

Object-Oriented Conceptual Modeling of Web Application Interfaces: the OO- \mathcal{H} Method Abstract Presentation Model

Cristina Cachero^{1*}, Jaime Gómez¹, and Oscar Pastor²

¹ Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante. SPAIN
{ccachero, jgomez}@dlsi.ua.es

² Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia. SPAIN
opastor@dsic.upv.es

Abstract Object-oriented conceptual modeling approaches must be re-considered in order to address the particulars associated with the design of web application interfaces. In this context, the paper introduces the presentation layer of OO- \mathcal{H} Method, an extension of the OO-Method conceptual modeling approach that is devoted to the specification of this kind of interfaces. The OO- \mathcal{H} Method presentation approach is based on the concept of templates. Each page template may fall into one among a set of categories, which together cover the different presentation perspectives captured in the model. In order to better define the page template structure, a new diagram is introduced: the Abstract Presentation Diagram (APD). The APD does not need to be drawn from scratch: the navigation structure previously defined in the OO- \mathcal{H} Method Navigation Access Diagram (NAD) provides the information needed to automatically generate a default APD. This skeleton template structure may be further refined and enriched with the aid of the OO- \mathcal{H} Method Interface Pattern Catalog. As a result, a web application interface is generated in an automated way.

1 Introduction

The research effort inverted by the scientific community in hypermedia modeling approaches specifically devoted to the development of web sites has led to different projects and products. Some of the most relevant examples studied so far are HDM [7], HDM-lite [6], OOHDM [17], RMM [9], ADM [1, 11] or Strudel [5]. However there is still, as far as we know, a gap to be filled: that of web applications' interaction issues. In this context our research efforts have been focused on the proposal of 'Not Yet Another Method' for web modeling, but on a set of semantics and notation that allows the development of web-based interfaces for existing OO-Method [14, 15] applications. This proposal, known as

* This article has been written with the sponsorship of the Conselleria de Cultura, Educació i Ciència de la Comunitat Valenciana

OO- \mathcal{H} Method [8], extends OO-Method with two new diagrams: (1) the Navigation Access Diagram (NAD) and (2) the Abstract Presentation Diagram (APD).

Both the NAD and the APD can be further enriched and refined by means of a set of interface patterns, which are defined in the OO- \mathcal{H} Method Pattern Catalog[2]. The OO- \mathcal{H} Method Pattern Catalog provides a user-centered Hypermedia Interface Pattern Language [16] that offers alternative solutions to well-known hypermedia problems, considered from the user point of view. Furthermore, its use allows the designer to choose the most suitable among a set of possible implementations. The patterns can fall into one of the following three categories: (1) Information patterns, that provide the user with useful context information, (2) Interaction Patterns, which cover user-interface communication issues such as protocol-related features for invoking services and (3) Navigation Patterns, that determine the way the user is going to move through the system. The information and patterns captured at the NAD level suffice to automatically generate a default APD, which provides the designer with the skeleton page template structure on which to perform further refinements. This article introduces the APD main semantic and structural features.

The remainder of the article is structured as follows: section 2 gives an overview of the NAD and the concepts captured there, which are the basis for the generation of the default APD. Section 3 introduces the APD and describes in detail, by means of an example, both the concepts and the template constructs associated with this diagram. It also defines its construction process (automatic default generation and refinement). The web interface that is generated from the information captured both in the NAD and in the APD is shown in section 4. Section 5 makes a comparison with related work, and section 6 presents the conclusions and further work.

2 OO- \mathcal{H} Method Navigation Access Diagram

For a more general perspective of the approach, a small example is going to be employed all along the paper: a Chat Management System. As a basic explanation (for reasons of brevity) it is assumed that there are several possible chat topics. Each message corresponds to a single topic. Besides, messages are hierarchically structured so that a message can be the start point of a new discussion line inside its topic or, otherwise, be a response to another previous message. The chat user, whose behaviour we will model, is able to read messages and reply to an existing message. OO- \mathcal{H} Method associates a different NAD diagram with each agent (user-type). This diagram is based on the following constructs:

1. Navigation Classes (NC): they are domain classes whose attributes and methods have been filtered and enriched in order to better accommodate the specific features of hypertext. This enrichment causes different types of attributes to appear: V-Attributes (attributes that are always visible), R-Attributes (available to the user on demand, by means of any kind of

reference) and H-Attributes (attributes hidden to the user except on very specific occasions, such as when displaying detailed views of the system).

- Navigation Targets (NT): they group the elements of the model that collaborate in the coverage of a certain user navigation requirement. In our example (see Fig. 1) there is one NT, corresponding to the user requirement 'Participate in Chat'. Inside it, we can observe two navigation domain classes: the 'Chat' class, which determines the available discussion topics, and the 'Messages' class, which contains the messages stored in the system. We can observe that the 'Chat' class has a single attribute ('nameChat'), which is labelled as 'Visible' (V) and specifies the identifying name for each discussion topic.

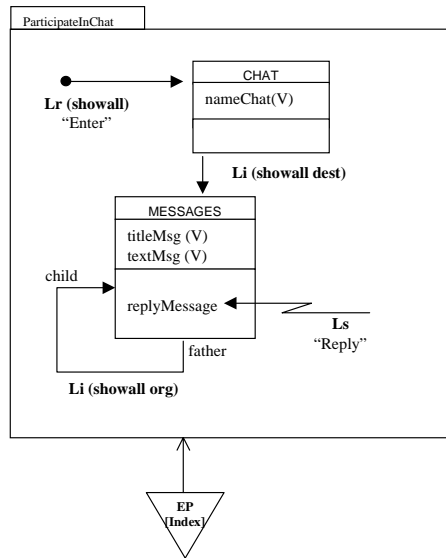


Figure1. NAD Diagram of a Chat Manager System

- Navigation Links (NL): they define the navigation paths through the information. They have a set of associated Dynamic Flow Navigation Patterns, which are defined in the Pattern Catalog and qualify the user navigation behaviour. Also, they are accompanied by a set of Navigation Filters, which restrict and qualify the target information. We can distinguish among four different link types: I-Links (Internal Links), which provide navigation paths among objects inside a given NT, T-Links (Traversal Links), which are defined between navigation classes belonging to different NT, R-Links (Requirement Links), which define the entry point to each NT and S-Links (Service Links), which determine the available services for the user-type as-

sociated to that NAD. In Fig. 1 we can observe three out of the four possible link types. The structural relationship (composition) that exists between 'Chat' and 'Messages' allows the designer to define a Li that makes possible the access to all the messages corresponding to a given chat topic. The Lr 'Enter' shows the navigation entry point to the NT. Also, an example of the Ls link type is provided, associated with the 'Reply' service.

4. Collections: they group objects following certain criteria under hierarchical, either static or dynamic, structures. They have a set of Dynamic Flow Navigation Patterns and Navigation Filters associated, which define both the traversal behaviour of the link structure and the set of objects on which this collection will apply. In OO- \mathcal{H} Method there are three main collection types: C-Collections (Classifying Collections), which provide an access structure, hierarchical or not, to groups of related objects, T-Collections (Transaction Collections), which group navigation services that are offered to the user as a whole and S-Collections (Selector Collections), which group objects that conform to a set of values gathered from the user. In our example we can observe a special kind of C-Collection, the EP (Entry Point), which, as every collection, is drawn as an inverted triangle. An EP determines the entry point to the application.

The NAD captures the navigation paths and the services the user can activate when working with the interface, and so a different NAD should be defined for each user type. From there a web interface might be generated without further work, because OO- \mathcal{H} Method provides a set of default values for the main presentation features. That allows the designer to shorten the time needed to develop application prototypes. However, once an agreement between designer and client has been reached, the designer will most probably need to modify this default structure in order to improve both its appearance and usability features. In order to do so, OO- \mathcal{H} Method defines another diagram: the APD, which will be detailed in next section. In order to get more information about the NAD diagram, interested readers are referred to [8].

3 The Abstract Presentation Diagram

We agree with [1, 5, 6, 11] in the adoption of a template approach for the specification of not only the visual appearance but also the page structure of the web. OO- \mathcal{H} Method defines five template types, expressed as XML (eXtensible Markup Language) documents[4, 10]. In order to define the tags and the structure of the documents we have associated a Document Type Definition (DTD)¹ with each type of template. For reasons of space, the five DTD specifications are left out of the article. Interested readers are referred to [3].

¹ A new proposal, called XML-SCHEMA, is being discussed at [4] as an alternative to the DTD definition language

3.1 Template Types

The five template types defined in our approach are, namely: (1) **tStruct**, which defines the information that will appear in the materialized page, (2) **tStyle**, which reflects the visual features of the page, (3) **tForm**, which defines the data required from the user in order to interact with the system, (4) **tFunction**, which captures language-independent client functionality and is based on the DOM (Document Object Model) specification[4] and (5) **tWindow**, which reflects two or more simultaneous views of the information. With this approach, the addition of new document types to our model simply consists on the addition of (1) a new DTD defining the structure of such document, and (2) a set of mapping rules to each one of the different target environments. Furthermore, defining a common set of XML templates could serve as a framework for comparison among different proposals.

The default template structure can be derived from the information captured in the NAD by using a set of default APD generation rules. Following these rules, V-Attributes, I-Links, T-Links and R-Links are automatically transformed into link-elements inside the tStruct page. Also C-Collections and S-Collections generate a new tStruct abstract page that contains a tree-like structure made up of link elements pointing to other tStruct elements, and so on. In our example, the abstract template pages **Home Page**, **Chat List**, **Message View** and **Reply Message** (which can be seen in Fig. 2) have been automatically derived from the NAD diagram, together with its corresponding links. Also a general **Style** page has been automatically added to the template structure. In the following section we will present the possible ways of refinement that cause the template structure to evolve towards its final appearance.

3.2 APD Refinement

The default APD provides the user with a functional interface, that can serve as a prototype on which to validate the user-requirements. But, in order to get a more sophisticated appearance, the designer will probably need to perform further refinements. OO-*H*Method provides the user with two refinement mechanisms, the simpler one consisting on manually adding structures and/or individual pages to the default APD diagram. As an example, in Fig. 2 we have added a TWindow structure that adds a multi view capability to the default one-view-at-a-time interface. The other mechanism consists on the application of a series of APD-related patterns captured in the Pattern Catalog. These patterns provide the designer with additional hypermedia features and techniques, which are known to be useful to improve the interface quality. Also, the use of patterns makes possible the automation of the refinement process. APD-related patterns have a set of application rules that drive the APD evolution when they are applied. As an example of this second approach, in Fig. 2 the application of the 'head and foot' implementation corresponding to the 'Location Pattern' has caused two new abstract pages (head and foot) of type tStruct, to appear on the diagram.

These two pages are connected to every page where the designer wants them to be included. In fact patterns might cause the appearance and/or modification of any kind of abstract page. For instance, the generated 'ChatList' tStruct abstract page, after applying the refinements, is as follows:

```
<?XML version="1.0"?>

<!DOCTYPE tStruct SYSTEM "tStruct.dtd" encoding="UTF-8">
<tStruct>
  <label style="" text="List of available chats" />
  <link name="error" type="automatic" show="new"
    pointsTo="tStruct" dest="errorPage"/>
  <link name="head" type="automatic" show="here"
    pointsTo="tStruct" dest="head"/>
  <collection format="ulist" style="schatlist">
    <object type="chat">
      <attrib name="nameChat" type="STRING">
      </attrib>
      <call event="onClick" function="validate">
      </call>
    </object>
  </collection>
  <link name="foot" type="automatic" show="here"
    pointsTo="tStruct" dest="foot"/>
</tStruct>
```

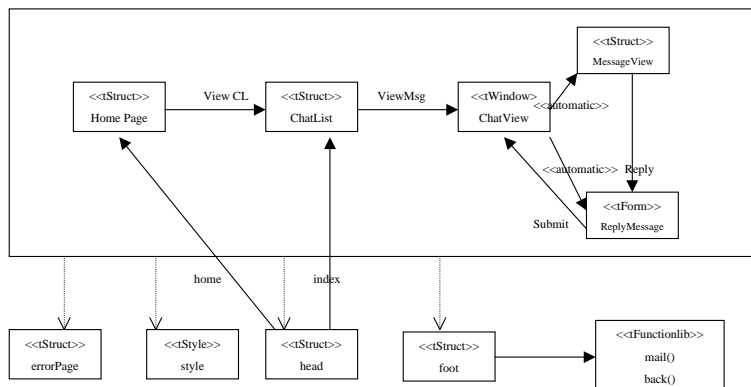


Figure2. Simplified APD of the Chat User Agent

Once the selected patterns have been applied to the diagram, a set of interaction tasks and techniques² will define the implementation constructs for a given environment. Last, but not least, we can enrich the model with redundant implementations for the same pattern: e.g. the Navigation Observer Pattern, which

² both concepts will be introduced in the following section

allows the interface to keep track of the navigation path followed by a given user, might be present in the interface by both a 'back' and a 'reload' mechanism.

4 Implementation of the APD

We might define an Interaction Task as a mechanism that groups, according to the action to be performed, the set of Abstract Interaction Objects (AIO's) that collaborate in the coverage of such action. These elements might have been explicitly chosen by the designer or might be part of any pattern applied to these models. On the other hand, we define an 'Interaction Technique' as each one of the materialization possibilities an Interaction Task has. Interaction Techniques are always associated with a concrete strategy and/or programming environment, and are made up of Concrete Interaction Objects (CIO's). The complexity of the mapping among the abstract pages in the APD and the final constructs (first to AIO's and then to CIO's [12]) goes beyond the purpose of this article. In Fig. 3 to 5 the interface generated from the APD of Fig. 2, which corresponds to the Chat Manager example, is shown.

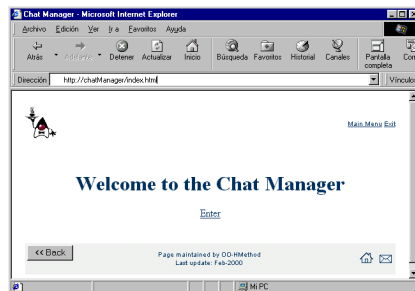


Figure3. Chat Manager entry point

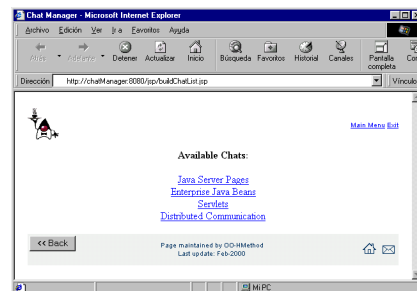


Figure4. List of available Chat Lists

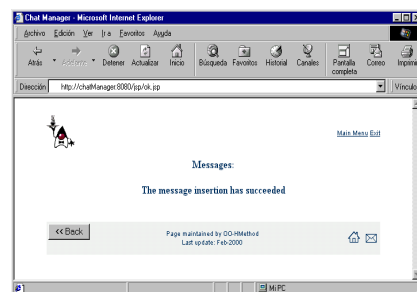
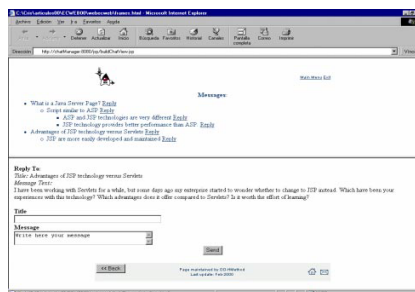


Figure5. Adding an opinion to the chat

The process is as follows: first, the generator tool looks for the page template derived from the Application Entry Point (see Fig. 3). Note that every page of the diagram has the same head/foot associated, which provides the interface with a common visual context (Location Pattern). When the user clicks on the 'Enter' link, the ChatList tStruct page is populated with the active application objects and the materialized HTML page is shown (see Fig. 4). Again, when the user clicks on the name of one of the Chat topics, the materialization of the tWindow abstract page is performed. This template defines the generation of the, from now on, two different and simultaneously available views of the system by means of two 'automatic' links, that is, links that don't require the user interaction in order to be activated. Those views are (1) the messages kept on the system (again a tStruct abstract page) and (2) a tForm abstract page that encapsulates the fields required to add a new message to the application. The `replyMessage` function returns a Boolean value, which provokes the final Ok message to appear once the operation has been successfully fulfilled (see Fig. 5).

The sample application has been developed using JavaServer Pages and Bean components [18] as the chosen server technology, and HTML as the chosen client technology.

5 Comparison with Related Work

Many commercial applications make use of some kind of templates in their hypermedia development approach. IDC's or ASP's from Microsoft, or Cold Fusion from Allaire are some examples. The main drawback of these approaches is that they remain too close to the implementation space, and thus the designer has to deal with error-prone activities such as specifying exact names of database fields, or explicitly managing the linkage of pages. Our template structure on the contrary follows, like others [6, 13, 17] a declarative approach that covers every aspect of the interface, from content to style. As an example, the tStyle template type adds a further level of abstraction to the CSS approach, followed in many traditional applications: while CSS pages work on documents with a definite hypermedia structure, tStyle templates act on abstract pages on which the hypermedia structure is not still defined. CSS are, again, just a possible final materialization (among many others) of tStyle templates.

Although the template approach is not the only possible approach (see for example [17]), it however provides us with the required flexibility and extensibility we consider vital for our method in such a changing environment as the web. Our template concept shares with other models studied so far [1, 5, 6, 11, 17] many similarities: it implicitly assumes a page visualization schema and somehow specifies the data it is going to show. But there are just as many features in which our work is different: generally speaking, we consider mixing in a single template features regarding structure, content, presentation and behaviour, overdimensions such templates, and difficults the designer task. On the contrary,

the OO- \mathcal{H} Method taxonomy of templates facilitates to focus on complementary aspects of the final implementation. We also claim that the separation of content and layout [6] doesn't suffice to deal with the different aspects involved in presentation, such as client functionality, interaction with logic or several simultaneous views of the system. We haven't found, up to now, any other abstraction proposal for such characteristics apart from that included in OO- \mathcal{H} Method. Also, the definition of the different aspects of the interface by means of a set of patterns greatly simplifies both the construction and modification of the APD, and also improves its usability.

6 Conclusions and further work

OO- \mathcal{H} Method is an extension of the OO-Method conceptual modeling approach to address the particulars associated with the design of web interfaces. In this article we have presented a new diagram, the APD, based on the concept of constructive templates, which provides the designer with an intuitive way of refining the default interface structure, previously captured in the NAD diagram. We have also proposed an interface enrichment process driven by patterns. Furthermore, the article illustrates how OO- \mathcal{H} Method captures application interaction issues that, in spite of its relevance in web application interfaces, we have found missing in other proposals.

Summarizing, the most relevant contributions of this paper are the following:

1. A taxonomy of templates, defined in XML, that separates the different complementary views involved in the complete definition of the interface.
2. A process for the APD refinement.
3. A set of ways in which the use of the OO- \mathcal{H} Method Interface Pattern Catalog improves and facilitates the diagrams construction and refinement and influences the user interface quality.

At the moment we are applying this method to an e-commerce application. The experience gained in the development of this interface will surely enrich our pattern catalog and refine our template structure. Also, a taxonomy of interaction tasks and techniques (already in the solution space) is being defined.

Acknowledgments We would like to thank the anonymous referees for their valuable comments to this work

References

- [1] P. Atzeni, G. Mecca, and P. Merialdo. Design and Maintenance of Data-Intensive Web Sites. In *Advances in Database Technology - EDTB'98*, pages 436–449, 03 1998.

- [2] C. Cachero. The OO- \mathcal{H} Method Pattern Catalog. Technical report, Universidad de Alicante, 12 1999.
- [3] C. Cachero. The OO- \mathcal{H} Method Template Taxonomy. Technical report, Universidad de Alicante, 02 2000.
- [4] eXtensible Markup Language (XML). <http://www.w3.org/XML/>.
- [5] F. M. Fernández, D. Florescu, J. Kang, A. Levy, and D. Suci. Catching the Boat with Strudel: Experiences with a Web-Site Management System. In *Proceedings of ACM SIGMOD International conference on Management of data*, pages 414–425, 10 1998.
- [6] P. Fraternali and P. Paolini. A Conceptual Model and a Tool Environment for Developing more Scalable, Dynamic, and Customizable Web Applications. In *Advances in Database Technology - EDBT'98*, pages 421–435, 1998.
- [7] F. Garzotto and P. Paolini. HDM A Model-Based Approach to Hypertext Application Design. *ACM Transactions on Information Systems (TOIS)*, 11(1):1–26, 01 1993.
- [8] J. Gómez, C. Cachero, and O. Pastor. Extending a Conceptual Modelling Approach to Web Application Design. In *CAiSE '00 (to appear). 12th International Conference on Advanced Information Systems*. Springer-Verlag. Lecture Notes in Computer Science, 06 2000.
- [9] T. Isakowitz, E. A. Stohr, and V. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. *CACM: Communications of the ACM.*, pages 34–44, 08 1995.
- [10] S. McGrath. *XML by Example. Building e-commerce Applications*. Prentice Hall, 1998.
- [11] G. Mecca, P. Merialdo, P. Atzeni, and V. Crescenzi. The ARANEUS Guide to Web-Site Development. Technical report, Universidad de Roma, 03 1999.
- [12] P. Molina. Especificación de la Interfaz de Usuario en OO-Method. Technical report, Universidad Politécnica de Valencia, 1998.
- [13] M. Nanard, J. Nanard, and P. Kahn. Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates. In *HYPERTEXT '98. Proceedings of the ninth ACM conference on Hypertext and hypermedia: links, objects, time and space—structure in hypermedia systems*, pages 11–20, 1998.
- [14] O. Pastor, E. Insfrán, V. Pelechano, J. Romero, and J. Merseguer. OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods. In *CAiSE '97. International Conference on Advanced Information Systems*, pages 145–158, 1997.
- [15] O. Pastor, V. Pelechano, E. Insfrán, and J. Gómez. From Object Oriented Conceptual Modeling to Automated Programming in Java. In *ER '98. International Conference on the Entity Relationship Approach*, pages 183–196, 1998.
- [16] G. Rossi, D. Schwabe, and A. Garrido. Design Reuse in Hypermedia Applications Development. In *Proceedings of the eight ACM conference on HYPERTEXT '97*, pages 57–66, 1997.
- [17] D. Schwabe, G. Rossi, and D. J. Barbosa. Systematic Hypermedia Application Design with OOHDM. In *Proceedings of the the seventh ACM conference on HYPERTEXT '96*, page 166, 1996.
- [18] The source for java technology. <http://java.sun.com>.