocol, etc.)

The refinement process consists thus on the modification of the default APD structure. This process is greatly simplified with the application of a series of APD-related patterns captured in the OO-H catalog (Conallen, 1999). Furthermore, the catalog provides an executable python routine (transformation rule in OO-H terminology) for each materialization of the patterns. This routine, when loaded in the OO-H CASE tool and executed, changes the contents of the required APD abstract pages. Also, these routines may cause any new page, link or dependency to appear, disappear, or be modified on the diagram. OO-H provides yet another way to manipulate some of the abstract pages (namely, Texternal, Tlayout, Tstyle, Twidget), by means of the composite layout diagram (CLD). In this view, the location and visual style of elements can be edited, widgets (implementation constructs) can be specified and new elements can be added to improve the visual impact of the generated interface.

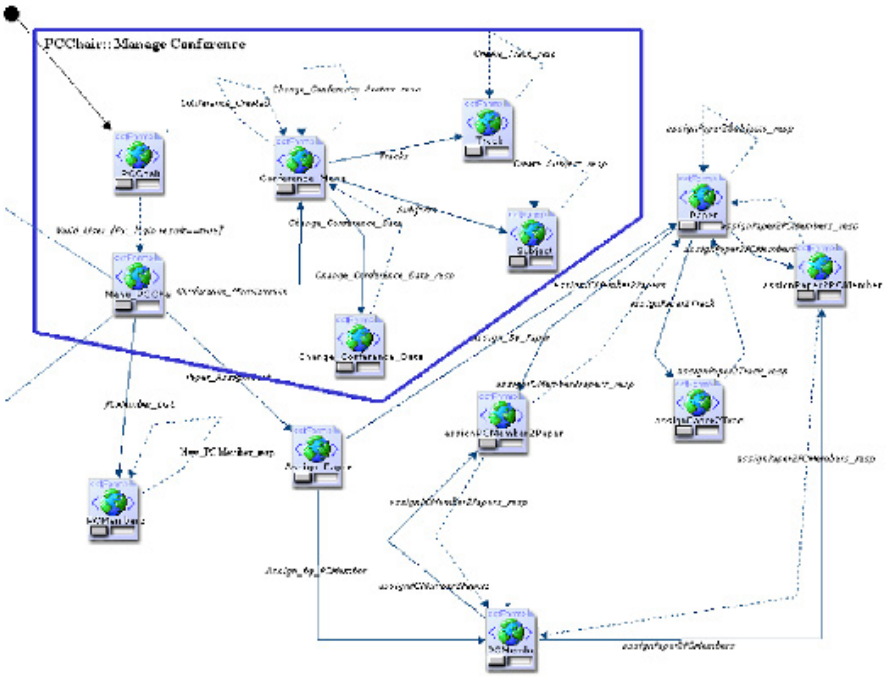
The last step of the process, once the abstract pages (XML documents) have been refined, is to feed the system description (those XML documents) to a model compiler (not discussed here) that has the target-environment knowledge that permits the generation of an operational Web interface. It is important to note that the code-generator receives as its only input that XML-based system description, which therefore contains all the information kept in the different diagrams. Changes in such diagrams cause the XML specification to be regenerated, which assures that no inconsistencies arise between both representations of the system.

In the next section, we will show part of the default APD generated from the PC Chair NAD diagram.

PC Chair Default Interface Site View

The OO-H CASE tool includes an algorithm to generate, departing from the NAD diagram, the set of pages that makes up the site view of the interface (the detailed algorithm can be found in Gómez, Cachero, & Pastor, 2001) . Note how only Tstruct, Tform and Tlink pages are relevant to the site view of the system, and so how the APD diagram graphically shows them. OO-H includes a default Tfunction (client-side logic to control user inputs for method parameters), a default Tlocation and a default Tstyle. Furthermore, a default Twidget is applied to the different interface elements depending on the target implementation language.[10]. Also note how the contents of the Tlink global page is shown by means of links connecting the Tstruct and Tform pages. The

Figure 14: APD PCChair Profile



construction of Tlogic, based on the definition of interfaces, is out of the scope of this chapter.

If we look back to Figure 13 (storyboard snapshot interconnection diagram) and compare it with Figure 14 (APD partial view), we can see how they match: the only difference is that snapshots S2 and S3 (Figures 5 and 6) are now gathered in a single APD page (*Conference_Menu*) due to the fact that, in Figure 7 (NAD diagram), we defined that both views were shown in origin, and so embedded in the same abstract page, represented by the collection *Conference Menu*. The remaining snapshots can be directly associated with exactly one abstract page at the APD level.

DISCUSSION[11]

The goal when developing OO-H has never been to provide "yet another method" but to take advantage of proven successful ideas of existing methodologies and practices in related fields, from which we borrowed concepts such as use cases, patterns, collections or links, just to name a few. The following step has been to integrate them into a sound proposal, while contributing some

new ideas to improve identified gaps. Therefore, OO-H shares many concepts with other methods for hypermedia and Web design, many regarding the navigation model. In fact, these similarities also exist among the most relevant methods and methodologies being actually supported, among which we could cite UWE, WSDM, ADM-2, WebML, OO-H, OOHDM or HDM2000. Some of these similarities are as follows:

- All proposals identify the necessity of a brand new model: the navigation model, to tackle the Web application development process. Inside this diagram, all proposals distinguish between information and access structures.
- All proposals clearly separate content, navigation and presentation space by means of different models.
- There is a general agreement in the necessity of any kind of navigation diagram, for which standard methodologies do not provide the necessary constructs. So, such standard methodologies have been extended or a broad new notation and model have been proposed.
- All proposals aim at defining a precise and systematic, even in some steps automated, process for the development of the Web application.
- All proposals present an exhaustive authoring process (a method and a notation).
- All proposals avow the necessity of any kind of constraint language to augment the precision of the Web application models.
- All proposals avow the necessity of a CASE tool that supports the whole process.

Far from being a drawback, we think this sharing of concepts is one of the reasons for the rapid development of Web engineering. The IIWOST'01 experience demonstrates, from our point of view, that hypermedia design methods are reaching a state of maturity in which a set of premises is being settled. This makes it relatively straightforward to understand the concepts behind other proposals and to reflect together on missing semantics and accurate definitions of concepts and constructs.

However, there are many differences among the different proposals. Some approaches (e.g., UWE) aim at covering the whole life-cycle of the Web application, while others (e.g., OO-H) center on the authoring process. Also, the data-orientation of some approaches (e.g., ADM-2, WebML) contrasts with the user-orientation of many others (e.g., OOHDM, OO-H). Only some approaches formalise the sketching and storyboarding techniques (e.g., OOHDM) widely used by user interface designers.

The design activity also greatly differs in the level of detail. While some approaches prefer to use external tools for refining interfaces, or even connect them with their models (e.g., WebML), others prefer the integration of layout on the same tool, thus providing an integrated development environment, even at the cost of less elaborated visual interfaces (e.g., OO-H).

Another important aspect regards the extent to which different proposals abstract Web application functionality. Most methods studied so far provide traditional database functionality (insert, update, delete). However, the kind of services required by actual Web applications is far more complex, and the degree to which those services are integrated in the proposals greatly differs. Although database models could be extended (and in fact are being extended, see e.g., WebML) to deal with such services, we claim OO models, in which services are first-class concepts, provide a more flexible platform with which to model Web functionality. Specifically, the class diagram proposed in such methods is prepared to include the interfaces needed to connect with underlying business logic modules, thus facilitating the functionality integration process. However, and although traditional software engineering methodologies provide the designer with the notation and models already proven successful for the development of business services and transactions (which is the reason why OO-H does not provide any new diagram or construct for this part of the Web application), they were not designed to deal with concepts such as Internet transactions (set of services that may be executed on different independent databases, with no knowledge of each other), client functionality, Web services, etc. OO-H is currently working on the abstraction of those concepts, which are part of the research trends in the field.

Also, personalization features, which should be present at every stage of development, are diversely covered in the different approaches. In this sense, OO-H provides a basic level of personalization, based on departing from user profiles and designing a different model for each user type. More evolved proposals (e.g., WebML) provide frameworks for one-to-one personalization and even support for interface proactive behavior. Although this topic would need a whole chapter on its own, we would like to note that OO-H regards personalization as a core activity for Web engineering. This task, together with navigation design, makes, from our point of view, one of the greatest differences between traditional software development processes (including that of distributed applications) and the Web development process.

Moreover, the degree to which those proposals are supported by a CASE tool greatly differs: while some approaches have a fully operational CASE, with generation code capabilities, others have support just for the notation and evolution of the different models, while still others have their associated tool

acting as a mere diagram editor. In this sense, OO-H already has a CASE capable of generating fully operative interfaces, and we are currently working on the generation of modules that implement the connection with business logic. Furthermore, this tool has powerful extension capabilities due to the belief that Web concepts are likely to greatly evolve in the next years. Recent studies show the low impact of Web methodologies in current enterprise practices. We think the existence of fully operative, easy-to-use CASE tools supporting such methodologies is a must if the research community wants to change this trend.

Finally, and despite the great number of shared concepts, there are great controversies regarding navigation, especially the level of abstraction at which it should be defined. From OO-H, we claim that although a design navigation model (gathering the abstract site view of the interface) is needed, and even more if we aim at providing any kind of automatic code generation support, there is, however, also a need for a device independent navigation model, which is, as far as we know, not provided by any methodology. Until now, all models (including ours) have had in mind the concept of abstract page when dealing with navigation constructs, probably due to our background as Web developers. However, from OO-H, we think it would be advisable to preserve a certain "flavor" among implementations of the same application for different devices. That is the reason why we are currently working on a navigation view with which the concept of node is based on the user requirements, rather than on the class diagram representing the domain information structure, which is the basis adopted so far in other proposals.

CONCLUSIONS AND OO-H RESEARCH DIRECTIONS

It is commonly avowed that current interface-related languages are too target-environment dependent and, therefore, lack the flexibility needed for the development of complex Web applications. Well-defined software development processes are necessary in order for the community of software engineers to design Web-based applications in a systematic way and thus avoid the risks of failure involved in ad hoc development processes. Our purpose has been to address these problems in the Web engineering context by means of a conceptual modeling approach that has been proven successful for software production from conceptual models.

In order to properly capture the particulars associated with the design of Web interfaces, OO-H method adds several navigation and interface constructs that define the semantics suitable for capturing the specific functionality

of Web application interfaces. Two new kinds of diagrams, the navigation access diagram and the abstract presentation diagram, have been introduced. Both the NAD and the APD capture relevant information for the interface design by means of a set of patterns, which are defined in an interface pattern catalog. The NAD, which is based on an OO class diagram, is centered on information and navigation user requirements and provides each user-type with a different view of the system. Each piece of information introduced in the NAD is mapped into a different construct in a default APD. The APD is based on the concept of templates, and its underlying structure is expressed in standard notation. We can perform further refinements on the APD in order to improve the interface visual quality and usability.

From there, a functional interface can be generated in an automated way. In contrast to other existing Web methods, the approach presented in this paper does not intend to be yet another method for Web modeling but rather an extension of any consolidated OO conceptual modeling approach. Summarizing, the most relevant contributions of this chapter are the following:

- The detailed presentation of OO-H method as an interface modeling approach that extends conventional methods starting from navigation user requirements.
- The idea of integration, departing from the conceptual description of the problem, of complex application behavior by means of explicit service interaction modeling. In this sense, OO-H provides the mechanisms to define how each parameter is going to be fed to each service and how the user is going to visualize the service results.
- The use of a pattern catalog to evolve the schemas in order to speed up the development process, improve interface quality and guarantee design experience reuse.
- The notion of transformation rule, associated with each of the possible pattern implementations, as a way to simplify and systematize the modification process of the APD schema.
- The use of a taxonomy of construct templates, which is defined in a standard notation, to build the different constituents (information, layout, input data, client functionality) of the APD as necessary.
- The notion of default APD, which is built by applying a set of mapping rules from the different elements of the NAD.

OO-H is not a closed proposal: the modeling of existing and to-come applications will cause its semantic constructs to evolve and be extended to capture new interface requirements. As an example, we are already working on

the interface-related event modeling. Although Web applications require an event mechanism similar to other traditional applications (e.g., to perform punctual or periodic tasks without need for user interaction, to respond to the finish event of asynchronous services, etc.), we think they also require an interface-related event mechanism that manages which views are available for each user at each moment. In this sense, one possibility is the use of high-level state charts, where each state represents a set of views available for a given user at a given time. A transition condition in this case would imply a change in the set of views the user can access. This last point can be viewed as part of a broader problem, which is that of the integration of workflow activities as part of the authoring process. Also, we think a powerful but yet simple mechanism to specify synchronization of views is necessary in this environment. In this sense, OO-H could benefit from work developed in the multimedia community, where synchronization is a first-order aspect.

Ongoing work in OO-H further includes (1) how to achieve a greater level of personalization (including the definition of a personalization framework), (2) how to detect a greater range of patterns and definition of their corresponding OO-H transformation rules and (3) how to provide support for a broader range of implementation architectures, platforms and languages.

REFERENCES

Bell, R. (1998). Code Generation from Object Models. *Embedded Systems Programming*. 3, 1–9.

Cachero, C. Gómez, J., et al. (2001). Conference Review System: A Case of Study. (2001). In D. Schwabe, (Ed.), Proceedings of the First International Workshop on Web-Oriented Software Technology (195–227). Valencia University of Technology.

Ceri, S., Fraternali, P., et al. (2000). Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. In I. Herman, (Ed.), Proceedings of 9th International WWW Conference. IEEE Press.

Conallen, J. (1999). Modeling Web Application Architectures with UML. *Communications of the ACM, 42*(10), 63–70.

Directory Services Markup Language (DSML). (2001). http://www.dsml.org/.

Gartner Group, Inc. (2001). The Future of Web Services: Dynamic Business Webs. Market Analysis.

Gellersen, H.W. & Gaedke, M. (1999). Object-Oriented Web Application Development. *IEEE Internet Computing, 3*(1), 60–68.

Gómez, J., Cachero, C., & Pastor, O. (2001). Conceptual Modeling of Device-Independent Web Applications. *IEEE Multimedia, 8*(2), 26–39.

Manola, F. (1999). Technologies for a Web Object Model. *IEEE Internet Computing, 3*(1), 38–47.

Martin, D., Birbeck, M., et al. (2000). Professional XML. WROX.

Mecca, G., Merialdo, P., & Atzeni, P. (1999). Araneus in the Era of XML. *IEEE Data Engineering Bulletin*, *42*(10), 63–70.

Murugesan, S., Deshpande, Y., et al. (1999). Web Engineering: A New Discipline for Development of Web-Based Systems. In S. Murugesan (Ed.), Proceedings of First ICSE Workshop on Web Engineering.

Pressman, R.S. (2000). *Software Engineering: A Practitioner's Approach*. Fifth ed. New York: McGraw-Hill.

Schwabe, D. & Almeida, R. (1999). A Method-Based Web Application Development Environment. In D. Schwabe, (Ed.) Proceedings of 8th International WWW Conference. IEEE Press.

Simple Object Access Protocol (SOAP). (2001). http://www.develop.com/soap/.

Semantic Web. (2001). http://www.w3.org/2001/sw.

Universal Description, Discovery and Integration (UDDI). (2001). http://uddi.microsoft.com/.

User Interface Modeling Language (UIML). (2001). http://www.uiml.org.

UML Specification. V1.3. (1999). http://www.rational.com/uml/index.jsp.

Warmer, J. & Kleppe, A. (1998). *The Object Constraint Language. Precise Modeling with UML*. Reading, MA: Addison-Wesley.

Web Services Description Language (WSDL). (2001).http: //msdn.microsoft.com/xml/general/wsdl.asp.

ENDNOTES

1 The special characteristics and requirements of Web applications and the Web development process are beyond the scope of this chapter. Interested readers are referred to Pressman (2000), where a good introduction is provided.

2 Automatic links tend to be visualized in origin, while manual links tend to be visualized in destination. However, there are cases in which it can be useful to combine both characteristics in a different way.

3 More detailed information on the semantics of the different constructs and on the OO-H process to specify navigation features can be found in (Gómez, Cachero, & Pastor, 2001).

4 The selected case of study has been proposed by Daniel Schwabe in the context of the first International Workshop on Web Oriented Software Technology (IWOOST'2001).

5 In UML 1.3, the class-scope *$* notation has been deprecated and substituted by an underlined attribute/method name. OO-H will change the notation support accordingly in future versions of the tool.

6 This way of defining enumerated types has been included in UML 1.4

7 These grouping criteria will be further explained in Section 2.8, when we show the PC Chair navigation profile.

8 In the OO-H diagrams, an asterisk next to the link name means that the set of activation links is not complete (that is, is not made up by every link from which the user might have arrived to the actual view). Also, an arrow with a filled head means that its visualization metamodel attribute is set to *show in destination*, while, if it has a hollow head, the corresponding value is *show in origin*.

9 See Section 1 for a description of the different introduction modes for method parameters. More information on services and input parameters can be found at http://www.dlsi.ua.es/~ccachero/reports/services.pdf.

10 The XML specification of every page is available through the tool by clicking on the corresponding page or by means of a menu.

11 An interesting experience regarding modeling methods was held in Valencia in June 2001 during the IIWOST'01. In this workshop, people from some of the most relevant research groups proposed their solution to a common problem (the Conference Review System presented in this chapter). This experience provided a suitable forum, where it was possible to discuss the different approaches from a practical point of view. The different solutions to the case are in the roots of this brief comparison. Readers interested in a deeper discussion or in a description of the different proposals are referred to the congress proceedings.