[1]An Object Oriented Approach to Web-Based Application Design

Daniel Schwabe* and Gustavo Rossi**
(*)Departamento de Informática. PUC-RIO, Brazil
E-mail: schwabe@inf.puc-rio.br
(**)LIFIA, Fac Cs. Exactas, UNLP, Argentina; CONICET; UNLM
E-mail: gustavo@sol.info.unlp.edu.ar

Abstract

In this paper we discuss the use of an object-oriented approach for web-based applications design, based on a method named Object-Oriented Hypermedia Design Method (OOHDM).

We first motivate our work discussing the problems encountered while designing large scale, dynamic web-based applications, which combine complex navigation patterns with sophisticated computational behavior. We argue that a method providing systematic guidance to design is needed. Next, we introduce OOHDM, describing its main activities, namely: conceptual design, navigational design, abstract interface design and implementation, and discuss how OOHDM designs can be implemented in the WWW. Finally, related work and future research in this area are further discussed.

1. Introduction. The problems of hypermedia design

The emergence of the World Wide Web has raised a new generation of information systems: those combining navigation through a heterogeneous information space with operations querying or affecting that information. The WWW provides a simple client-server architecture, and, most importantly, from the point of view of application design, it introduces the hypertext (or hypermedia) paradigm.

The first hypermedia applications, typically distributed in CD ROMs, were thought of as unchangeable applications that were not meant to be maintained or modified for new releases, possibly as a consequence of physical properties of their support media. The evolution of technology, most notably the phenomenal growth of the WWW, has given rise to applications that are constantly modified, that are enriched with new services, and new navigation and interface features are added according to the organization's marketing policy. As such, we argue that good web-based applications should be, first of all, be good hypermedia applications; furthermore, the WWW poses certain constraints that are reflected mostly in the interface and in the architecture of the implementation.

Traditional software engineering methodologies, or methodologies for developing information systems using databases, do not contain useful abstractions capable of easing the task of specifying applications that embody the hypertext metaphor. For example, they do not provide any notion of linking, and very little is said about how to incorporate hypertext into the interface. In addition, as the size, complexity and number of applications grows, a systematic approach is needed that helps dealing with complexity, and allows evolution and reuse of previously gathered design knowledge.

Producing applications in which the user may benefit from the hypertext paradigm for navigating through Web-sites, while performing complex transactions over its information base, is a difficult task [Gellersen97]. In [Bieber97], the authors point out that while many organizations will embrace the WWW, there is a risk of losing well-known advantages of hypertext in the web. Meanwhile, if we look carefully at new products supporting the development of Web-applications

To appear in Theory and Practice on Object Systems. Wiley and Sons, October 1998 we find that the suggested approach for building these applications is just designing them as if they were to run in a non-web environment. As exemplified by Parc-Place’s Visual Wave [VWave96] or Applied Reasoning’s Classic Blend [CBlend96]), only some aspects related with concurrent access to shared databases are taken into account. Navigation, which is a distinguishing feature of Web-based applications, is not considered at all.

What are the inherent complexities in this kind of software? First, navigation poses many problems, as repeatedly reported in the literature [Nielsen90]. A healthy navigational structure is one of the keys to success in hypermedia applications. When the user understands where he can go, and how he can reach a desired target point, he can benefit the most from the application. When navigation is combined with multimedia data, new problems appear [Hardman93]. For example, what happens when we navigate (in) to a node with an active multimedia presentation? In addition, what happens if we leave it?

Building the interface of a web application is also complex; not only do we need to specify which interface objects should be implemented, but also the way in which those interface objects will interact with the rest of the application. For example, we need to distinguish when an interface action causes navigation to another page, when it is just a local interface effect (for example activating a pop-up menu), etc.

Experience already gathered in building interactive applications shows that using object-oriented techniques is a key approach, especially in the case of large scale, evolvable applications. Many companies are now retrofitting existing interactive applications to the WWW (or to intranets using the WWW protocols), which entails adding hypertext functionality on top of existing application behavior. The combination of these functionalities is often difficult, since different design concerns must be accommodated.

As we discuss in section 2, modern design methodologies tend to neglect one of those dimensions: in general object-oriented design methods do not provide primitives for specifying navigation, while hypermedia design approaches emphasize structural aspects, ignoring computational behavior.

The approach presented in this paper is based on the Object-Oriented Hypermedia Design Method (OOHDM) [Schwabe 96a; Rossi 96d; Schwabe 95 a; Schwabe 95b]; we show that our method provides high level abstraction and composition mechanisms aimed at solving most of the previously mentioned problems. We first describe each of the OOHDM underlying design models and their design concerns. We discuss how OOHDM designs can be implemented in the WWW. Finally, we compare our work with others in the hypermedia and object-oriented design field and discuss further research in this area.

2. The Object-Oriented Hypermedia Design Method

There is a growing consensus about the kind of activities that must be performed with respect to the software product: modeling or analysis, design, implementation, testing and maintenance. This is true regardless of the different life-cycle models specifying the sequencing of processes and products involved in the development of an application (e.g. the spiral and waterfall model). In this respect, the process of building web-based (or more general hypermedia) applications is not intrinsically different from the one used when building conventional applications.

In the hypermedia domain there are conflicting requirements that must be satisfied in a unifying framework. On one hand, in the final application, navigation and functional behavior must be seamlessly integrated. On the other hand, during the design process we should be able to decouple design decisions related with the application's navigational structure from those related with the domain model itself. Since most implementation environments do not give full support to object-oriented concepts, our design models should easily translate into existing platforms.

According to OOHDM, the development of hypermedia applications occurs as a four activities process – Conceptual Design, Navigation Design, Abstract Interface Design, and

Implementation – that are performed in a mix of iterative and incremental styles of development; in each step a model is built or enriched.

The cornerstones of the OOHDM approach are

1. The notion that navigation objects are views, in the database sense, of conceptual objects;

2. The use of appropriate abstractions to organize the navigation space, with the introduction of navigational contexts;

3. The separation of interface issues from navigation issues;

4. An explicit identification that there are design decisions that need only be made at implementation time.

The starting point is the elaboration of a model of the application domain, which determines the universe of discourse. This is done during the Conceptual Design phase, using well known object-oriented modeling principles [Wirfs-Brock 90, Rumbaugh 91] augmented with some primitives such as attribute perspectives and sub-systems.

The Conceptual Model represents two kinds of objects: those that will be eventually perceived as nodes in the navigational model (called Entity Objects by Jacobson [Jacobson 92]); and those that provide computational support for the application, encapsulating behaviors such as algorithms, access to databases, etc. The resulting model may possible serve as a basis for many applications, and does not include any navigation specific information.

One essential distinguishing feature of hypermedia applications is the notion of navigation, in which the user of an application in this domain navigates in a space made out of objects. These objects are not the same as the conceptual objects, but rather objects customized to the user's profile and tasks. This customization is achieved using the view mechanism between objects, analogously to views in databases.

In this fashion, navigation object attributes are a "cut and paste" of possibly several different conceptual object attributes, which is indicated by the patterned boxes in Figure 1. Navigation objects may also have their own behavior, implementing functionalities beyond browsing and navigation, e.g., updates and computations in general. When a full O-O environment is used, nodes may be implemented as observers of conceptual objects.

Another step in structuring the Navigation space is provided by collecting the objects of the navigational space in meaningful sets called Navigational Contexts. There are several possible criteria for defining such collections of nodes, based on class attributes and relationships. During Navigational Design we also define the way in which navigation will proceed by specifying transformations in the navigational space, i.e. the set of accessible navigational objects at a given time.

Navigation objects are not directly perceived by the user; rather, they are accessed via interface objects. Accordingly, the Abstract Interface Design specifies interface objects that are responsible for mediating user interaction with navigation objects. The interface model specifies which interface objects the user will perceive; which interface objects will activate navigation; how multimedia interface objects will be synchronized; and the interface transformations that will take place.

Finally, the Implementation phase is responsible for mapping conceptual objects, navigation objects and interface objects onto the particular runtime environment being targeted. When the target implementation environment is not fully object-oriented, we have to map the conceptual, navigational and abstract interface objects into concrete objects, i.e. those available in the chosen implementation environment. This may involve defining HTML pages (or, for example, Toolbook objects in non Web-based environments), scripts in some language, queries to a relational database, etc; in this way the author produces the actual hypermedia application to be run.

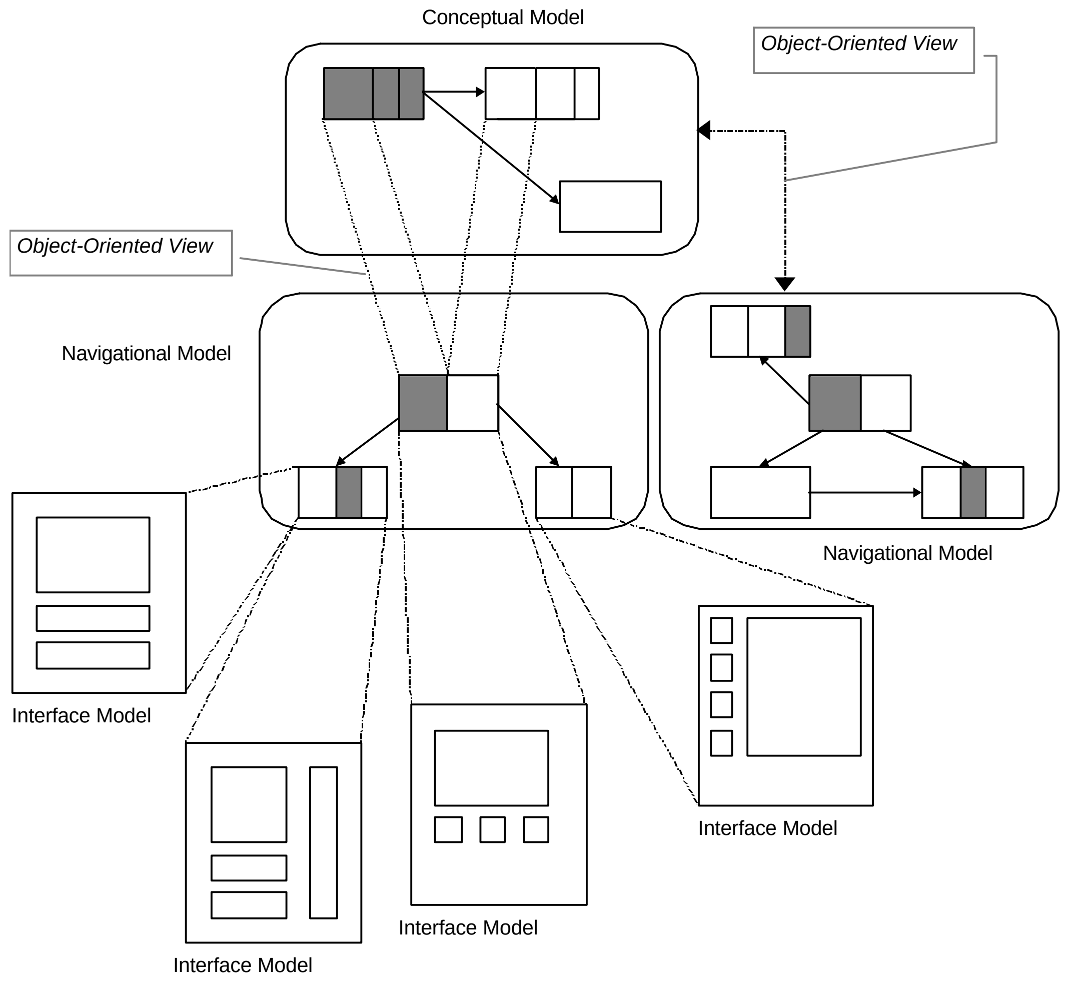


Figure 1 - Relation among Conceptual, Navigational and Interface Objects in OOHDM. Navigation classes are views over Conceptual Classes; Interface objects mediate interaction of Navigation objects with the outside world, including users. (Shaded boxes stand for Class Attributes).

In this work we will describe each activity in more detail, discussing with concrete examples the way in which each design formalism helps in gaining understanding of hypermedia, and in particular web-based, applications. We define how the designer proceeds from Conceptual modeling through Navigation design to Interface design and Implementation.

2.1 Conceptual Modeling

During this activity, we build a conceptual schema representing objects, their relationships and collaborations existing in the target domain. In "conventional" hypermedia applications, i.e. those in which the hypermedia components will not be modified during their execution, we could use a structural semantic model [Banerjee87]. However, when either the information base may change dynamically or we intend to perform complex computations or queries on the objects or the schema, we need a behaviorally richer object-oriented model.

In OOHDM, the conceptual schema is built upon classes, relationships and sub-systems. Classes are described as usual in object-oriented models, though attributes may be multi-typed representing different perspectives of the same real-world entity. We use a notation that is similar to UML [UML 97]. Class and Relationships Cards, similar to CRC cards [Wirfs-Brock 90] are used as a documentation aid, helping to trace design decisions forward and backwards. Different processes of object modeling and design are proposed and discussed by object-oriented methodologies and bibliography in the field. However, there are some modeling decisions appearing in any process that may impact on the navigational structure of hypermedia applications. In this paper, we will focus on design decisions affecting such structure.

We briefly describe our modeling primitives in the following paragraphs. As most of them are slight variations of their counterparts in object-oriented modeling and design methods, we do not elaborate them further. Instead, we will focus on those issues that affect the navigational design activity.

In OOHDM the class schema consists of a set of classes connected by relationships. Objects are instances of classes, and thus, when a relationship holds between classes, it abstracts corresponding object-to-object relationships. Classes will be used later during Navigational Design to derive Nodes, and Relationships will be used to derive Links.

Figure 2 contains a simple schema for an online magazine. In this model, there are stories, which can be essays, translations or interviews; an interview is an aggregation of questions and answers. Every story has an author, and an interview is also related to the person who grants the interview.

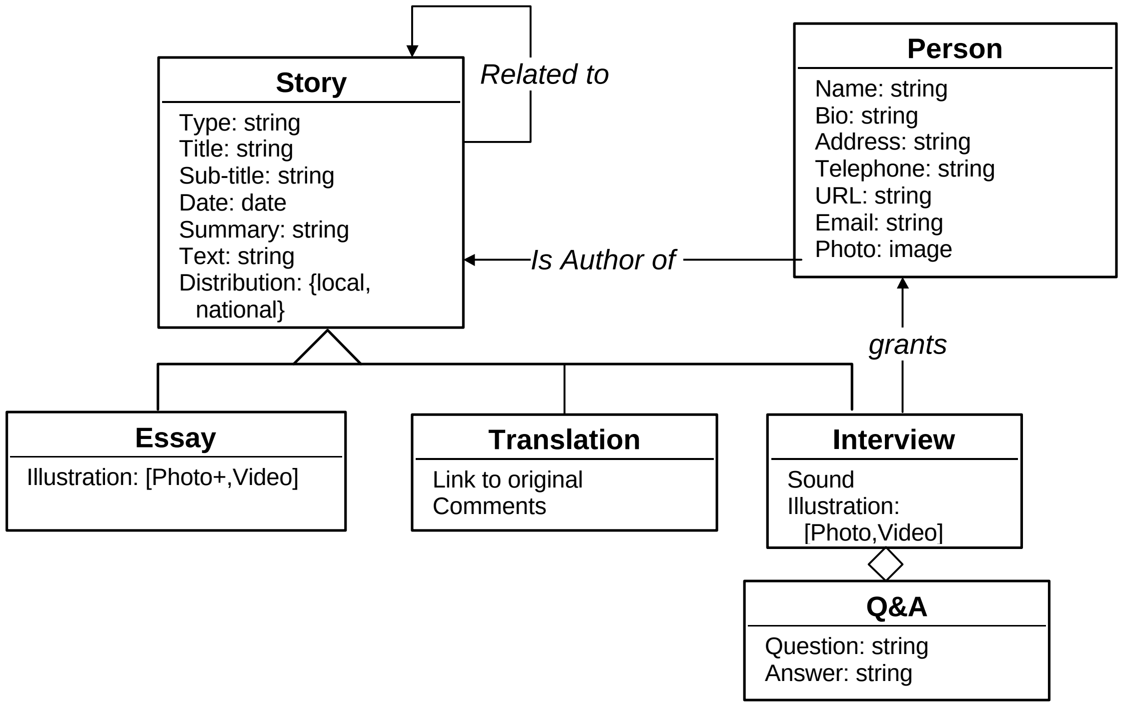


Figure 2: Conceptual Model for an online Magazine

It should be noted that there is no class “Magazine” in this schema; the magazine as such will be realized in the Navigational Design phase.

In this Class Schema, we have emphasized relationships over class behavior for simplicity. Class attributes are typed, and represent the object’s intrinsic properties (e.g., the story’s title, the author’s name, etc.). Each possible type is called a perspective, similar to HDM’s [Garzotto 93], exemplified in Figure 3: the attribute *Illustration* of an *Essay* may be either a *Photo* or a *Video*. When multiple perspectives exist, the notation "[p1, p2]" is used, and a “+” may be added next to a perspective to mark it as default, as is the case for *Photo* of *Illustration*. Only the default perspective must be present in all instances, whereas the others may or may not be implemented.

We will refine this specification during Navigational and Abstract Interface Design by defining which perspective is shown, when and how. Clearly, when we implement classes in the conceptual model, we need to specify one instance variable for each perspective.

Deciding whether to express a relationship as an attribute, a method or a combination of both is a design issue that has been largely discussed in the object community. In fact current object-oriented modeling approaches use relationships as “first-class citizens”, thus relegating the discussion to the design and implementation step. As an alternative, behavioral approaches such as Wirfs-Brock’s responsibility driven design [Wirfs-Brock 90] favor using collaboration diagrams instead of relationships in the style of UML.

From the hypermedia design point of view relationships express an important aspect of the application domain and they should not be hidden in class attributes (at least until we need to implement those classes). This means that a relationship should be specified whenever an attribute represents a complex conceptual entity intended to be explored in the final hypermedia. In fact, when implementing web applications using an object-oriented architecture, providing access to relationships will be a responsibility of the source class and will be implemented as a method, perhaps accessing an instance variable. In [Lange94], an extension to the OMT class diagram [Rumbaugh91] is presented to enhance relationships.

In our design approach, we provide three abstraction constructs for dealing with complexity: Aggregation, Generalization / Specialization and a packaging concept, Sub-Systems. The first one is useful for describing complex classes as aggregation of simpler ones and the second for building Class Hierarchies and using inheritance as a sharing mechanism. Sub-Systems are a packaging mechanism for abstracting entire domain models.

Part-of relationships (Book chapters, Car parts) are described using aggregation relationships. Aggregates in a class definition are similar to components in HDM [Garzotto 93]. Implicit relationships exist between a complex object and its parts (and vice versa) and between part themselves. These are called structural links in HDM. In the example in Figure 4, an *Interview* is composed of *Q&A*s (question-answer pairs).

Choosing to use aggregate objects, complex attributes or relationships is a modeling decision that may vary from application to application. Given that aggregation is an important structural relationship, we suggest decomposing an object into parts each time these parts are supposed to be explored in the hypermedia application as non-atomic ones. Existing heuristics in the object-oriented modeling domain may be used to identify aggregation structures.

We use the same inheritance semantics as in [Rumbaugh 91] i.e. classes inherit attributes, part-structure, relationships and behavior of their super-classes. In Figure 5, a *Story* may be either an *Essay* or a *Translation* or an *Interview*.

Using a behavioral object-oriented model for describing different aspects of a hypermedia application allows expressing a rich variety of computing activities such as dynamic queries to an object base, on-line object modifications, heuristics-based searches, etc. The kind of behavior required in the conceptual model depends upon the desired features of the application. For many web applications, in particular those implementing plain browsing (i.e. read-only) functionality, class behavior beyond linking functionality may seem unnecessary. In this case, the behavior amounts to accessing a multimedia information base by navigating over relationships, and could be “built-in” into the model itself.

In contrast, in applications in which the underlying information base may change dynamically as the effect of user actions, or when the hypermedia network is just a component of a larger corporate application, we may need to define methods that implement this behavior in conceptual classes. During Navigational and Interface design, we will specify the way in which interface objects trigger behavior both in the navigational and conceptual models.

Since OOHDM is a design method and not an implementation framework, we shall assume that methods for getting the value of attributes of an object are automatically generated. When other computations must be done, corresponding methods should be specified. Using the terminology of object-oriented methods, we may say that the Conceptual Model will contain features of both Analysis and Design Models. When the application will run on top of an environment supporting (distributed) objects, classes in the conceptual model will be implemented directly as they are defined, specifying multiple perspectives as separate attributes. If not, they will serve as design specifications for the navigational and interface design activities.

2.2 Navigational Design

The first generation of web applications was intended to perform navigation through an information space using a single hypermedia data model. The simplicity of HTML as the implementation language and the lack of a hypermedia design culture resulted in web sites in which users experienced cognitive overhead and dis-orientation while navigating the web. To make matters worse, there are no good orientation tools for existing browsers, and most existing tools (such as MAPA, http://www.dynamicdiagrams.com) provide a map of the "node and link" level, which quickly becomes unwieldy. In addition, such maps are not well defined in the case of dynamically generated pages.

In spite of these problems being well known in the hypermedia community (see for example [Nielsen 90]), they have seldom been taken into account by web-site designers. Most of the design effort has been usually placed in user interface aspects, and the navigation structure is built on simple hierarchies. Now that web browsers are the interface, and sometimes the host, for different kinds of applications, there is a risk that navigation be considered just another kind of application behavior.

In OOHDM, navigation is considered a critical step in the design of a hypermedia application. A Navigational Model is built as a *view* over a conceptual model, thus allowing the construction of different models according to different users profiles. Each navigational model provides a "subjective" view of the conceptual model [Harrison 93]. While designing the navigational structure of a web application, we will take into account several aspects such as:

- Which objects will be navigated, which attributes they possess, and what are the relationships among these objects and the ones defined in the conceptual schema? We will do this by defining nodes and links as object-oriented views of conceptual objects and relationships.

- What kind of composition structures exist among navigational objects and how are they related?

- What is the underlying structure of navigation? In which contexts will the user navigate? We will introduce the concept of navigational contexts, an architectural primitive for organizing the navigation space.

- We need to decide whether navigated objects may look different according to the context in which they are visited, and we must clearly specify those differences. We shall use InContext classes to "decorate" navigational objects.

- Which connections and access structures exist among objects that will be navigated (links, paths, indices, guided-tours, etc.)?

- How does navigation proceed when the user "jumps" from one object to another, i.e., what is the effect of navigation on the source and target object and possibly on other related objects as well?

Navigation design is expressed in two schemas, the Navigational Class schema and the Navigational Context schema. The navigable objects of a hypermedia application are defined by a navigational class schema, whose classes reflect the chosen view over the application domain. In OOHDM, similarly to HDM [Garzotto 93] and RMD [Isakowitz 95], there is a set of pre-defined types of navigational classes: nodes, links, and access structures. The semantics of nodes and links are the usual in hypermedia applications, and access structures, such as indices and guided tours, represent possible ways of accessing nodes.

The Navigational Transformations specification describes the dynamics of the application, showing the way the navigational space changes when the user navigates, i.e. which nodes are activated and which are deactivated when a link is followed. The default navigational semantics in OOHDM is that when a link is followed, the source node is deactivated and the target node activated. This interpretation is the default one commonly found in Web browsers. Since navigational dynamics that are more complex are hard or infeasible for implementation in the web, we will not discuss the primitives of the Navigational Transformations schema further. The reader can refer to [Meré 96, Rossi 96c] to see a temporal logic based specification of navigational transformations and [Rossi 96d] where a variant of Harel's Statecharts and Coleman's Objectcharts have been used.

In order to define nodes as object-oriented views of conceptual classes defined during conceptual design, we use a query language similar to the one in [Kim 94]. We allow a node to be defined by combining attributes of different related classes in the conceptual schema. Nodes possess single typed attributes, link anchors, and may be atomic or composite, as in [Gronbaek 94]. Anchors are instances of Class Anchor (or one of its sub-classes) and are parameterized with the type of Link they host. The standard navigational behavior when a node receives the message "anchorSelected ()" is to delegate the message to the corresponding anchor, which in turn activates the corresponding link. In this way it should be possible to re-define the standard Node or Anchor behavior so as to perform checks before navigation proceeds.

In an analogous way, links reflect relationships intended to be explored by the final user and are also defined as views on relationships in the conceptual schema.

Links connect navigational objects and may be one-to-one or one-to-many. The result of traversing a link is expressed by either defining the navigational semantics procedurally as a result of the link's behavior or by using an object-oriented state transition machine similar to Statecharts [Rossi 96d]. Access structures (such as indices or guided tours) are also defined as classes and present alternative ways for navigation in the hypermedia application. In object-oriented architectural terms, the relationships among nodes and conceptual objects, and among links and relationships in the schema, are expressed as instances of the Observer design pattern (see "Observer" pattern in [Gamma 95]). The general syntax for defining a node's attributes is shown below (the syntax for links is similar).

```
NODE name [FROM className: varName] [INHERITS FROM nodeClass]
attri:  type1   [SELECT  name1] [FROM class1:varName1, classj: varNamej
        WHERE logical expression]
attr2:  type2   [SELECT  name2]...
...,
attrn:  typen [idem]
END
```

Where

- *name* is the name of the class of nodes we are creating.
- *className* is the name of a Conceptual Class (from which the node is being mapped).
- *nodeClass* is the name of the super-class
- *attri* are the names of attributes for that class, typei the attribute's types.
- *namei* are the subjects for the query expression and vari are mute variables used to express logical conditions.
- -*logical expression* allows defining classes whose instances are a combination of objects defined in the conceptual schema when certain conditions on their attributes and/or relationships hold.

As an example, consider the online magazine example in Figure 2. With this simple syntax we can define, for example, a Node Class *Story*, including as one of its attributes the *name* and *bio* of the *Person* who wrote it and an anchor of a link to that *Person*. Note that, in the

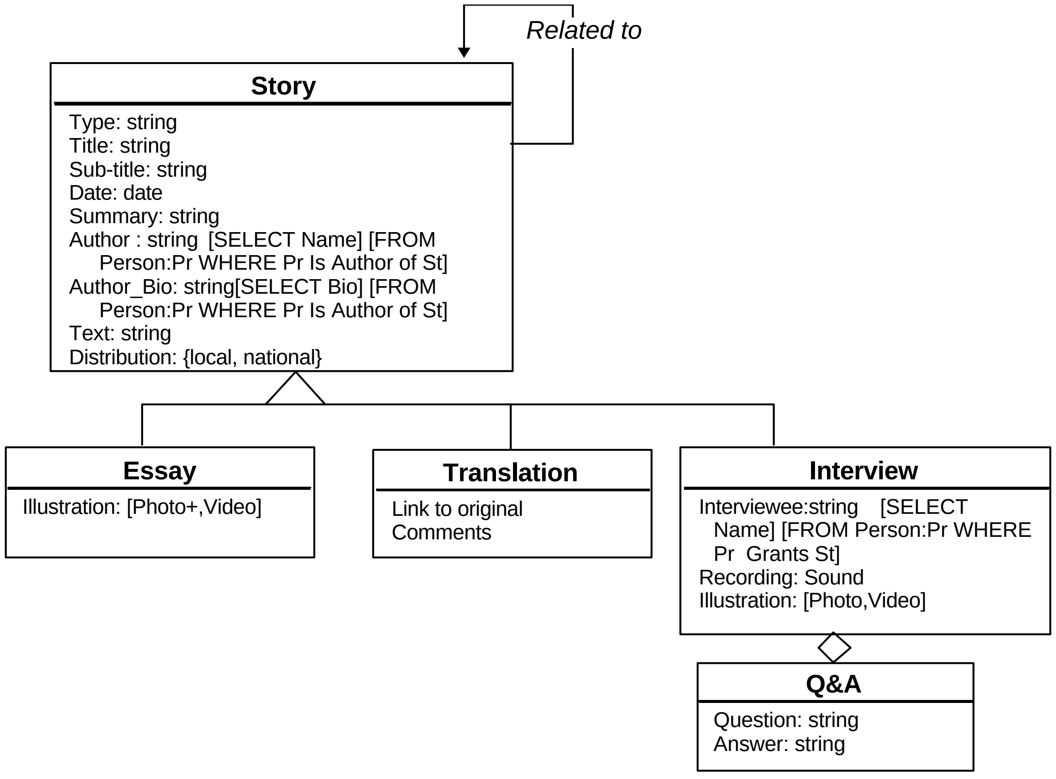
Conceptual model, the *Person's name* and *bio* are attributes of Class *Person*. Considering Nodes and Links as views on the Conceptual Classes allows one to create navigational objects opportunistically, according to user profiles and tasks, without modifying the conceptual objects.

The example above will result in the following Node definition:

```
NODE Story [FROM Story:St] [INHERITS FROM Person]
author: String   [SELECT Name] [FROM Person:Pr WHERE Pr Is Author of St]
auhtor_bio: String  [SELECT Bio] [FROM Person:Pr WHERE Pr Is Author of St]
....  (other attributes "preserved" from the conceptual class Story}
toAuthor: Anchor (Is Author of)
END
```

Note that the value of the *toAuthor* attribute is an anchor that is parameterized with the Link class *Is Author of*. When defining the interface appearance of Node Class *Story* we can, for example, make the anchor appear as an invisible button on top of the name of the author (attribute *author*). Though it may seem that both attributes have the same behavior, only the anchor acts in response to an interface event.

Figure 3 contains a navigational schema for the online magazine. Notice that in this application it was decided that *Persons* (cf. Figure 2) should not appear, and therefore the relevant information was included in class *Story*, as exemplified in the previous paragraph. The same has been done to the *Person* who granted an *Interview*, which is a sub-class of *Story*.



Related to
Story
Type: string
Title: string
Sub-title: string
Date: date
Summary: string
Author : string [SELECT Name] [FROM Person:Pr WHERE Pr Is Author of St]
Author_Bio: string[SELECT Bio] [FROM Person:Pr WHERE Pr Is Author of St]
Text: string
Distribution: {local, national}
Essay
Illustration: [Photo+,Video]
Translation
Link to original
Comments
Interview
Interviewee:string [SELECT Name] [FROM Person:Pr WHERE Pr Grants St]
Recording: Sound
Illustration: [Photo,Video]
Q&A
Question: string
Answer: string

Figure 3 - Navigational Schema of the online Magazine. Notice that class Person does not appear; the author's attributes are now part of the Story. The same happens to the Person who grants an Interview.

Well-designed hypermedia applications must take into account the way in which the user explores the hypermedia space. Redundant information should be judiciously used and we must be able to help the user to choose the way in which he navigates in a consistent and controlled way. Unfortunately, nodes and links are not enough to fulfill this goal, even if we allow compositions as in the Dexter model [Gronbaek 94]. Though the usual solution to this problem is to implement orientation tools, we also think that higher level architectural navigation primitives must be used – this is the point where navigational contexts appear.

In OOHDM, the main structuring primitive of the navigational space is the notion of navigational context (see [Schwabe94, Schwabe 95b, Barbosa 95] for a more extensive discussion). A navigational context is a set of nodes, links, context classes and other (nested) navigational contexts. It may be defined intensionally or extensionally, by either defining a property that all nodes and links in the context possess, or by enumerating its members. The definition of a context also includes a traversal order of its elements, and the existence or not of associated access structures.

A context can be defined in six basic ways:

1. Simple class based – Objects in this kind of context belong to the same class C and are selected by giving a property P that must be satisfied by all elements: Context = $\{e \mid P(e),\ e \in C\}$. A common case is when it includes all instances of a class (P is identically true) – e.g., all stories.

2. Class based group - Is a set of contexts, each of which is a simple class based context. It is specified by giving a parameterized property and letting the parameter assume all possible values (in a finite enumerable domain). For example, *Stories by type* is a group of contexts; its elements are simple class based contexts, one for each possible value of *type*, containing stories whose “Type” attribute equals *type*. Group = $\{\text{Context}_{\text{theme}}\}$, $\text{Context}_{\text{theme}} = \{p \mid p.type=type,\ p \in Story\}$.

3. Link based – Objects in this kind of context are of the same class and are selected when they belong to a 1-to-n relationship. For example, “all Stories by Bob Woodward”. Context = $\{p \mid Is\ Author\ Of(Bob\ Woodward, p), p \in Story$. Note that a particular case of this type is the context formed by all elements that are part of a composite object.

4. Link based group - Is a set of contexts, each of which is a link-based context. It is specified by giving a 1-to-n relationship and forming the link based contexts for each possible value of the source of the relationship. For example, “Stories by Author”. Group = $\{\text{Context}_{\text{Author}}\}$, $\text{Context}_{\text{Author}} = \{p \mid Is\ Author\ Of(Author, p), p \in Story\}$.

5. Enumerated - In this kind of context, elements are explicitly enumerated, and may belong to different classes. A typical example is a Guided Tour.

6. Dynamic - In this kind of context elements are inserted during navigation. An example of this type of context is the “history” maintained by many browsers. Another example is a “shopping basket”, which the reader builds while browsing through objects (for example, books) in other contexts.

Navigational contexts play a similar role as collections [Garzotto 94] and have been inspired by the concept of nested contexts [Casanova 91].

Navigational contexts organize the navigational space in consistent sets that can be traversed following a particular order; they should be defined in such a way as to help the user to perform his intended task.

In Figure 4 we show the Navigational Contexts Schema for the online magazine; notice that the magazine (issue) is formed by stories that are grouped according to several different criteria.

We briefly explain the notation used in the diagram. Indices are indicated by boxes with bold, dashed lines, such as the “Main Menu”. The simplest type of context (sets of nodes) is denoted by

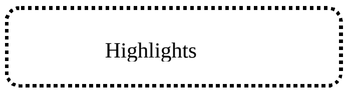


Figure 4: Notation for highlights

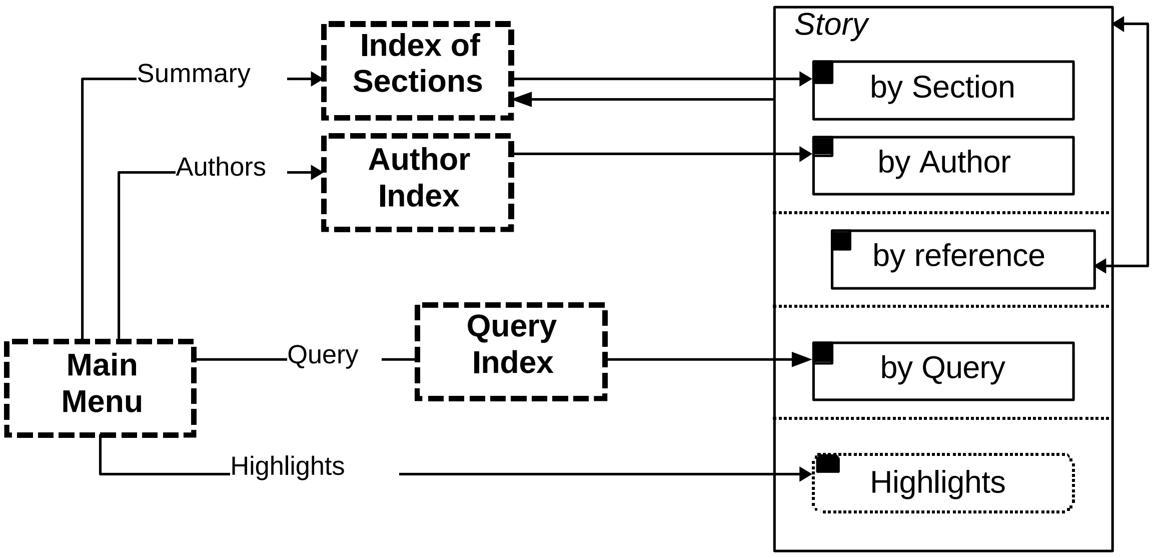


Figure 5 - Navigational Context Schema for the online magazine application

This indicates, in this example, a set of stories that are featured as “highlights” of a given issue.

In many situations, there will be groups of contexts that encompass nodes of the same class, according to different criteria. For example, Stories may be organized according to the different sections or authors. We denote this by

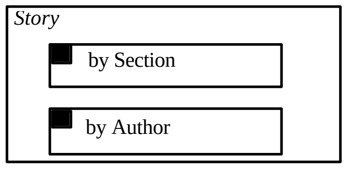


Figure 6: Notation for Story

In this case, we have two groups of contexts – one made out of contexts that gather stories for each section and one that gather stories by each author. The black box on the top left corner indicates that the group possesses an index to its components. The definition of the context itself will specify the types of navigation allowed inside the context; typical values are “sequential”, “circular sequential”, “index” (where it is possible to navigate only from an index to an element and back), or “index sequential”.

The absence of a dashed line between the groups indicates that it is possible to switch automatically from a context in one group to a context in the other group. This is not the case for context groups "by reference", "by query", and for the simple context "highlights" , which are all separated by dashed lines in Figure . This means, for example, that if the reader is looking at a story in a given section, he will be allowed to navigate to either "the next story in this section" or to "the next story by the same author". However, he would not be allowed to navigate "to the next highlight" (which would not even make sense!).

The arrow going from "Story" back to itself indicates that stories may refer to other related stories. If the reader is looking at a story, and then follows a link to a related story, he will reach the context in which all related stories are grouped. At this point, he must go back either to the original story, as indicated by the double arrow, or to the summary index, as indicated by the arrow from "Story" to the "Index of Sections".

Nodes are enriched with a set of special classes (we call them InContext classes) that allow a node to look different and present different attributes (including anchors), as well as methods (behavior) when navigated within a particular context (see Figure 5). For example, when traversing "Stories by Bob Woodward", the author's bio might not be included as an attribute of the story, whereas it might be included when traversing "Highlights" – see Figure 6. This enrichment follows closely the structure of the Decorator Design Pattern [Gamma95].

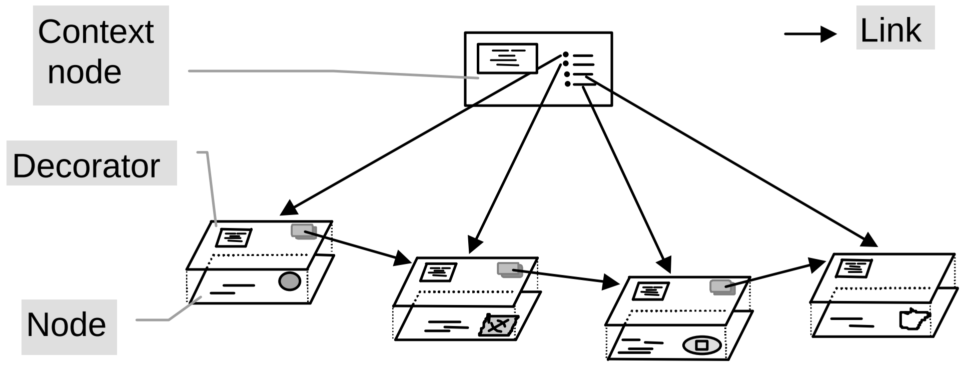


Figure 7 -A node in a context is constructed from a basic node and a decorator.

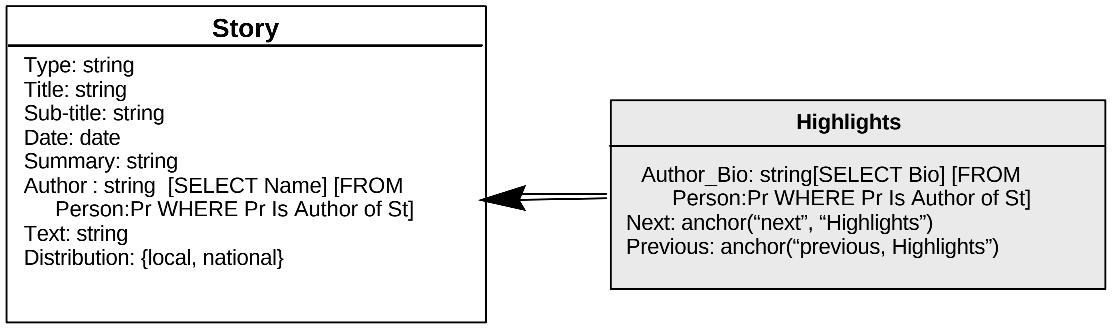

Story
Type: string
Title: string
Sub-title: string
Date: date
Summary: string
Author : string [SELECT Name] [FROM Person:Pr WHERE Pr Is Author of St]
Text: string
Distribution: {local, national}
Highlights
Author_Bio: string[SELECT Bio] [FROM Person:Pr WHERE Pr Is Author of St]
Next: anchor("next", "Highlights")
Previous: anchor("previous, Highlights")

Figure 8 Specification of InContext class for "Story" within "Highlights" contexts. In this case, the author's bio is only seen when a story is being accessed within the "Highlights" context.

Notice that the "previous" and "next" links are also attributes that belong to the InContext class for "Story" in the "Highlights" context. Other contexts, if they have sequential navigation, would add their own separate "next" and "previous" anchors.

2.3 Abstract Interface Design

Once the application’s navigational structure has been defined, we must specify its interface aspects. This means defining the way in which different navigational objects will appear, which interface objects will activate navigation and other application functionality, and which interface transformations will take place and when.

A clean separation between both concerns, navigational and abstract interface design, allows building different interfaces for the same navigational model, leading to a higher degree of independence from user-interface technology. In addition, this separation allows a better understanding of the overall application structure by clearly indicating which transformations in the interface are also navigational transformations. It also indicates which are simply local interface transformations that do not affect the state of the navigation and therefore do not require accessing the web server.

Though it has been argued that the user interface aspect of interactive applications (in particular web applications) is a critical component, modern methodologies tend to neglect this aspect. They relegate the specification of the interface to implementation-dependent tools, and therefore design decisions at this level are seldom documented. Moreover, as implementing the interface of Web applications is usually done by means of specialized HTML editors, many critical aspects of the interface may be ignored.

In OOHDM, we use the Abstract Data View design approach for describing the user interface of a hypermedia application [Cowan 95]. ADVs are objects in that they have a state and an interface, where the interface can be exercised through messages (in particular, external events generated by the user). ADVs are abstract in that they only represent the interface and the state, and not the implementation. ADVs have been used to represent interfaces between two different media such as a user, a network or a device (a timer, for example) or as an interface between two or more Abstract Data Objects (ADOs). ADOs are objects that do not support external, user-generated events [Cowan 95]. From an architectural point of view, ADVs are observers for ADOs, so the communication protocol among interface and application objects follows the rules described in the Observer Design Pattern [Gamma 95].

An ADV used in the design of web applications can be viewed as an interface object. It comprises a set of attributes (and nested interface objects) which define its perception properties, and the set of events it can handle, such as user-generated events. Examples of user-generated events are MouseClick, MouseDoubleClick, MouseOn, etc.. ADVs can be easily implemented in object-oriented environments for the Web (such as VisualWave, Classic Blend or Java’s ones) or may be translated to HTML documents.

Attribute values may be defined as constants, thereby defining particular styles of appearance such as position, color, or sound. The ADV interface model allows treating these features in an abstract way, relegating them to the implementation step. In general, the ADVs specify the organization and behavior of the interface, but the actual physical appearance or the attributes, and layout of the ADV on screen real estate is done in the Implementation phase.

A reserved variable, "perceptionContext", is used to indicate modifications to the perception space (i.e. the set of perceivable objects at a given moment). When we want to make some object perceivable we add it to perceptionContext, and elements removed from perceptionContext are no longer perceivable.

In the context of OOHDM, navigational objects such as nodes, and indexes will act as ADOs, and their associated ADVs will be used for specifying their appearance to the user. We will use the term ADV for referring to interface classes and objects. When necessary we will talk about ADV classes.

Different abstraction and composition mechanisms are used in the ADV design approach; first ADVs may be composed by aggregation or composition of lower-level ADVs, thus allowing the construction of user-interfaces with nested perceivable objects. Suppose, for example, that we have an application about paintings. **¡Error!No se encuentra el origen de la referencia.**7.a shows how an ADV describing a painting could be made out of three ADVs, containing an image, text, and a button. Furthermore, ADVs may be organized in generalization/specialization hierarchies that provide a powerful conceptual framework for defining hierarchies of interface objects. In **¡Error!No se encuentra el origen de la referencia.**7.b, we show the use of inheritance in ADVs.

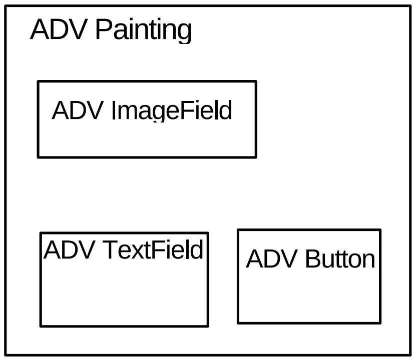


Figure 9: ADVs can be aggregations

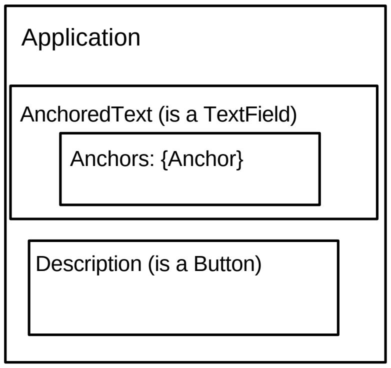


Figure 10: ADVs use inheritance

In **¡Error!No se encuentra el origen de la referencia.**7.b, AnchoredText is an Interface Object that adds a set of anchors (implemented for example as hotwords) to the more general TextField. Meanwhile Description is a specialized Button by adding a more customized behavior (not shown in the Figure). When we implement our Web application using an environment supporting certain particular kinds of interface objects, we may use them as the primitive ADVs for producing our design specification.

In summary, ADVs, allow us to express:

- the way in which interface objects are structured using aggregation and generalization/specialization as abstraction mechanisms. ADVs express the static layout structure that implements the interface metaphor [Hannemann 93]. ADVs allow defining the interface appearance of navigational objects and other useful interface objects (such as menu bars, buttons and menus).
- the way in which they are statically related with navigation objects. We use Configuration Diagrams [Coleman 92] as a diagrammatic tool for expressing these relationships.
- how they behave when reacting to external events; in particular how they trigger navigation and which interface transformations occur when the user interacts with the application. We use ADV-Charts [Carneiro 94], a derivative of Statecharts, that adds both structural and behavioral nesting and a Petri-Net like notation for expressing synchronization issues typical of multimedia data. For reasons of space, will not elaborate ADV-Charts in this paper.

The modeling constructs we use during navigational and abstract interface design are very similar – in fact, we use classes and objects with a formal connection model. Consequently, we obtain a seamless transition between both activities, allowing incremental construction of the navigational and abstract interface models. At the same time, relevant design decisions are recorded using a notation that is powerful and concise. In Figure 8 we show an ADV corresponding to the design of the Portinari web site (http://www.portinari.org.br/) a hypermedia application containing part of the work and documents related to Candido Portinari, a famous Brazilian painter (see also Figure 9).

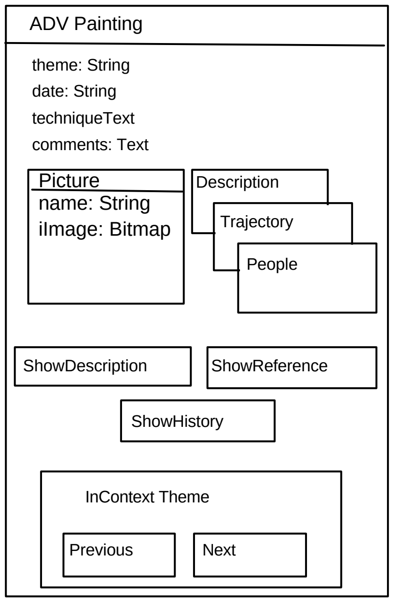


Figure 11: ADV Painting in Portinari Web Site

ADV Painting contains some attributes describing certain aspects of the painting and many nested ADVs such as Picture, Description, Trajectory and People. In the notation of Figure 9 Description, Trajectory and People are not intended to be shown at the same time and so their ADVs are overlapped. ADVs ShowDescription, ShowReference and ShowHistory are active controls allowing to show one of the previously mentioned ADVs. ADV InContext_Theme implements the interface of the InContext Class Theme and provides navigation controls inside the Navigational Context: Paintings of a Theme. Similar ADVs must be specified for other Navigational Contexts such as Paintings of a Technique.

While the above diagram shows the static nature of Painting's interfaces, the dynamics are described using ADV-charts. In a slight variation of Statecharts, we specify that when ShowDescription is clicked, it sends the message display to Description and the same occurs for ShowReference and ShowHistory. Note that this is a pure interface effect not involving navigating to another node. Meanwhile, when the Previous interface object is clicked, it sends the message anchorSelected (previous) to the corresponding ADO, in this case an InContext Object that communicates with the corresponding anchor and thus we navigate to another painting. Even when the implementation is not object-oriented, this communication model is easy to implement in most platforms, as discussed in section 2.4.

To end this section we show in Figure 9 the real interface of Portinari web site and how abstract interface objects are related with their implemented counterparts.

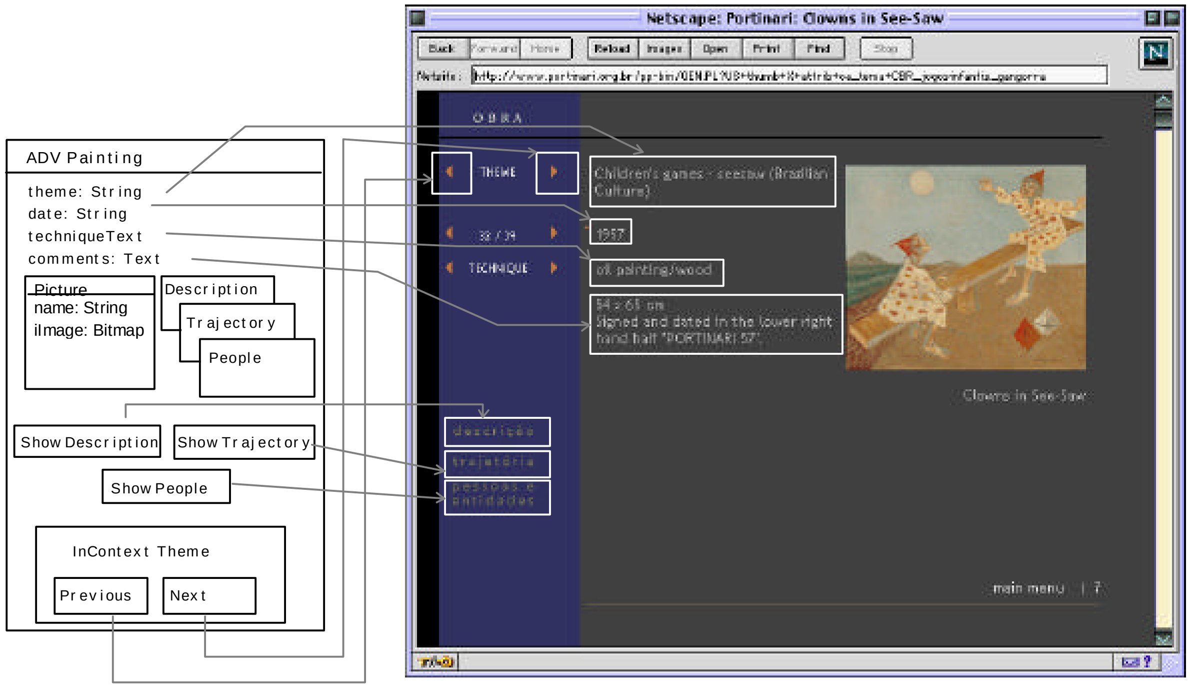


Figure 12: ADVs and their relationship with "real" interface objects

2.4 *Implementation*

In this phase, the designer will actually implement the design. Up to now, all models were deliberately constructed in such a way as to be independent of the implementation platform; in this phase the particular runtime environment is taken into account. We will concentrate on how OOHDM designs can be implemented in the WWW, taking care not to fix a single alternative, since there are many possible approaches through which this can be achieved.

When the implementation phase is reached, the designer has already defined the information items that are part of the problem domain. He also has identified how these items should be organized according to the user's profile and tasks; he has decided what the interface will look like, and how it will behave. In order to implement all of this in the WWW environment, the designer has to decide how the information items (both conceptual and navigation objects) will be stored. He must also decide how the interface appearance and behavior will be realized using HTML and possibly use some extensions. Notice that, in general, the actual appearance will be defined by a graphics design professional that should be part of the design team.

Although OOHDM is cast in terms of OO models, it does not require an OO implementation environment; an implementation based on an OODMBS (O2) (but not on the web) is described in [Milet 96]; Java-based implementations are under development. Section 3 contains a brief description of other implementation approaches.

2.4.1 Mapping Information Items

The information items (which correspond to the ADOs in the Abstract Interface Model) may be stored in files or in a database. Due to the nature and complexity of the types of applications for which OOHDM is most suited, we strongly recommend using a database to store the Conceptual and Navigation objects. Since the majority of DBMSs available on the market today are relational, a mapping of the OO models onto equivalent relational models must be made. There are several techniques and heuristics for doing this – see for example, [Keller 97]. The methods associated with the classes are implemented as a set of procedures that access the database to perform their computations.

To illustrate the type of design decisions, we will briefly discuss one mapping alternative. Each class in the OO model to be implemented is mapped onto a table, where each column stores an attribute, and each row corresponds to an object of that class. A distinguished attribute may be used as a database key, or an internal identifier, which corresponds to an object handle, can be generated and used as a key.

For attributes whose type is not supported directly in the DBMS (e.g. multimedia data), a separate auxiliary table must be implemented, and the object's attribute stores the Id of a row in the corresponding auxiliary table. Alternatively, it stores the name of a file in the operating system, which contains the actual value. Both alternatives have shortcomings; for instance, the first requires extra joins, and the second is vulnerable to changes outside the control of the DBMS. Unfortunately, this can only be avoided if the DBMS offers support for complex data types, as is becoming more common in the latest generation of products reaching the market.

In section 2.2 it was stated that the Navigation Model is a view over the Conceptual model. The designer has the option of reflecting this organization in the databases corresponding to each model. In other words, he may define the database containing the Navigation objects (nodes, links, etc...) as a view, supported by the DBMS, of the database corresponding to the Conceptual model. In the case where the DBMS does not directly support the view mechanism, or for efficiency reasons, the designer has the option of computing the view by hand. In this case, he will only implement the Navigation model, since it is the one the user will be accessing. Evidently, this alternative has shortcomings in terms of schema evolution, which become evident when the same Conceptual database is used as the basis for several applications. This is the case, for example, when companies have sites in their intranets, and part of these sites is visible (usually with a different interface) in the WWW.

In addition to mapping class definitions into whatever database model (relational, OO, etc...) being used, it is also necessary to implement "InContext" classes, which function as decorators for objects within particular contexts. Typically, this entails enriching the data model used in the database to account for the added attributes, and defining control functions that make these attributes accessible in the appropriate contexts. If the implementation is based directly on the file system, these control functions will access additional files containing the contextual information.

Once the databases are defined, they must be integrated in the WWW environment. There are many ways in which this can be done [Hunter 95, Varela 95], and we will not elaborate this further; it suffices to say that any of these techniques may be employed. In this respect, the criteria for choosing the integration method are the same as other applications, as discussed in the literature.

2.4.2 Implementing Contexts

Whereas the mapping of ADO's into implementation objects is somewhat obvious, the implementation of contexts is more complicated. The supporting database model or set of files must also contain the context definitions. With the exception of arbitrary contexts (whose specification is essentially an enumeration of its members), other types of contexts include a query or function specification that must be evaluated to compute the members of the context.

Navigation operations within contexts require keeping state information. For example, to determine "what is the next story by this author" requires knowing which story the user is currently looking at, which stories make up the referenced context ("Stories by author"), and what is the ordering defined for that context.

In terms of the WWW, this means that either this state information is kept within the database or file structure being used, or special control information is kept on the side to represent the navigation state. In this case, any of the better known techniques for keeping state in the WWW may be employed: passing state information within URLs, from page to page; keeping state information in hidden fields passed on from page to page; or using cookies. All of these techniques require using CGI scripts to implement navigation, which poses no additional requirements since CGI scripts are most likely already being used for database integration.

A different technique that has also been employed is to represent ADOs using frames and Javascript. The ADVs are mapped onto framesets, and the root frame contains variables that store both instance data for the ADOs, and context information, as well as scripts to manipulate them. These scripts also implement navigation operations within documents, which are stored in other frames in the frameset. The advantage of this approach is that state maintenance is done entirely within the client machine; the disadvantage is that it breaks down for large systems as one reaches the limits in size for Javascript programs. It should be noted that this technique to distribute computation to clients might also be used in combination with the previously mentioned ones.

2.4.3 Implementation of the Interface

The actual interface organization and behavior is specified in the ADVs, and the physical layout and appearance must be defined in this phase. Client interfaces in the WWW may be implemented using several alternatives – plain HTML, HTML with Javascript, HTML with plugins, HTML with Java, pure Java, etc... – but we will concentrate on the first two.

The implementation of ADVs requires defining page layouts in HTML that are consistent with the ADV specifications. In those cases when the values of instance variables are computed at runtime, pages must be generated dynamically, based on HTML templates previously specified by the designer. These templates usually contain a mixture of HTML code and calls to functions in some scripting language. The execution of these functions will retrieve or compute the instance data to fill in the missing data that make up the final HTML page.

We have designed and implemented an environment based on the scripting language Lua [Ierusalimschy 96] and on the CGILua environment [Hester 97] called OOHDM-Web [Pontes97]. This environment implements templates that are a mixture of plain HTML and calls to functions in a library giving access to the Navigation objects. These objects are stored in a relational database accessed via ODBC using an approach similar to what has been described in section 2.4.1. Figure 10 contains a schematic representation of the architecture of the OOHDM-Web environment. Other commercial examples of the use of templates are Cold Fusion [http://www.allaire.com] and StoryServer [http://www.vignette.com]; however, they are not integrated with any methodology or model.

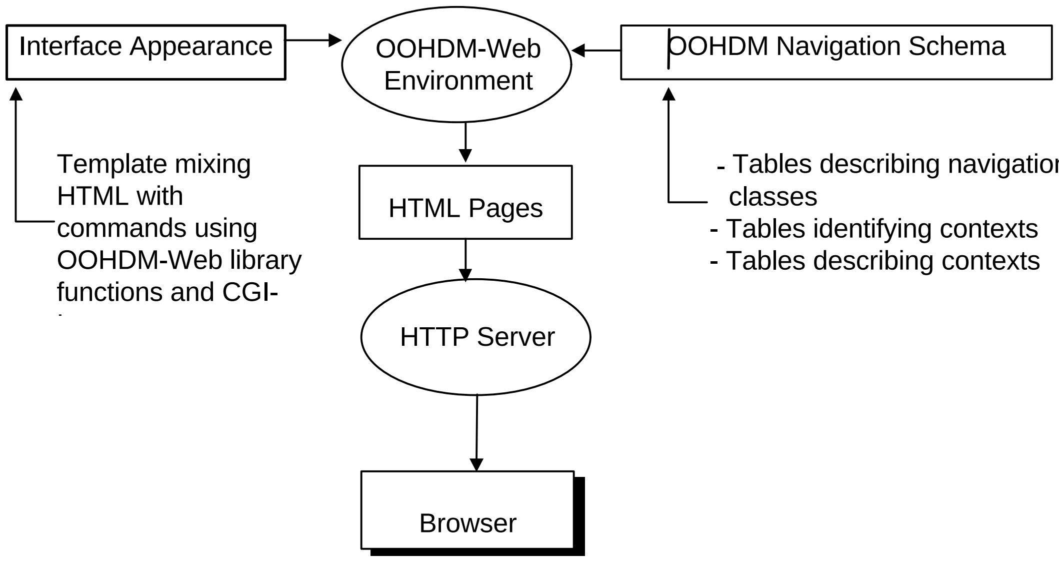

Interface Appearance
OOHDM-Web Environment
OOHDM Navigation Schema
Template mixing HTML with commands using OOHDM-Web library functions and CGI-
HTML Pages
- Tables describing navigation classes
- Tables identifying contexts
- Tables describing contexts
HTTP Server
Browser

Figure 13 - The structure of the OOHDM-Web Environment

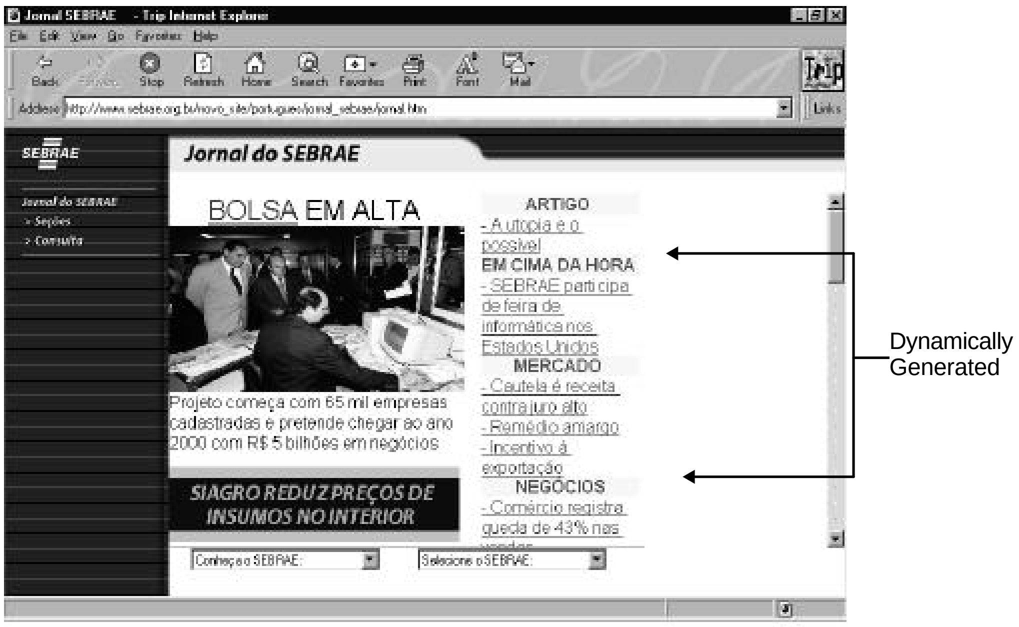
As a brief illustration, consider the table in Figure 1, which would be a simplified implementation of class Story in Figure 3.

Class Story

#key	title	text	image
gov	Governo aposta nos pequenos	bla bl bla	<IMG src = "\aaa.gif">
part	participação é tímida	no no no	<IMG src = "\bbb.gif">
balc	Balcões SEBRAE	bla no no	<IMG src = "\ccc.gif">

Figure 14 A relational table implementing the class "Researcher"

In Figure 2, we show a screen of an index to the "Story by Section" context (see Figure 4). This screen is actually generated from the template shown in Figure 3 where one can see the function calls to the OOHDM-Web library functions "Index" and "Vertical".


Jornal SEBRAE - Trip Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Print Font Mail
Address http://www.sebrae.org.br/novo_site/portugues/jornal_sebrae/jornal.htm
Links
SEBRAE
Jornal do SEBRAE
> Seções
> Consulta
Conheça o SEBRAE:
Selecione o SEBRAE:
Dynamically Generated

Jornal do SEBRAE

BOLSA EM ALTA

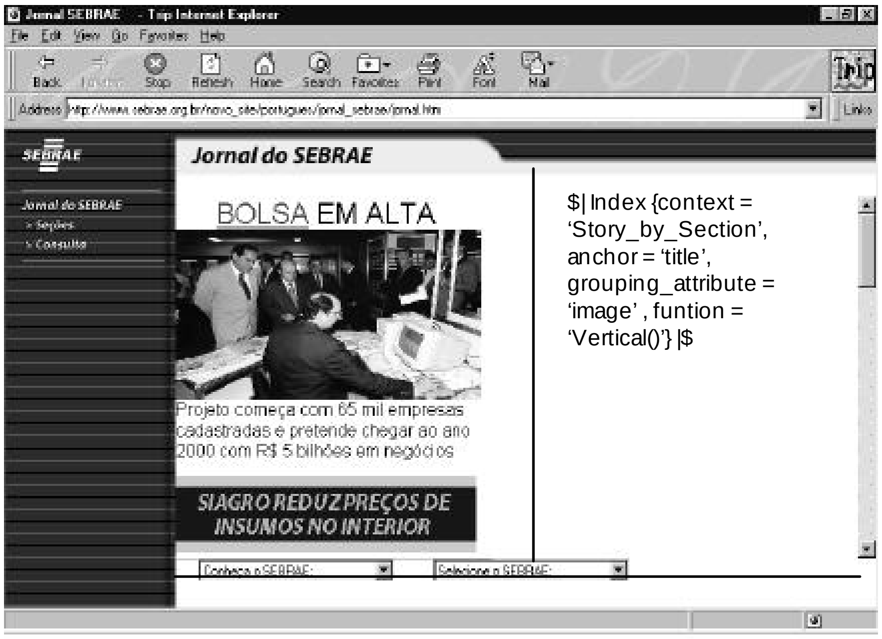
Projeto começa com 65 mil empresas cadastradas e pretende chegar ao ano 2000 com R$ 5 bilhões em negócios

SIAGRO REDUZ PREÇOS DE INSUMOS NO INTERIOR

ARTIGO
- A utopia e o possível
EM CIMA DA HORA
- SEBRAE participa de feira de informática nos Estados Unidos
MERCADO
- Cautela é receita contra juro alto
- Remédio amargo
- Incentivo à exportação
NEGÓCIOS
- Comércio registra queda de 43% nas

Figure 15 Index of Stories by Section

In this example, the "Index" function call will generate a list of Story titles, in alphabetical order, accessing the table in Figure 1, and display it using the function "Vertical". This function, in turn, generates a one-column HTML table where each element of the list is in a separate row. It should be noted that in this example, the designer had to define the contexts by creating the appropriate entries in the database, populate the databases with class instances, and define the HTML templates for presentation. The rest of the implementation is automatically generated by functions in the OOHDM-Web environment.


Jornal SEBRAE - Trip Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites Print Font Mail
Address http://www.sebrae.org.br/novo_site/portugues/jornal_sebrae/jornal.htm
Links
SEBRAE
Jornal do SEBRAE
Jornal do SEBRAE
> Seções
> Consulta
BOLSA EM ALTA
SIAGRO REDUZ PREÇOS DE INSUMOS NO INTERIOR
Conheça o SEBRAE:
Selecione o SEBRAE:

Projeto começa com 65 mil empresas cadastradas e pretende chegar ao ano 2000 com R$ 5 bilhões em negócios

```
$| Index {context =
'Story_by_Section',
anchor = 'title',
grouping_attribute =
'image' , funtion =
'Vertical()'} |$
```

Figure 16: An uninterpreted template for the page shown in Figure . Notice the call to OOHDM-Web library functions "Index" and "Vertical".

There are several implementation decisions that must be made regarding the design of page layouts. We briefly discuss a few in order to illustrate the set of tasks the designer must undertake. One of the first decisions to be made is whether anchors will be inserted within the contents of nodes, or will be kept separate. The first alternative usually requires insertion by hand; the second alternative is more amenable for either automated or semi-automated processing of link authoring,

For example, many sites carrying news stories, such as CNET's http://www.news.com, or ZDNet's AnchorDesk (http://www.anchordesk. com) show links in a separate area , except for links to companies appearing within the text, which are likely to be generated automatically. Some authors also argue that putting anchors inside the text stimulates users to navigate away from the site, so this technique encourages users to focus on the node's contents before looking for other places to go.

A second type of decision has to do with dealing with screen real estate. Quite frequently, the amount of information to be displayed is much bigger than what is possible to show without cluttering the screen. In such situations, a common solution is to use the "Information on Demand" design pattern (see [Rossi 97]), whereby only the most important information is shown up front, and the rest is only shown on demand by the user.

For more complex interface behavior, current basic browser capabilities are somewhat limited. For example, the ADV-chart may specify that an anchor (active ADV) be highlighted when it is the focus of attention, i.e. when it receives the external event MouseOn; its reaction is to send itself the message highlight. To achieve such effects, the browser functionality can be extended with functions in Javascript; an example can be seen in http://www.reference.com/javascript. Similarly and other extensions to browser functionality can be obtained using Java applets or plugins such as Macromedia's Flash or Shockwave. Many features may also implemented using Cascading Style Sheets and Dynamic HTML, although only the most recent generation of browsers supports it.

The use of templates for generating pages has many advantages. First, it is more consistent with the modeling approach, reducing the gap between design steps. Second, it makes it very easy to change the appearance of an entire application without extensive recoding of individual pages. Third, it allows the incorporation of other types of approaches that extrapolate plain HTML, such as the ones mentioned in the previous paragraph.

The constant evolution of standards, highlights the need to factor out as many design decisions as possible, to allow an application to change its implementation of the interface, taking advantages of newly available functionalities. This separation minimizes the amount of re-coding that must be done in the implementation when new functionalities are used in the runtime environment, as a consequence, for example, of new standards.

2.4.4 Statically versus Dynamically Generated Sites

The use of databases to store information items and context information, and the use of templates, seem to indicate that applications using this approach must necessarily be implemented as dynamically generated sites. These are sites where pages do not exist as files, but are the result of computations triggered every time a document is requested from the server. While this may often be the case, it is not always true.

Take for example an application domain in which all the information items are known before application deployment - in fact, most sites that are currently in the WWW are of this kind. This means that, even if the data is stored in databases, these databases do not change (or change very infrequently) over time. In this case, entire sites defined using the approach described in the previous sub-sections may be "compiled" before application deployment, pre-computing and generating static pages and storing them in files that are directly accessed by the server. Accordingly, all links are represented as static URLs, and it is not necessary to use any of the mechanisms previously mentioned.

This solution keeps the advantages in terms of application maintenance and evolution, without paying the performance price. This approach is used in the NetObjects Fusion tool [http://www.netobjects.com]

There are, of course, cases in which this approach is not feasible, either because the data changes very frequently (e.g., a news site such as http://www.news.com), or because it depends on user input (e.g., an online store such as http://www.amazon.com). Even in such cases, it might be worthwhile to partition the site into sub-sites, such that the dynamically generated part is in one sub-site, and the rest, which can be static, is in another sub-site(s)

3. Using OOHDM

The models used in the four phases discussed in the previous section are sufficient to allow the design of most web-based information systems. Nevertheless, as with any method, there is additional knowledge that is gathered by designers in practice, which is not part of the method itself. The research around OOHDM includes several developments in this direction that are being carried out, but are not reported here for reasons of space; we briefly outline them to give an overall picture of OOHDM and related research.

One approach that has been recently used to capture design knowledge, especially in the OO field, is the use of Design Patterns, which systematically name, explain and evaluate important and recurrent designs in software systems. They describe problems that occur repeatedly, and describe the core of the solution to that problem, in such a way that we can use this solution many times in different contexts and applications. Looking at known uses of a particular design pattern, we can see how a successful designer solves recurrent problems. In this way, we claim that using design patterns designers can profit from design knowledge that exists in several communities, such as hypermedia or user interface design

We have been collecting design patterns suitable for hypermedia application design [Rossi 97]. Our objective is to develop a system of inter-related patterns dense enough to be able to express complete designs as the successive application of patterns in this system. We have structured these patterns in three sub-groups, namely architectural, navigation and interface patterns.

Architectural patterns give guidelines for implementing software substrates for hypermedia applications. These patterns are quite similar to patterns in [Gamma 95], since they address problems such as decoupling navigation from other kinds of behaviors, organizing hierarchies of link and node types, decoupling link activation from the process of determining the link end-point. More details can be found in [Rossi96a, Garrido 97].

Patterns in the navigation category help the organization of the navigational structure of a hypermedia application to make it clear and meaningful for the intended readers. They address recurrent problems whose solution determines the degree of success of hypermedia applications. An interesting example of a navigation pattern is the “Active Reference” pattern, whose goal is to provide a perceivable and permanent reference about the current status of navigation. It combines an orientation tool with an easy way to navigate to a set of related nodes, at the same or higher position in the navigation structure. This pattern helps in building simple path viewers currently not provided by current WWW browsers.

The “Active Reference” pattern has been used in many web-sites for improving navigation. For example, in http://city.net/countries/ brazil/rio_de_janeiro, there is a bar with a representation of the logical path from the root to the current node. The reader has a simple way to understand where he is, where he can go next while accessing data about a city, in this case Rio de Janeiro. See [Rossi 97] for a complete description of this pattern.

Interface patterns are meant for hypermedia GUI designers. They are abstract and therefore independent of the environment used for the implementation. Graphical interface design is a complex task, concerned mainly with finding the right combination of elements (both in quantity and in their spatial relations), in such way that those elements interact for an effective presentation of the information.

Patterns in this group can be also applied outside the realm of hypermedia applications, in the broader context of GUI design. For example the “Information/Interaction decoupling” design pattern is aimed at solving the problem of how to make the interaction between the application and the user clearer, at the graphical interface of a node. This pattern is particularly useful in web sites when pages are generated dynamically and we cannot define link anchors as hotwords embedded in the text.

The “Information/Interaction Decoupling” pattern gives clear guidelines with respect to the physical placement of navigation anchors. The “Behavioral Grouping” design pattern helps the designer build an interface in such a way that the user can easily understand the kind of operations he is allowed to perform in the interface. This pattern solves the problem of organizing the interface when many different kinds of transactional, navigation and interface functionalities must be provided simultaneously. A deeper description of interface patterns can be found in [Garrido 97].

Although we have stated that Navigation Design should be done taking into account user profiles and tasks, OOHDM itself does not provide, so far, any indication on how this should actually be done. We have been investigating [Barroso 98] the use of user-centered scenarios to help identify user classes and typical tasks to be supported by the application.

The proposed method starts with a preliminary conceptual model drawn by the designer from his understanding of the domain. Looking at scenarios described by different classes of users, the designer builds partial navigation trails, and partial Navigation Class schemas. After all scenarios have been analyzed, the designer begins a process of merging partial trails and schemas, which culminates with a Navigation Context diagram and an updated Navigation Class schema.

In the course of this research, we have extended OOHDM to incorporate a security model to allow controlled access to objects. This model takes into account user classes and contexts, and defines mechanisms for specifying certain kinds of dynamic contexts that are built as a result of specified user actions.

Another important issue is building software environments for supporting the method; we have followed two different approaches:

- We have built a CASE environment that allows a designer to describe the conceptual, navigational and interface models using the OOHDM notation and provides him with automated documentation about those models. He can next generate implementation templates for different settings, such as Asymetrix's Toolbook or HTML. (See [Lyardet 96]).

- We have designed and implemented an object-oriented framework (OONavigator) for enhancing object-oriented information systems, improving the access to their information resources by adding a "navigational" front-end, seamlessly integrating this navigation functionality with the application's own computations. In OONavigator, hypermedia concepts (nodes, links, indices, and contexts) are modeled as components that are interleaved with application objects and their interface. Application classes play the role of OOHDM conceptual classes, while framework objects (nodes and links) comprise the navigational component of the application. We have enriched the standard MVC interface paradigm [Krasner 88] with anchoring facilities in such a way that nodes interfaces support the "point and click" hypertext interface metaphor. The interface can be published in Web browsers using tools like VisualWave. Using OONavigator, a designer can enrich an object-oriented application with hypertext features by following OOHDM guidelines. In this case the designer instantiates hypermedia and interface classes (using a visual tool) and connects them to his application classes to allow navigation through the application's information space. (See [Garrido 96, Rossi 98a, Rossi98b])

4. Concluding Remarks

4.1 Related Work

As we previously said, methodologies for Web applications design are still in their infancy (provided the Web also is); existing approaches tend to neglect either the navigation or the behavioral aspects of Web applications.

In [Takahashi 97] the authors propose using an extension of RM [Izakowitz 95] as the supporting method for building web information systems. They enrich the Entity-Relationship based model with scenarios showing the interactions among entities, agents and products. While we find their proposal appealing, it suffers the same problems of existing behavioral extensions to structured information models; instead of using objects they build the application's behavior in an artificial way by separating static and dynamic models. Similar approaches for separating static from dynamic and functional aspects, even in the context of object-oriented modeling such as Rumbaugh's OMT [Rumbaugh 91], have failed and have been discarded by their proponents (see for example the UML notation in [UML 97]). This happens because they result in systems that are difficult to extend and maintain. From the hypermedia point of view, our approach is stronger because it provides higher level design primitives, such as Navigational Contexts, which allow a better organization of the hyperspace. Besides, separating the user-interface design aspect of these applications allows defining a language that we can share with interface experts. Our interface design patterns are a step in that direction.

From an architectural point of view, separating concerns among conceptual, navigational and interface objects helps to produce applications that are easier to extend and/or maintain. This happens because relationships among objects in different levels follow the kind of collaborations in well-proven design patterns (such as the Observer),

In [Gellersen 97], an object-oriented support system for the development of Web applications is proposed. Our methodology can be used as a design front-end for this engineering tool. In addition, as we have discussed in this paper, the underlying philosophy behind OOHDM is that it yields design artifacts that can be implemented even using non object-oriented or hybrid tools.

Finally, as we mentioned previously, OOHDM provides a design dimension that is usually neglected in object-oriented environments and methodologies: the navigational design activity. Good object-oriented products for web applications, such as VisualWave or ClassicBlend, can be used with modern methodologies such as OOSE [Jacobson92] or UML [UML 97]. However, they fail to consider navigation as a design problem, and applications are built using the usual two level design partition of the MVC model. The only difference is that interfaces are generated as a combination of HTML and Java applets.

As another case in point, there are a number of more recent tools in the market, such as NetObjects "Fusion", that look at WWW based applications ("sites") as more than an unstructured collection of HTML pages. They allow the definition of (visual) styles (templates) to be uniformly applied to pages in the site, which is a step in the right direction. However, they still confuse the navigation topology with the physical directory structure used to store the pages in the site. Therefore, a given page can only have one "next" page (or one "parent" page), independently of the way the reader has arrived in it.

We consider, similarly to [Bieber 97], that designing the hypermedia features of a Web application is as critical as the behavioral ones and both must be considered as "first class" problems. Furthermore, we firmly believe that interface design must be done separately from navigation design.

4.2 Summary and future work

We have presented an object-oriented method for designing and implementing web applications. OOHDM supports the design of applications ranging from simple Web sites to more complex applications; it is also well suited as the navigational design front-end for other object-oriented methods such as UML [UML 97]. We have discussed the main issues involved in implementing OOHDM designs in the WWW, showing the main decision items and discussing alternatives.

We have designed and implemented many web applications, ranging from simple ones to quite elaborate ones where pages are generated dynamically depending on the context in which they are accessed, and retrieving data from databases. The use of OOHDM and the design patterns has reduced the overall development time, and has allowed us to discuss designs before actually having running implementations. Another benefit of the OOHDM models has been improving communication both with clients and with other non-computer science professionals such as graphics designers and marketing specialists that are part of the design team.

We are now pursuing several lines of research, as outgrowth or continuation of the research reported here:

- development of a rich and dense set of design patterns that will allow complete designs to be expressed almost entirely in terms of pattern compositions and instantiations;
- development of a direct manipulation language that will allow rapid prototyping of interface specifications for HTML documents;
- design and implementation of a set of tools that will constitute a development environment based on OOHDM, for web-based applications, so that designers are able to deal with entire sites at the more appropriate level of abstraction; a first component in this environment is OOHDM-Web, mentioned in section 2.4.1;
- design and implementation of a Java-based substrate to support direct implementation of OOHDM designs;
- extension of OOHDM to incorporate user models and tasks, a security model and definition of dynamic contexts;
- extension of OOHDM to support groupware, and investigation on support for distributed group authoring using OOHDM.

5. References

[Alexander77] Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel: "A Pattern Language". Oxford University Press, New York 1977.

[Barroso 98] Barroso, N.G.; “User centered design of hypermedia applications”, MSc thesis, Dept. of Informatics, PUC-Rio, 1998 (in Portuguese)

[Banerjee87b] J. Banerjee et al, “Data model issues for object oriented applications", ACM TOIS 5, 1987.

[Barbosa 95] S. Barbosa de Oliveira : “Modeling and Specification of Navigation in Hypermedia Applications”, Master Dissertation, PUC-RIO, 1995, (in Portuguese).

[Beck97] K. Beck: “Smalltalk Best Practice Patterns”. Prentice Hall, NJ, 1997

[Bieber97] Bieber, M; Vitali, F.;"Toward Support for Hypermedia on the World Wide Web" *IEEE Computer* 30(1), January 1997. Also availabale at http://www.cs.unibo.it/~fabio/bio/papers/1997/IEEEC97/January/IEEEC0197.html

[C Blend96] Classic Blend. Applied Reasoning Systems, 1996. http://www.arscorp.com.

[Carneiro 94] Carneiro, L.M.F.; Coffin, M.H.; Coewan, D.D.; Lucena, C.J.P.L; “ADVCharts: a Visual Formalism for Highly Interactive Systems”, in M.D. Harrison, C. Johnson, eds, *Software Engineering in Human-Computer Interaction*, Cambridge University Press, 1994.

[Coleman92] D. Coleman; F. Hayes; S. Bear, “Introducing Objectcharts or How to use Statecharts in Object-Oriented Design”, IEEE Transactions on Software Engineering, 18(1), 9-18, January 1992.

[Cowan93] D. D. Cowan; R. Ierusalimschy; C.J.P. Lucena; T.M. Septien, “Abstract Data Views”, Structured Programming, 14(1):1-13, January 1993.

[Cowan95] D. D. Cowan; C. J. P.Lucena, “Abstract Data Views, An Interface Specification Concept to Enhance Design for Reuse”, IEEE Transactions on Software Engineering, Vol.21, No.3, March 1995.

[Gamma95] Gamma, R. Helm, R. Johnson and J. Vlissides: "Design Patterns: Elements of reusable object-oriented software", Addison Wesley, 1995.

[Garrido96] A. Garrido, G. Rossi: “A framework for extending object-oriented applications with hypermedia functionality”. The New Review of Hypermedia and Multimedia, pp. 25-42. 1996,

[Garrido97] A. Garrido, G. Rossi, and D. Schwabe: “Patterns Systems for Hypermedia”. Submitted to PLoP’97, Pattern Language of Program, 1997.

[Garzotto93] Garzotto, D. Schwabe and P. Paolini: “HDM - A Model Based Approach to Hypermedia Application Design”. ACM Transactions on information Systems, 11 (1), Jan. 1993, pp. 1-26.

[Garzotto94] Garzotto, L. Mainetti and P. Paolini: “Adding Multimedia Collections to the Dexter Model”. In Proc. ECHT’94-ACM Conference on Hypermedia Technology. Edinburgh, UK, Sept. 1994.

[Garzotto96] Garzotto, L. Mainetti and Paolo Paolini: “Information reuse in hypermedia applications". Proceedings of Hypertext'96, Washington, 1996, pp. 93-104.

[Gellersen97] H. Gellersen, R. Wicke, M. Gaedke: "WebComposition: An Object-Oriented Support System for the Web Engineering Lifecycle" Electronic Proceedings of The Sixth International WWW Conference, Santa Clara, USA, April, 1997.

[Gronbaek94] Gronbaek: "Composites in a Dexter-Based Hypermedia framework". Proceedings of the ACM European Conference on Hypertext, (ECHT'94), Edinburgh 1994, pp. 59-69.

[Halasz94] Halasz e M. Schwartz: "The Dexter Hypertext Reference Model". Comm., of the ACM, February 1994, pp. 30-39.

[Hardman 93] L. Hardman; D. Bulterman; G. Van Rossum, "Links in Hypermedia, the Requirements for Context", Proceedings Hypertext'93, ACM, pp. 183-191.

[Hannemann93] J. Hannemann, M. Thuring, "What matters in developing interfaces for hyperdocument presentation?", Workshop in Methodological Issues on the Design of Hypertext-based User Interfaces, Darmstadt, Germany, July 1993.

[Harrison93] W. Harrison and H. Ossher. "Subject-Oriented Programming (a critique of pure objects). In Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'93), pp. 411-428, Washington, September 1993.

[Hester 97] A.M. Hester; R.C.Borges; R. Ierusalimschy; "CGILua: A Multi-Paradigmatic Tool for Creating Dynamic WWW Pages", Proceedings of the XI Brazilian Software Engineering Symposium (SBES'97) pp.347-360, Fortaleza, Brazil, 1997 (available at http://www.tecgraf.puc-rio.br/~anna/cgilua/ cgilua.ps.gz)

[Hunter 95] A. Hunter, I. Ferguson, S. Hedges: "Swoop: An application generator for Oracle/WWW Systems". Proceedings of the Fourth International World Wide Web Conference. pp. 185-194, 1995.

[Ierusalimschy 96] R. Ierusalimschy, L. H. de Figueiredo and W. Celes, "Lua - an extensible extension language", Software: Practice & Experience 26 #6 (1996) 635-652. (see also http://www.tecgraf.puc-rio.br/lua/).

[Izakowitz95] Izakowitz, E. Stohr and P. Balasubramaniam: "RMM: A methodology for structured hypermedia design". Comm. of the ACM, October 1995, pp. 34-44.

[Jacobson 92] Jacobson, I; Christerson, M.; Jonsson, P., Övergaard, G.; "Object Oriented Software Engineering - A Use Case Driven Approach", Addison Wesley, 1992.

[Keller 97] Keller, W.; "Mapping Objects to Tables – A Pattern Language", Proc. Of European Conference on Pattern Languages of Programming Conference (EuroPLOP)''97, Bushman, F. and Riehle, D.; (eds), Irsee, Germany, 1997. (http://www.sdm.de/g/arcus/publicat/index.phtml#Mapping_Objects_To_ Tables)

[Kim 94] W. Kim, "Advanced Database systems", ACM Press, 1994.

[Kirste93] T. Kirste: "Some issues of defining a user interface with general purpose hypermedia toolkits". Workshop in Methodological Issues on the Design of Hypertext-based User Interfaces, Darmstadt, Germany, July 1993.

[Krasner 88] G. Krasner, S. Pope. " A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80". Journal of Object-Oriented programming, 1(3), pp. 26-49, August/September, 1988.

[Lange94] D. Lange. “An Object-Oriented Design Method for Hypermedia Information Systems”. Proceedings of the 27th. Annual Hawaii International Conference on System Science, Hawaii, Jan., 1994.

[Lyardet 96] F. Lyardet and G. Rossi "Enhancing productivity in the development of hypermedia applications". Proceedings of the 7th workshop on the next generation case tools, Heraklion, Crete, Greece, May, 1996.

[Mere96] M.C. Meré, G. Rossi: “Specifying navigational transformations in hypermedia. A temporal logic framework”. In Bodo Urban (ed) Multimedia’96. pp. 20-31. Springer Computer Science, Springer Verlag New York, 1996.

[Milet 96] J.Renato Milet; D. Schwabe; R. S. G. Lanzelote, “Hypermidia Application Authoring Using Object Oriented Databases”, Proc. of the XI Brazilian Symposium on Databases (SBBD), SBC, São Carlos, Oct. 1996 (in Portuguese).

[Nielsen90] J. Nielsen: “Hypertext and Hypermedia”. Academic Press, 1990.

[Pontes 97] Pontes, R.C.A., “An Environment to Support Hypermedia Applications in the WWW”, MSc thesis, PUC-Rio, 1997 (in Portuguese).

[Rossi95a] G. Rossi; D. Schwabe; C.J.P. de Lucena; D.D. Cowan, “An Object-Oriented Model for Designing the Human-Computer Interface of Hypermedia Applications”, Proc. of the International Workshop on Hypermedia Design (IWHD'95), Springer Verlag Workshops in Computing Series, forthcoming. (available at <ftp://ftp.inf.puc-rio.br/pub/docs/techreports/95_07_rossi.ps.gz>).

[Rossi96a] Rossi, A. Garrido and S. Carvalho: "Design Patterns for Object-Oriented Hypermedia Applications". Pattern Languages of Programs 2, Vlissides, Coplien and Kerth *eds.*, Addison Wesley, 1996.

[Rossi96b] Rossi, D. Schwabe and A. Garrido: "Towards a Pattern Language for Hypermedia Applications". Proceedings of the 3rd. Annual Conference on Pattern Languages of Programs, Monticello, Illinois, September 1996.

[Rossi96c] G. Rossi, M.C. Meré: “A Temporal Logic framework for representing knowledge about navigation in hypermedia applications”. Proceedings of the Workshop on Knowledge Representation for Interactive Multimedia Systems in the European Conference on Artificial Intelligence (ECAI’96), pp. 66-71. G. Vouros (editor), Budapest, 1996.

[Rossi96d] G. Rossi: “An Object-Oriented Method for Designing Hypermedia Applications”. PHD Thesis, Departamento de Informática, PUC-Rio, Brazil, July 1996 (in Portuguese).

[Rossi97] G. Rossi, D. Schwabe and A. Garrido: “Design Reuse in Hypermedia Applications Development” Proceedings of ACM International Conference on Hypertext (Hypertext’97), Southampton, April 7-11, 1997, ACM Press.

[Rossi 98 a] G. Rossi, A. Garrido and D. Schwabe: "Navigating between Objects: Lessons from an Object-Oriented Framework Perspective", to appear in ACM Computing Surveys.

[Rossi 98 b] G. Rossi and A. Garrido: "Capturing Hypermedia Functionality in an Object-Oriented Framework", to appear in Object-Oriented Frameworks, Wiley 1998.

[Rumbaugh91] Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W.Lorensen: "Object Oriented Modeling and Design", Prentice Hall Inc. 1991.

[Schmidt95] D. Schmidt: "Using Design Patterns to develop reusable object-oriented communication software" Comm. of the ACM, October 1995, 38(10), pp. 65-74.

[Schwabe94b] D. Schwabe; S. D. J. Barbosa, “Navigation Modeling of Hypermedia Applications”, Technical Report MCC 42/94, Departamento de Informática, PUC-Rio, 1994 (available at <ftp://ftp.inf.puc-rio.br/pub/docs/techreports/ 94_42_barbosa.ps.gz>)

[Schwabe95a] D. Schwabe and G. Rossi, “Building Hypermedia Applications as Navigational Views of Information Models”, Proc. of the Hawaii International Conference on System Sciences, Hawaii, Jan. 1995. (available at <ftp://ftp.inf.puc-rio.br/pub/docs/techreports/ 94_41_schwabe.ps.gz>)

[Schwabe 95b] D. Schwabe and G. Rossi:, “The Object Oriented Hypermedia Design Model”, Comm. of the ACM, Vol. 38, #8, pp45-46 Aug. 1995. (available at <http://irss.njit.edu:5080/cgi-bin/bin/option.csh?sidebars/schwabe.html>).

[Schwabe96] Schwabe, G. Rossi and S. Barbosa: "Systematic Hypermedia Design with OOHDM". Proceedings of the ACM International Conference on Hypertext (Hypertext'96), Washington, March 1996.

[Takahashi 97] K. Takahashi, E. Liang: “Analysis and Design of Web-based information systems” Electronic Proceedings of The Sixth International WWW Conference, Santa Clara, USA, 1997.

[UML 97] UML Document Set. Version 1.013 January, 1997, Rational, 1997. (available at http://www.rational.com/uml/references/index.html)

[Varela 95] C. Varela, D. Nekhayev, P. Chandrasekharan, C. Krishnan, V. Govindan, D. Modgil, S. Siddiqui, , D. Lebedenko, M. Winslett: “DB: Browsing Object-Oriented Databases over the Web”. Proceedings of the Fourth International World Wide Web Conference. pp. 209-220, 1995.

[Vives 97] Francisco Vives, Pablo Zanetti, Alejandra Garrido "Adding Hypermedia Functionality to Object Oriented Applications". To be presented in the 4th International Conference. Hypertexts and Hypermedia: Products, Tools and Methods. 25th & 26th September 1997. Paris, France.

[Vwork96] The Visual Work Programming Environment. Parc Place-Digitalk, 1996.

[Vwave96] The VisualWave Programming Environment. Parc Place Systems. In http://www.parcplace.com/products/vwave/vwv_prod.htm.

[Wirfs-Brock90] R. Wirfs-Brock, B. Wilkerson, and L. Wiener: “Designing Object-Oriented Software”. Prentice Hall, Englewood Cliffs, NJ, 1990.