On Conceptual Modeling of Device-Independent Web Applications: Towards a Web Engineering Approach

Jaime Gómez[1], Cristina Cachero[1], and Oscar Pastor[2]

[1] Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante. SPAIN
{jgomez,cachero}@dlsi.ua.es
[2] Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia. SPAIN
opastor@dsic.upv.es

Abstract Existing tools intended to build and deploy engaging complex Web sites (including functionality) have shown to be inadequate to face the software production process in an unified and systematic way, from a precise system specification to the corresponding final implementation. In this context, where new technologies are continuously emerging, new approaches are required for the web developer to address the entire software lifecycle, from design to development to deployment to maintenance, in a way that is consistent, efficient and lets developers write device-independent applications.

To address this concern we report on OO-$\mathcal{H}$Method, a methodological proposal that, using an Object-Oriented approach, captures all the relevant properties involved in the modeling and implementation of Web Application Interfaces. The OO-$\mathcal{H}$Method design process involves the construction of two additional views, complementary to those captured in traditional, UML-compliant, conceptual modeling approaches. These are, namely, the Navigation View, which extends a class diagram with hypermedia navigation features, and the Presentation View, in which the different elements regarding interface appearance and behavior are modeled by a series of interconnected template structures, expressed in XML. An Interface Pattern Catalog and a CASE tool complete our proposal. As a result, a device-independent front-end specification is obtained. From there a web interface, easily integrable with preexistent logic modules, can be generated in an automated way.

1 Introduction

The well-known characteristics of the web, such as ubiquity, existence of open standards, interlinking, easy access to information and services or easy content creation has not only offered a set of new business opportunities, such as e-commerce or on-line auction systems, but it has also nurtured a migration need for both Business to Consumer and Business to Business applications inside the enterprise boundaries, in order to take advantage of the latest technologies on the web. Some of the several generally avowed benefits in using this new platform are a lower software development and maintenance budget, and a flatter learning curve for new users of the system.

However, these benefits are not always being achieved: the mushrooming of ad hoc developed Web-based applications is generating great problems when the enterprise faces aspects such as its maintenance or evolution, which are basic requirements in such a changing environment as the web. Furthermore, this new brand of software products differ from traditional information delivery systems in that they require non-trivial functionality to be provided to the user, and ad hoc development processes have already proven unsuccessful to address functionality issues. In order to keep the possibility of failure to a minimum, the development process for such applications, either new or migrated, should evolve in a sound scientific way in order to adapt to the new environment. Such is the aim of the emerging Web Engineering[21] discipline, which deals with development, deployment and evaluation issues regarding high quality Web-based systems and applications. In this development process, we have placed our work inside the Product Model phase[15], where information, presentation and behavioral issues related to web applications are meant to be captured in a systematic way.

In order to approach the construction of this model, Web applications have usually been regarded as hypermedia applications and so several existing proposals (see e.g. [10, 2, 11, 14]) have been developed

just taking into account the information dissemination and presentation aim of such systems. More recent proposals [6] have already detected the need for modeling functionality, and so have extended the models with some simple operational behaviour. Also, the increasing complexity of web interfaces has caused research efforts to be directed towards sound development methodologies centered on the idiosyncrasy of the interface and its communication mechanisms with underlying application logic. Such is the case of UIML[29].

We claim that existent software engineering approaches can be successfully extended with new models that specifically gather web-related characteristics. Our decision has been based on three assumptions: (1) behaviour 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 behaviour inside web applications should be ideally addressed with already tested OO software engineering methods, as in any other distributed application, and (3) in spite of the same underlying logical layer, interface appearance and behaviour greatly differ between web and traditional applications.

Following the trend presented in [17, 7], our approach, known as OO-$\mathcal{H}$Method, looks at web systems as unified software artifacts where structure, behaviour and presentation are all of them basic pieces that must be properly combined to get a correct final software product. OO-$\mathcal{H}$Method provides as a main contribution a framework to capture all the relevant properties involved in the modeling and implementation of Web Application Interfaces. The OO-$\mathcal{H}$Method design process involves the construction of two additional views, complementary to those captured in traditional, UML-compliant, conceptual modeling approaches. These are, namely, the Navigation View, which extends a class diagram with hypermedia navigation features, and the Presentation View, in which the different elements regarding interface appearance and behavior are modeled by a series of interconnected template structures, expressed in XML. An Interface Pattern Catalog and a CASE tool complete our proposal. As a result, a device-independent front-end specification is obtained. From there a web interface, easily integrable with preexistent logic modules, is generated in an automated way.

The remaining of the article is structured as follows: section 2 provides a brief overview of OO-$\mathcal{H}$Method in the context of traditional software development. Section 3 presents the OO-$\mathcal{H}$Method framework and describes in detail, by means of a comprehensive example, its main components, namely a design process, a navigation and a presentation diagram and Interface Pattern Catalog that enriches the development process. This section is completed with the resulting web interface that is generated after applying our method. Section 4 introduces the main characteristics of the CASE tool that gives support to the different methodological aspects of OO-$\mathcal{H}$Method. A comparison with related work is presented in section 5, while section 6 sketches the conclusions and main contributions of our approach.

2   OO-$\mathcal{H}$Method: An Overview

OO-$\mathcal{H}$Method is a generic model 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. In order to achieve this goal, OO-$\mathcal{H}$Method[16, 5] is based on the information reflected in a UML-compliant[23] approach, known as OO-Method[25, 24]. For the purpose of this paper (see Fig. 1) it suffices to know that OO-Method is an automated software production environment whose main constituents are:

- A set of views to capture the system structure (statics) and behaviour (dinamics).
- A model compiler that, departing from the captured specification, is able to generate the data sources and the logic modules in the desired implementation environment.

OO-$\mathcal{H}$Method extends these views with two new complementary diagrams: (1) the Navigational Access Diagram (NAD), that defines a navigation view, and (2) the Abstract Presentation Diagram (APD), that 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 which is integrated in the OO-$\mathcal{H}$Method proposal.

Following the OO-Method philosophy, OO-$\mathcal{H}$Method provides a model compiler that generates the internet application front-end for the desired client platform and/or language (HTML, XML, WML).
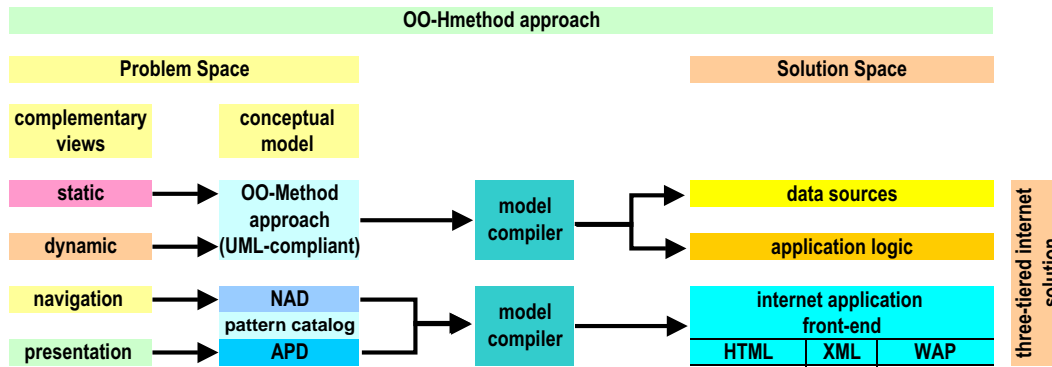
Figure1. *OO-HMethod: An Overview*

This extension provides a true three-tiered internet solution as can be observed in Fig. 1. Next, we are going to present how the OO-HMethod proposal provides the mechanisms to support this process.

3   The OO-HMethod Proposal

OO-HMethod includes a set of notations, techniques and tools that together conform a sound approach to the web product modeling phase. These are:
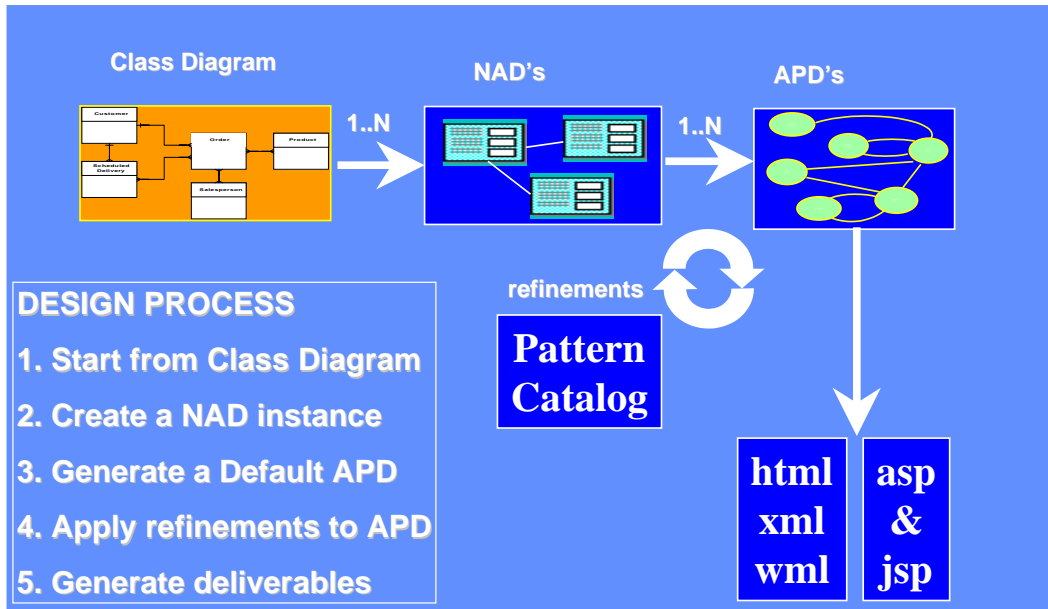
- a design process
- a Pattern Catalog
- a NAD
- an APD
- a CASE tool that allows to automate the development of web applications modelled with OO-HMethod

In the following sections we will further develop each of the concepts.

3.1   Design Process

The design process defines the phases the designer has to cover in order to build a functional interface that fulfills the user requirements. As shown in Fig. 2, the OO-HMethod design process departs from a UML-compliant class diagram that captures the domain information structure. From there, different NAD instances are modeled 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, following a set of mapping steps, a default web interface. This automatic generation feature allows the designer to 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. In order to improve the interface quality, OO-HMethod introduces a second diagram, the APD, based on the concept of templates [2, 9, 11, 20] and whose default structure is also directly derived from the NAD. In order to help the designer to refine this structure while maintaining its quality, the Pattern Catalog contains a set of constructs proven as effective solutions to so far identified problems inside web environments. This approach facilitates the reuse of design experiences and the consistency both among the different interface modules and among application interfaces. Once the APD has been refined, a web application front-end, either static or dynamic, can be generated for the desired environment (HTML, WML, ASP's, JSP's...). This independence from final implementation issues is a must-have feature in an environment where new appliances and languages for internet access are constantly emerging.

Both the NAD and the APD make extensive use of a Pattern Catalog, which is presented next.

Figure2. *OO-HMethod: The Design Process*

3.2 Pattern Catalog

The OO-HMethod Pattern Catalog provides a Hypermedia Interface Pattern Language. This language can be seen as a partially ordered collection of related patterns that work together in the context of Hypermedia Interfaces [26]. They help to capture the abstract interaction model between the user and the application. We have chosen the pattern style defined in [4] for the specification of the patterns. This style, largely based on the Alexandrian style [1] is best suited for abstract patterns with several possible ways of implementation, which are common characteristics of those found in hypermedia environments. We have however enriched the implementation specification in order to allow them to drive the evolution of the diagrams: each pattern has one of its possible implementations set to 'default', in order to help the designer to obtain the desirable interface features with a minimum effort. Also, each implementation has an associated Transformation Rule, expressed in OCL-like[30] syntax. These transformation rules can be instantiated and applied at different levels (from page level to schema level), and they drive the changes on the diagram where the patterns are applied. These changes include the creation of new pages, the redirection of links among pages or the creation of new page dependencies, among others. Furthermore, in order to improve its usability, the catalog contains a Sample Usage section that points to working web examples where it has been successfully applied.

One of the main features of our catalog is that it is, as the rest of the model, 'User-Centered', that is, the granularity at which the patterns are described is such as to provide the designer with additional mechanisms to fulfill the user requirements. The patterns included in the catalog offer 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 alternative implementations, depending on the target application domain and on the designer's experience.

The structure of the catalog can be seen in table 1. The main categories are:

1. Information Patterns: they provide the user with useful application context information. One of the most relevant examples is the 'Location Pattern' whose definition will be presented later.
2. Interaction Patterns: its use involves user-interface communication issues regarding both functionality and navigation.
3. User Schema Evolution Patterns: they cover structural advanced features. As an example we can cite the 'Multiview Pattern', which allows the designer to present two or more simultaneous views of the information.

In order to show the utility of the catalog in the design process, we are going to illustrate its use by means of an example.

Information Patterns
Location Pattern
Interface State Pattern
System State Pattern
Destination Announcement Pattern
Error Pattern
Success Pattern
Help Pattern
Population Observer Pattern
Active Agent Observer Pattern

Interaction Patterns				
Identification Pattern	Navigation Patterns			Command Control Patterns
Population Filtering Pattern	Static Navigation Patterns	Dynamic Navigation Patterns		Command Observer Pattern
	Cycle	Flow Patterns	Jump Patterns	Confirmation Pattern
	Tree	Index	Annotation Pattern	
	Sequence	Guided Tour	Chooser Pattern	
	Split-join	Indexed Guided Tour	Navigation Observer Pattern	
	Dynamic	Showall	Navigation Selector Pattern	

User Schema Evolution Patterns
Multiview Pattern

Table 1. *Pattern Catalog*

A Pattern Example In OO-$\mathcal{H}$Method, different sets of patterns can be applied to the two different diagrams of the modeling process, which are (1) the NAD diagram and (2) the APD diagram. At the NAD level we can apply patterns related to user information selection and navigation behaviour. At the APD level, on the contrary, we can apply patterns that provide the interface with non-mandatory additional features, aimed at improving 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. Two forces are associated with this problem:
   (a) The path used to reach the information components is not known a priori.
   (b) The location component should be loosely coupled: the system view should not 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 head and foot to each group of related pages.
   Transformation Rule[1]:
   ...
7. Other implementations:
   (a) Adapt the Observer Pattern[13] to hypermedia environments.
   ...

Next, we are going to introduce the main structural and semantic aspects of the NAD diagram, whose definition constitutes the first step in the interface modeling process.

3.3 Navigational Access Diagram

For a more general perspective of the approach, a small example is going to be employed throughout the paper: a Discussion List Management System. As a basic explanation (for reasons of brevity) it is assumed that the list manager system contains several discussion lists dealing with different Web Technology topics, and that each list is formed by a set of hierarchically ordered messages, related one to another by a parent-child unary relationship. The discussion list user is able to read all the messages included inside any of the lists, as well as to reply to any of them.

---

[1] The transformation rule for this pattern is presented in table 3, and will be understood after reading section 3.4, when the different APD concepts will have already been introduced.

As stated above, the navigation model is captured by means of one or more NAD's. The designer should construct as many NAD's as different views of the system are required, and he should provide at least a different NAD for each user-type (agent-type) who is allowed to navigate through the system. This diagram is based on four types of constructs: (1) Navigational Classes, (2) Navigational Targets, (3) Navigational Links and (4) Collections. Also, when defining the navigation structure, the designer must take into account some orthogonal aspects such as the desired navigation behaviour, 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. Next we are going to further develop all these concepts:
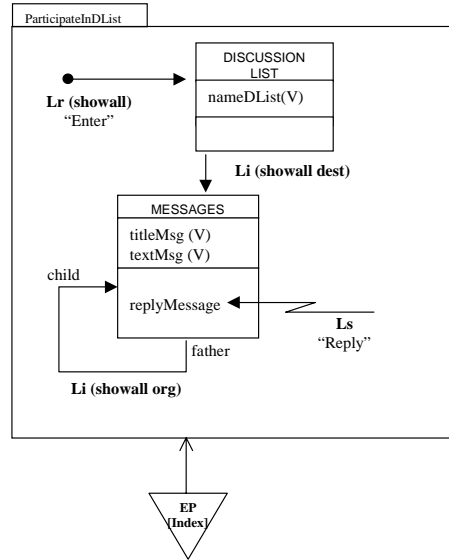
Figure3. *NAD Diagram of the discussion list system*

– Navigational Classes (NC): they are enriched domain classes whose attribute and method visibility has been restricted according to the user access permissions and navigation requirements. A sample enrichment is the differentiation among three types of attribute: 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).
  In our example (see Fig. 3) we have modeled 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. The reason for this visibility is that this name unambiguously identifies each discussion topic inside the interface, and so it must always be present. The 'Messages' class has, besides its identifying attribute 'titleMsg', 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 (NT): they group the elements of the model that collaborate in the coverage of each user navigational requirement.
  Again looking at our example (see Fig. 3), we can observe how we have, due to its simplicity, a single navigational requirement: 'ParticipateInDList', that contains both the information and the services needed for the user to effectively navigate through the system.
– Navigational Links (NL): they define the navigation paths the user is able to follow through the system. They may have both a Navigation Pattern and a set of Navigation Filters associated, which together provide the required 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 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; finally S-Links (Service Links) show the services available to the user type associated to that NAD.

In Fig. 3 we can 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, 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 labelled 'Enter' points at the navigation class from which the user will start the navigation inside that NT. Finally, an example of S-Link is associated with the 'ReplyMessage' service inside the class 'Messages', which allows the user to answer any previous message. All the service parameters (in our example the answer title and text) that are not feed to the service by means of filters associated to the S-Links will have to be introduced by the user at execution time.

– Collections: they are (possibly) hierarchical structures defined on Navigation Classes or Navigation Targets. They provide the user with new ways of accessing the information. OO-$\mathcal{H}$Method defines several types of collections, among which we could cite (1) C-Collections (Classifier Collections), where the object population criteria is predefined, and (2) S-Collections (Selector Collections), where these criteria need to be introduced by the user at execution time, and so require a new form to be generated on the fly from the parameters specified for such collection.

In the example we see a single C-Collection, drawn as an inverted triangle and associated to the NT 'ParticipateInDList'. This C-Collection determines the Entry Point (EP) to the web application.

For more information on the diagram, interested readers are referred to [16].

Commercial interfaces tend to require a greater level of sophistication than that provided by the NAD diagram, regarding both appearance and usability features. In order to refine the interface, OO-$\mathcal{H}$Method defines another diagram: the APD, which will be detailed in next section.

3.4 Abstract Presentation Diagram

We have adopted a template approach for the specification of not only the visual appearance but also the page structure of the web. In order to separate the different aspects that contribute to the final interface appearance and behavior, we have defined five types of templates, expressed as XML (eXtensible Markup Language) documents[8, 18]. In order to define the tags and the structure of the document we have associated a Document Type Definition (DTD)[2] with each type of template, whose description we will give in the following section.

Template Types The five template types defined in OO-$\mathcal{H}$Method are, namely:

1. tStruct: its instances define the information that has to appear in 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 in order to interact with the system.
4. tFunction: its instances capture client functionality. They are based on the DOM (Document Object Model) specification[8], which aims at giving a platform and language-independent interface to the structure and content of XML and HTML documents. It also seeks to standardize an interface to these objects for navigation and document processing. The functions defined in this kind of template are language-independent, and so they must be mapped to a target language (either JavaScript, which captures HTML-specific components of the DOM, or another) in order to become operative.
5. tWindow: its instances define a set of simultaneous views available to the user.

For reasons of brevity, we are just showing the tStruct DTD. A DTD specifies the set of rules that define a valid XML document. It thus defines the tags allowed to 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 etc.

---

[2] A new proposal, called XML-SCHEMA, is being discussed at [8] as an alternative to the DTD language

tStruct DTD

```
<!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 that:

1. A tStruct is formed by 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 be associated with every element of the page, defines the trigger condition for an action to be performed by the interface.
3. A link can have zero or more calls associated. A link is defined by a name, a type and a set of attributes related to its behaviour (where it is going to show the destination page, which is this destination page and its type). Only links with the 'show' attribute set to 'new' (meaning that they point to a new abstract page) are shown in the APD diagram.
4. An object contains one or more attributes and zero or more calls. It has also both a type (class to which it belongs) and a (possibly missing) style associated.
5. An attribute can have any number of calls associated. It has also a (required) name, a type and a boolean 'order' value that indicates whether the order in which the objects will appear on the screen depend on the value of that attribute.
6. Last, but not least, a tStruct page can have any number of labels which capture the static text appearing in the interface.

The default template structure can be derived from the information captured in the NAD combined with a set of defaults defined for the undefined values. This default generation will be explained in the following section.

Default APD OO-$\mathcal{H}$Method defines a set of steps for the mapping from 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 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. In our example (see Fig. 4) the automatically generated pages are: (1) the 'Home Page', 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 Fig. 3) provide the information necessary for the connection of these APD pages.
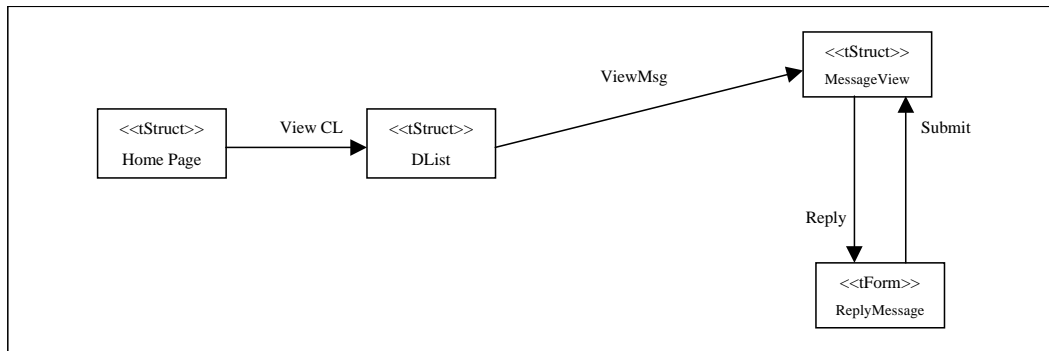
Figure4. *Default APD for the discussion list system*

The main mapping steps that drive this default APD generation are described in table 2.

1. V-Attributes, I-Links, T-Links and R-Links: they appear as elements of the tStruct page.
2. C-Collections, S-Collections: they are static trees, and are defined by means of a tStruct single abstract page that contains a tree-like structure made up of link elements pointing to other tStruct elements. An S-Collection also might imply a new tForm template to be created, if the filter associated with it involves any user-dependent value.
3. S-Links: they may generate a previous tForm abstract page that contains the parameters to be introduced by the user for 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 OO-Method). If the command returns anything, another tStruct page will be generated with the structure defined by this return value (either an object or a set of objects).
4. R-Attributes: they 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 will contain all the R-Attributes to be shown plus a meaningful reference to the original object.
5. NAD Navigation Patterns: all indexes, guided tours, indexed guided tours and showall patterns are captured in the APD by means of a set of link elements defined on a single tStruct page.
6. All objects to be shown are enclosed in collections, and there are default user-defined data validations defined for the main attribute types. These validations are implemented by means of call elements with 'onChange' events functions associated with validation functions that depend on the attribute type.

Table2. Default APD mapping rules

APD Refinement The refinement process consists 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 catalog. The changes introduced in the APD when the designer decides to apply a pattern are described by means of a Transformation Rule (expressed in an OCL-like syntax[30]) associated to any of its possible

implementations. As an example, Tables 3 and 4 illustrate the definition and instantiation of the Location Pattern transformation rule.

LOCATION PATTERN TRANSFORMATION RULE

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

//associated tfunctionlib functions (non mandatory)
[fvarpage= <APDSchema>->select(type=`tFunctionlib`);
<APDSchema>->select(name=varpage.name)->AddFunction(fvarpage.function);]*

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

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

]+

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

Table3. Location Pattern Transformation Rule

TRANSFORMATION RULE INSTANTIATION

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

Table4. Transformation Rule Instantiation for the Location Pattern

Every APD has an identifying name that acts as a root element from which every other element of the diagram is derived. That root element is of type 'APDSchema', and in our example corresponds with 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 a new page has been added to the schema, we must define its name and its type. In addition, these new pages can have any number of functions associated (as for example a 'mail to webmaster' service). These functions must have been previously defined in a function repository that is made available to the APD by means of a tFunctionLib abstract page. Also, we could decide to associate a style to these new created pages, and/or some links to other pages, such as to the home page of the site.

As we can observe in Fig. 5, the application of the transformation rule instantiation to the DList default APD showed in Fig. 4 has caused two new pages to appear: a head and a foot page. Those pages are related to every TStruct and TFunction page in the diagram, what essentially means that every page generated from that APD will share the same layout context. The foot page has two related functions: 'mail' and 'back', whose behavior is described in the abstract 'tFunctionLib' page. Also, the transformation rule has added a link from the 'head' construct to the application home page.

In our example, the inclusion of an Error Pattern has similarly caused a new abstract tStruct page to appear on the schema, as well as a set of dependency arrows to make the pages where an error might occur aware of it. Also, the designer has applied the Multiview Pattern, which has created a tWindow construct.

The patterns might cause the appearance and/or modification of any kind of abstract page. As a further example, the application of the Confirmation Pattern would cause a confirm function to be included before any change of the type specified is requested to the system. So the content of the function page, where the client-logic is included, would change.

In OO-$\mathcal{H}$Method we may also choose whether to make the chosen patterns visible or not. For example, the physical pages and the set of arrows implied by the 'Location Pattern' don't provide much meaningful information, so we could decide not to show them on the diagram. The same would happen with the Error Pattern. Using implicit patterns clarify the schema and avoid 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"
```

Figure5. *APD refinement of the discussion list system*

```
        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>
```
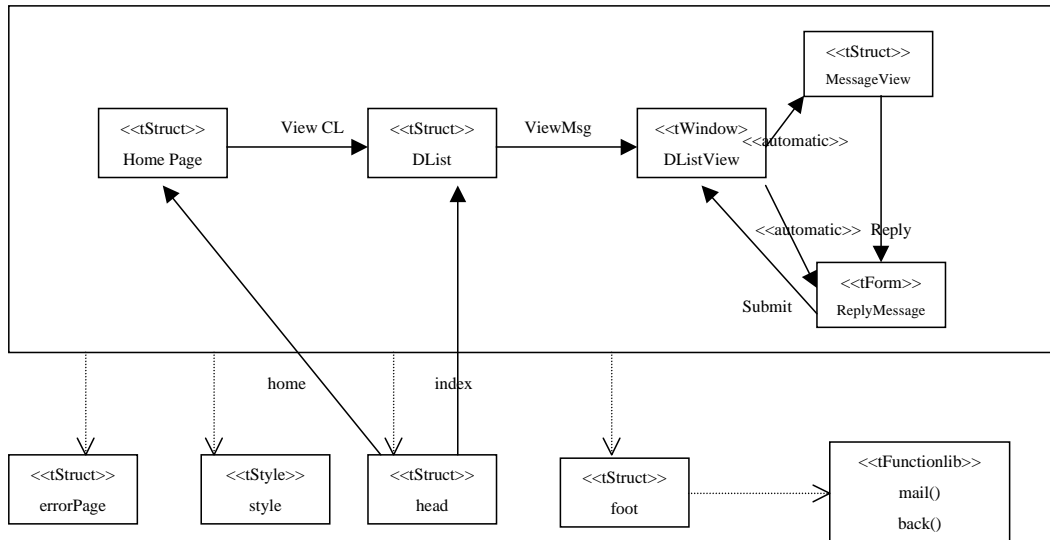
After having refined the APD, the device-independent modeled interface features are fed to a model compiler (not discussed here) that has the target-environment knowledge that allows it to generate an operational web interface.

The generated Front-End In Fig. 6 to 8 the interface generated from the APD of Fig. 5, which corresponds to the Discussion List system, is shown.

The process is as follows: first, the model compiler tool looks for the page template derived from the Application Entry Point (see Fig. 6). Note that every page of the diagram has the same head/foot associated, which provides the interface with a common visual context.

When the user clicks on the 'Enter' link (which is defined with the 'show new' navigation pattern associated), an object population of the 'DList' tStruct is performed and the materialized HTML page is shown (see Fig. 7). 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 materialization of the tWindow abstract page is performed. This template captures rather a different behaviour from that of tStruct: instead of populating the template with objects, it defines the characteristics of the, from now on, two different and simultaneously available views of the system. It defines thus two links whose type is set to 'automatic' (meaning that they don't require the user interaction in order to be followed) and that are in charge of the load of the two actual views: the messages kept on the system (again a tStruct abstract page) and a form with the fields that must be introduced by the user in order to add a new message to the application.

The function returns a Boolean value, which makes the final Ok message appear once the operation has been successfully fulfilled (see Fig. 8).
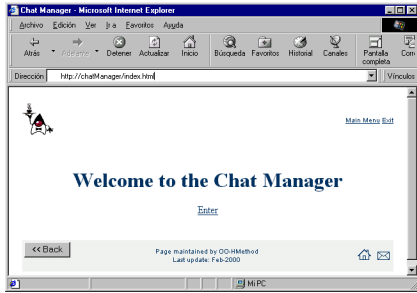
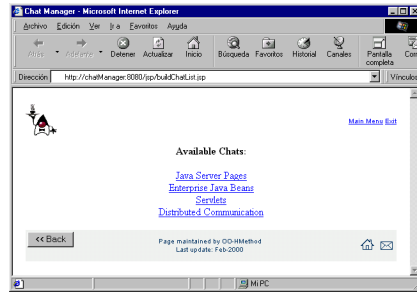**Figure6.** Discussion List entry point

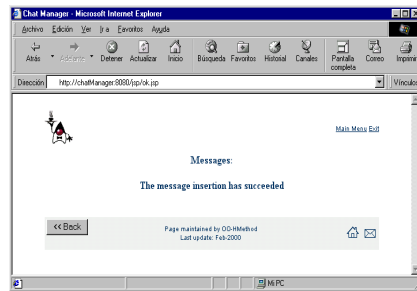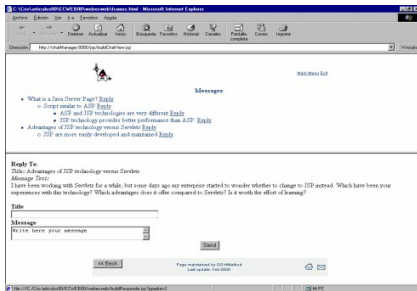

Figure7. Available Discussion Lists

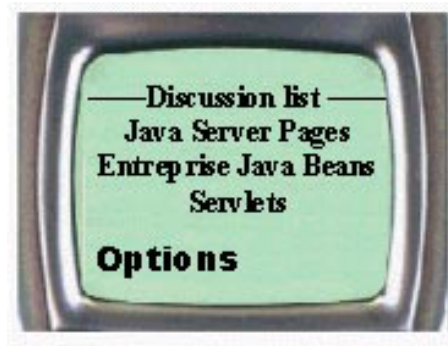**Figure8.** Adding an opinion to the list

Figure9. *Discussion list system in a WAP device*

The sample application has been developed using JavaServer Pages and Bean components [28] as the chosen server technology, and HTML as the chosen client technology.

The model compiler design allows the code generation for other environments. A snapshot of the interface generated from the same specification for a different appliance (in this case a WAP device) can be seen in Fig. 9.

4   The OO-$\mathcal{H}$Method CASE tool

The CASE tool provides an operational environment that supports all the methodological aspects of OO-$\mathcal{H}$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-$\mathcal{H}$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. Fig. 10 shows a snapshot of the CASE tool. It captures a NAD instance for the sample Discussion List application, where an abstraction of the 'ParticipateInDList' Navigational Target is included.
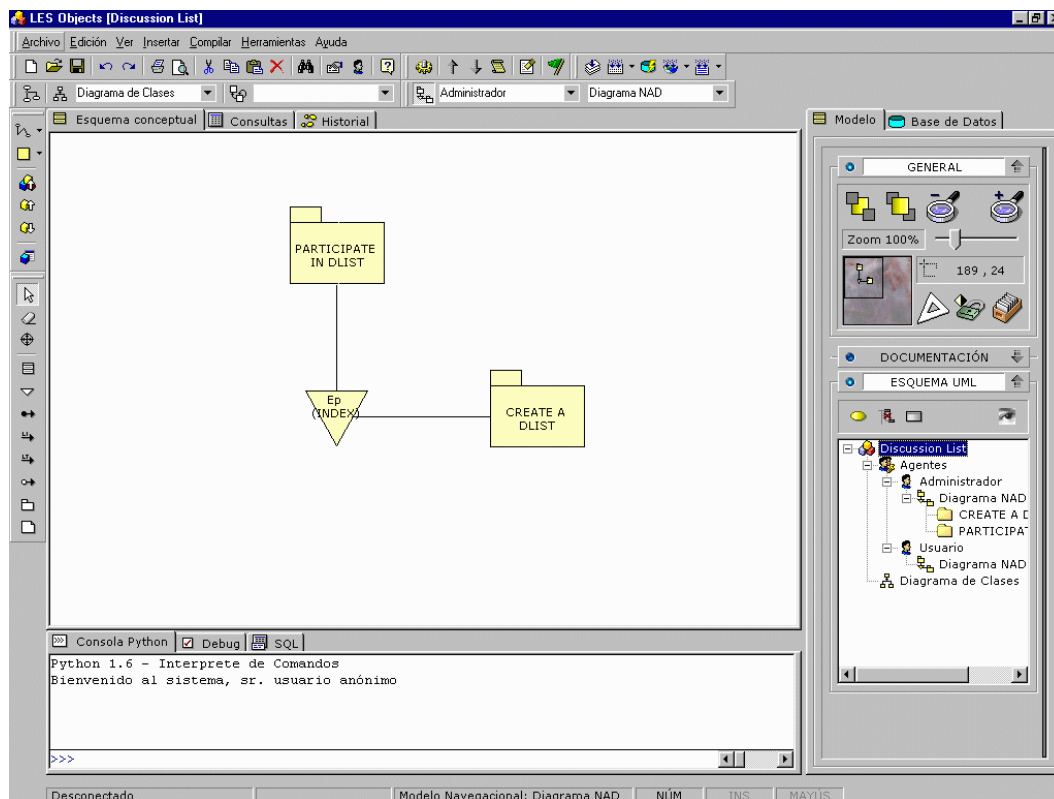

Figure10. *The OO-$\mathcal{H}$Method CASE Tool*

The CASE Tool is being used at this moment for the resolution of real e-commerce systems, in the context of a R&D project carried out jointly by the University of Alicante and a set of industrial partners.

5   Comparison with Related Work

Interface designers have traditionally tackled their task directly using appliance-specific languages. That is the case of many commercial applications, such as IDC's or ASP's (Microsoft) or Cold Fusion (Allaire).

This approach has many disadvantages [29], among which we could cite 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 interface evaluation difficulty.

Partial solutions to these problems can be found in template engines such as MyXML[22], which clearly separate logic, layout and structure. Other systems such as Strudel [10] have gone one step further and have defined a declarative approach to the integration of heterogeneous information sources from where to produce data-intensive web sites. As in MyXML, the layout structure is achieved by intermixing HTML with an specific template language that access the information structure. Although more abstract that commercial applications, they adopt an approach closer to the solution space than to the conceptual (domain problem) space.

There is another group of proposals that tackle the modeling problem from a more abstract point of view. Such is the case of Araneus [20] or Autoweb[11]. These proposals share with OO-$\mathcal{H}$Method the clear separation among 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. However, our proposal has a number of features that makes it overall different from other models. These models are focused on the modeling of hypermedia systems, oriented to the information navigation and visualization [3] but not to the interaction with complex logic. On the contrary, OO-$\mathcal{H}$Method extends the applications modeled with OO-Method, and provides specific mechanisms for the modeled interface to interact with the underlying logic modules gathered in OO-Method. The resulting hypermedia applications cover both the static structure and dynamic behaviour. Another important difference is that, opposed to proposals where underlying database structure drives the model process, we share with OO-HDM[27] the use of an user-requirement, object oriented approach. This fact has allowed us to use the knowledge domain, implicit in OO models, in order to improve the interface usability.

The appearance of new technologies and standards, such as XML, has caused many proposals to evolve. Such has been the case of Araneus-XML[19](Araneus model translation to XML), WebComposition[12, 14] (an extension to the OO-HDM model) or WebML[6] (Autoweb evolution). In Araneus-XML the ADM conceptual schema is translated into an XML specification (similar in concept to our tStruct templates). Over this specification, a page layout is defined in an HTML-like syntax. In this way the same web-site definition can produce web applications for different target environments.

WebComposition is an OO model that acts as an intermediate model between a design model (OO-HDM) and a coarse-grained web implementation model. In order to do so, it translates OO-HDM conceptual constructs into a set of WCML (WebComposition Markup Language) documents. The WCML language includes HTML code and allows web engineers to reuse design and code fragments.

WebML provides a language to define all stages of the development process, from structure to personalization of the site. The definition of an extensible set of unit types and the explicit consideration of personalization issues add great flexibility to this proposal. It shares with OO-$\mathcal{H}$Method the existence of a set of mapping mechanisms from one model to another that speed up the development process.

Still, none of these approaches deals with complex functionality, either on the client or on the server, which is needed in actual web environments. Closer to our approach, the use of UML for the modeling of web architectures[7] has also been proposed. The use of a standard UML notation, and the distinction between server side aspects and client side aspects are two of the main features of the approach. Also UIML [29], another XML-compliant markup language, centers on interface modeling, and provides an event mechanism to communicate with underlying logic modules. It, as OO-$\mathcal{H}$Method does, goes beyond the classical distinction between structure and layout templates, and proposes new orthogonal system views for functionality and interface widgets. Furthermore, the obtained interface description is both device and technology independent.

6 Conclusions

Actual interface-related languages are too target-environment dependent, and so lack the flexibility needed for the development of complex web applications. So 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 failure risks 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. The OO-$\mathcal{H}$Method can be seen as an extension of OO-Method to face the whole web product development process. In order to properly capture the particulars associated with the design of web interfaces, OO-$\mathcal{H}$Method adds several navigation and interface constructs to the OO-Method conceptual model, which 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, defined in an Interface Pattern Catalog. The NAD, 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 component in a default APD. The APD is based on the concept of templates, and its underlying structure is expressed in a 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. As oppose to other existing web methods, the approach presented in this paper does not intend to be 'yet another method' for web modeling but to extend a consolidated conceptual modeling approach.

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

1. The detailed presentation of OO-$\mathcal{H}$Method as an interface modeling approach that extends conventional methods departing from navigation user requirements.
2. The modeling of complex application behavior by means of explicit interaction with underlying logic modules.
3. 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.
4. The notion of Transformation Rule, associated with each one of the possible pattern implementations, as a way to simplify and systematize the modification process of the APD schema.
5. The use of a taxonomy of construct templates, defined in a standard notation, to build the different constituents (information, layout, input data, client functionality) of the APD as necessary.
6. The notion of Default APD, built by applying a set of mapping rules from the different elements of the NAD.

References

[1] C. Alexander. *The timeless Way of Building*. Oxford University Press, 1979.

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

[3] M. Bieber and C. Kacmar. Designing Hypertext Support for Computational Applications. *CACM: Communications of the ACM*, 38(8):99 – 107, 1998.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A System of Patterns*. Wiley, 1996.

[5] C. Cachero. The OO-$\mathcal{H}$Method Template Taxonomy. Technical report, Universidad de Alicante, 02 2000.

[6] S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites WWW9 Conference. In *First ICSE Workshop on Web Engineering, International Conference on Software Engineering*, 05 2000.

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

[8] eXtensible Markup Language (XML). http://www.w3.org/XML/.

[9] F. M. Fernández, D. Florescu, J. Kang, A. Levy, and D. Suciu. 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.

[10] M. Fernández, D. Suciu, and I. Tatarinov. Declarative Specification of Data-intensive Web sites. In *USENIX conference on Domain Specific Languages*, 05 1999.

[11] 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.

[12] M. Gaedke, F. Lyardet, and H. Werner-Gellersen. Hypermedia Patterns and Components for Building better Web Information Systems. In *Hypertext 99' Second Workshop on Hypermedia Development - Design Patterns*, 1999.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[14] H. Gellersen and M. Gaedke. Object-Oriented Web Application Development. *IEEE Internet Computing*, pages 60–68, January 1999.

[15] A. Ginige. Web Engineering: Methodologies for Developing Large and Maintainable Web-Based Information Systems. In *IEEE International Conference on Networking*, pages 89–92, 12 1998.

[16] J. Gómez, C. Cachero, and O. Pastor. Extending a Conceptual Modelling Approach to Web Application Design. In *CAiSE '00. 12th International Conference on Advanced Information Systems*, volume 1789, pages 79–93. Springer-Verlag. Lecture Notes in Computer Science, 06 2000.

[17] F. Manola. Technologies for a Web Object Model. *IEEE Internet Computing*, pages 38–47, January 1999.

[18] S. McGrath. *XML by Example. Building e-commerce Applications*. Prentice Hall, 1998.

[19] G. Mecca, P. Merialdo, and P. Atzeni. Araneus in the era of XML. *IEEE Data Engineering Bulletin*, 22(3):19–26, September 1999.

[20] G. Mecca, P. Merialdo, P. Atzeni, and V. Crescenzi. The ARANEUS Guide to Web-Site Development. Technical report, Universidad de Roma, 03 1999.

[21] S. Murugesan, Y. Deshpande, S. Hansen, and A. Ginige. Web Engineering: A New Discipline for Development of Web-based Systems. In *First ICSE Workshop on Web Engineering, International Conference on Software Engineering*, 05 1999.

[22] MyXML. http://www.infosys.tuwien.ac.at/myxml/.

[23] OMG Unified Modelling Language Specification. http://www.rational.com/uml/, June 1999.

[24] 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.

[25] 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.

[26] 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.

[27] D. Schwabe and R. Almeida Pontes. A Method-based Web Application Development Environment. In *Position Paper, Web Engineering Workshop, WWW8*, 1999.

[28] The source for java technology. http://java.sun.com.

[29] UIML. http://www.uiml.org.

[30] J. Warmer and A. Kleppe. *The Object Constraint Language. Precise Modeling with UML*. Addison-Wesley, 1998.