Web Design Frameworks: An approach to improve reuse in Web applications

Daniel Schwabe *, Gustavo Rossi **, Luiselena Esmeraldo *, Fernando Lyardet**

***Departamento de Informática, PUC-Rio, Brazil**
E-mail: {schwabe, luiselena} @inf.puc-rio.br
****LIFIA Facultad de Informática. UNLP.**
La Plata, Argentina
E-mail: {fer,gustavo}@sol.info.unlp.edu.ar

Abstract

In this paper we introduce Web design frameworks as a conceptual approach to maximize reuse in Web applications. We first discuss the need for building abstract and reusable navigational design structures, exemplifying with different kinds of Web Information Systems. Then, we briefly review the state of the art of object-oriented application frameworks and present the rationale for a slightly different approach focusing on *design* reuse instead of *code* reuse. Next, we present OOHDM-frame, a syntax for defining the hot-spots of generic Web application designs. We illustrate the use of OOHDM-frame with a case study in the field of electronic commerce. We finally discuss how to implement Web design frameworks in different kind of Web platforms.

1 Introduction

Building complex Web applications such as e-commerce applications is a time consuming task. We must carefully design their navigational architecture and user interface if we want them to be usable. We must understand the user tasks while he is navigating the hyperspace to decide which navigation facilities we should include; for example we may consider defining indexes, guided tours, landmarks, etc. according the user needs. The interface should help the user browse through the sea of information by giving him cues and feed-back on his actions, and by presenting the information in a clear and meaningful way. Moreover, this kind of application also includes complex behaviors, as they not only deal with buying or selling, but they are also integrated with the company's internal business; often providing different views of corporate databases, and acting as integrators of other applications. Another dimension in which these applications are different from what we may call "conventional" software is the need to reduce deployment and delivery times. Applications in the Web must be built quickly and with zero defects. We must improve not only development but also debugging and testing times.

To make matters worse, building applications in the Web involves using a myriad of different technologies such as mark-up languages (like HTML or XML), scripting languages (JavaScript, Pearl), general purpose object-oriented languages (Java), relational databases, etc. We should find ways to improve the process of building this kind of applications by systematically reusing both application code and design structures.

We have been designing Web applications using the Object Oriented Hypermedia Design Method (OOHDM) for some years [Schwabe98, Schwabe96]. OOHDM considers Web applications as *navigational views* over an object model [Rossi99c] and provides some basic constructs for navigation (contexts, indexes, etc) and for user interface design. Using OOHDM we can apply well-known object-oriented software engineering practices to the construction of applications involving navigation. In the context of OOHDM we have been looking at ways to maximize reuse in the development process, since we have observed a certain degree of commonality among solutions in similar application domains. For example, most online stores have similar navigation structures, and they provide similar functions to their users.

In this context we have found many recurrent patterns in Web applications; we have recorded them using a mixture of the GOF [Gamma95] and Alexandrian [Alexander77] styles. (See for example [Rossi99a, Lyardet99, Lyardet98, Rossi96]). We have found that micro-architectural reuse in Web applications is really feasible. However, if we want to move to architectural or design reuse, we need other concepts and tools in order to reason in terms of compositions of abstract and concrete Web application elements.

In this paper we introduce Web design frameworks as a novel concept to push design reuse in Web applications. We first review object-oriented frameworks and compare them with Web design frameworks. We next present OOHDM-Frame, a notation to specify Web design frameworks, and show an example in the field of electronic commerce. Then, we show how to map Web design frameworks to Web application frameworks and to Web applications, and present some ongoing research issues in this area.

2 Towards Web design frameworks

There are different ways to achieve reuse in the context of Web applications. We can for example reuse interface templates in the form of HTML or XML descriptions. We can reuse information accessing shared databases [Garzotto96]. We can go further and reuse components that exhibit some non-trivial behavior. For example we could reuse code

implementing shopping baskets in different e-commerce applications. Even though many of the supporting technologies may not support reuse (e.g. there is no inheritance or polymorphism mechanisms in HTML or XML, the code for shopping baskets might not be found in a single component, etc.) it seems that we can not go far beyond these examples.

As a consequence the most important kind of reuse, design reuse, has been largely unexplored in Web applications, perhaps due to the non object-oriented nature of the Web. In a previous paper [Rossi99a] we have introduced navigation patterns as a way to record, convey and reuse design experience. Though the kind of reuse provided by patterns is valuable, complex corporate applications need a way to maximize reuse of larger design structures. For example, the set of activities triggered when the user of an electronic store orders an item is usually similar in different stores. We should be able to express these commonalties in such a way that only the specific aspects of a particular store should be designed or programmed.

In the following sections we introduce Web design frameworks as a solution to this problem. We first review the state of the art in object-oriented frameworks and then highlight the differences with Web design frameworks.

2.1 Object-Oriented frameworks

Object-Oriented application frameworks are the state-of-the art solution for building high quality applications in a particular domain, by systematically reusing an abstract design for that domain [Fayad99]. An object-oriented (OO) application framework is a reusable design built from a set of abstract and concrete classes, and a model of object collaborations. A framework provides a set of classes that, when instantiated, work together to accomplish certain tasks in the intended domain. An application framework is thus the skeleton of a set of applications that can be customized by an application developer.

When many different applications must be constructed in the same domain, application frameworks provide "templates" for supporting their commonalties, and accommodating individual variations (differences). These "templates" usually have the form of abstract classes that must be sub-classified with concrete ones, or filled with "hook" methods that must be implemented by the application's designer [Pree94]. The framework's designer must understand the domain and be able to decouple the concrete model of a particular application from the abstract model of the whole domain. New applications can be built by simply plugging together framework and specific application components. Application frameworks have been built in areas such as user interface design, graphical editors, networks, financial applications, etc [Fayad99].

Let us suppose that we are building a framework for managing orders and delivery of products in different (non-electronic) stores. The framework will contain some abstract classes like Product, Client, Provider, Order, Invoice, etc. Their behavior will implement the usual flow of control in the store: when a client places an order for a product, a message is sent to the supplier, an invoice is generated, etc.

For a particular application (store) in this domain, one will need to either instantiate these classes or sub-classify them in order to accommodate both their structure and behavior to the particular features of this store, e.g. different kinds of products, various payment policies, etc. This is usually achieved (in the framework) by programming generic methods in abstract classes that are then used (in the specific application) as templates in concrete sub-classes.

This simple example helps to understand the problems with framework technology if we want to move to the Web environment – the need to adapt to an hybrid environment (object-oriented frameworks are usually programmed in a single programming language). In addition, Web applications involve another component, their navigational structure [Rossi99a, Rossi99b], since we are interested not only in the behavior of domain classes but also in the ways the user will navigate through them.

2.2 Why Web design frameworks

Web environments are not fully object-oriented. In the WWW we will have to define HTML pages, scripts in some language (such as JavaScript or Perl), queries to a relational database, etc. Conceptual and Navigation objects may have to be mapped onto a relational store, and behaviors defined during design may have to be programmed by mixing a scripting language, stored procedures, and so on. The main consequence of this fact is that "conventional" object-oriented application framework technology may still be inadequate in this domain, since we cannot suppose a single language environment as most frameworks do. Though there is a growing trend in "object-orienting" the WWW [IEEE99], we still need heuristics to perform these translations.

It may happen that we can program the full application using an object-oriented language (e.g. Java). Even in this case we will still lack an important part of the application's functionality: its navigational behavior. We have argued elsewhere [Schwabe98, Rossi99d] that Web application models require both an object (conceptual) model where we specify usual behavior, and a navigational model in which we define navigational components such as nodes, links, contexts, paths, etc. For Web applications to be successful, the navigational structure must be carefully defined, and current object-oriented approaches do not provide primitives for navigation design. As a consequence, framework technology is not completely adequate for this domain.

In the following sections we introduce Web design frameworks, which provide a bridge between current framework technology and Web environments.

3 Components of a Web design framework architecture

3.1 Definition

Let us consider a Web application as "a structured set of objects that may be navigated, and processed, in order to achieve one or more tasks". A Web *application framework* may be defined as "a generic definition of the possible application objects, together with a generic definition of the application's navigational and processing architecture". A Web application framework must then define the set possible objects to be navigated, how they can be structured in their navigation architecture, and how they may behave. Current framework technology would allow us to stress object relationships and behaviors in a specific programming language and wouldn't allow us to specify navigation architectures.

We define a Web *design framework* as a "generic design of possible Web application architectures, including conceptual, navigational and interface aspects, in a given domain". Web design frameworks must be environment and language-independent.

As previously said one of the important defining aspects of a framework are its hot-spots, i.e., the places in the framework where the designer may introduce the variations or differences for a particular application in the same domain of the framework. We have taken the approach of modeling many applications in the same domain (e.g., discussion lists, online publications, online stores...) using OOHDM, and comparing the resulting specifications. From this comparison, it was possible to determine the similarities and differences between them, which in turn allowed us to identify what should be the hot spots in a framework that could subsume the set of applications in each particular domain.

As a result, in order to define hot-spots for Web application frameworks, we used OOHDM models, namely Conceptual and Navigation, as a starting point. Before detailing hot-spot definitions, we briefly recapitulate some key concepts in the OOHDM approach that will serve as the basis for hot-spot definition.

The first key concept is that in a Web application, the user navigates over (navigation) objects that are *views* of conceptual objects; these views are defined opportunistically, according to particular user profiles and tasks. The second key concept is that navigation objects must be organized into useful structured sets, called contexts. The structure of these sets defines the intra-set navigational architecture, whereas the set of conceptual relations, which are mapped onto navigation links, defines the inter-set navigational architecture.

Since sets can be defined in different ways, this induces different types of contexts:

1 Simple class derived – includes all objects of a class that satisfy some property ranging over their attributes; e.g., "books with author=Umberto Eco", "CDs with performer = Rolling Stones", etc.
2 Simple link derived – includes all objects related to a given object; e.g., "reviews on "The Name of the Rose"", "CDs that were bought by persons who also bought "Flashpoint"", etc.
3 Arbitrary - The set is defined by enumeration. For example, a guided tour showing some pictures in a virtual museum, or some outstanding books in a collection.

Many times, contexts appear in families or groups of related contexts; the most common types are defined below

4 Class derived group – is a set of simple class derived contexts, where the defining property of each context is parameterized; e.g. "Products by Manufacturer", "Books by Keyword" (Manufacturer and Keyword can vary). (Notice that we are considering "Manufacturer" as an attribute of Product, as discussed previously).
5 Link derived group - a set of link derived contexts, each of which is obtained by varying the source element of the link; e.g. "Book by Order" (Order can vary).

A context is said to be dynamic if its elements may change during navigation. This can happen for two reasons – because it is possible to explicitly add to or remove elements from a context, or because it is possible to create new objects or links, or alter existing ones. In the latter case, all class (respectively, link) derived contexts automatically become dynamic.

A Web design framework may then be defined by an analogous set of models - a Conceptual Model, a Navigation Model, and rules for mappings between them. However, each of these models will be made up of different primitives than the ones used in OOHDM itself.

It must be emphasized that web design frameworks, while following the same philosophy as application frameworks, use quite different mechanisms for hot-spot definition and instantiation. Whereas the latter use subclassing and class instantiation to instantiate hot-spots, web design frameworks use selective mapping and generic context instantiations, as will be explained next.

For the sake of conciseness we do not include variability related with the Interface model in this paper.

3.2 Conceptual Model

A first question that must be answered is what characterizes the application domain. The first step in the design process is to define a Conceptual Model, upon which applications will be built by defining navigational views for each particular user profile. This architecture (present in OOHDM) already constitutes part of a framework, since it is possible to build many applications starting from the same Conceptual Model. Therefore, we define the application domain as being the model characterized by the Conceptual Schema; it defines the abstract classes, possibly some concrete classes, and the relationships that make up the domain of application. These classes may include (applicative) behavior specification as well. The hot-spots of an object-oriented framework for the application domain can be defined according to [Pree94].

In Figure 1 we present a generic conceptual model for electronic stores. Notice that genericity in the conceptual schema can be obtained by following well-known practices in object-oriented design [Fayad99]. In this paper we stress the novel aspect of Web design frameworks: the specification of generic navigation structures.

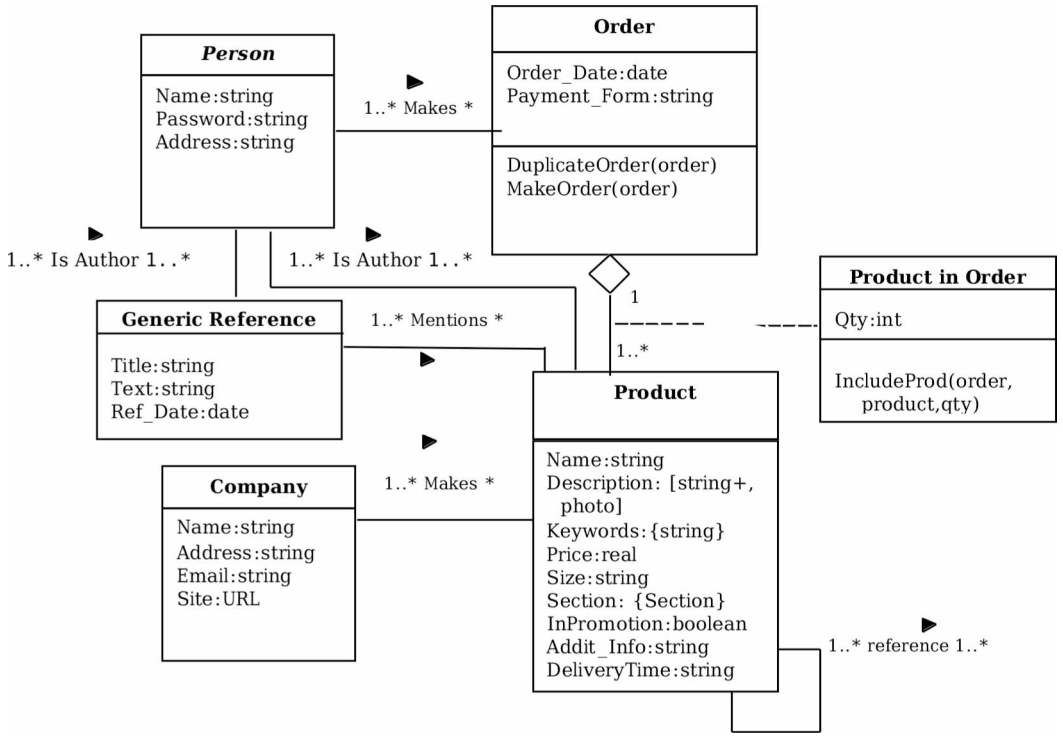


Figure 1 - An example of a conceptual schema for a generic electronic online store

3.3 Navigation Model

A Navigational model is defined by a Navigation Class schema (mapped from the Conceptual Model), and by a Navigation Context Schema. The elements in the Navigation Class Schema are abstract and concrete node classes, and links, defined as views over the Conceptual Class Schema. Differently from OOHDM, these classes may contain optional elements (attributes or methods); links may also be optional. Optional elements may be omitted when the framework is instantiated. Therefore, the optional inclusion of Navigation Class schema elements is a first hot-spot in a Web design framework.

As an example, in the e-commerce domain, one may define a Conceptual Class "Product" that has "Description_Image", and "Description_Text" attributes (See Figure 1). It is possible to define, in the framework's Navigation Class Schema, that there is a Navigation Class "Product", derived from the corresponding conceptual class, but where the "Description_Image" is optional. This means that this framework can be instantiated into applications that do not include an image of a product. Similarly, the framework may specify that the link "Related Product", between products, is optional, and therefore two different actual applications, one including it, another omitting it, are valid instances of this framework. In Figure 2 we show a possible navigational class schema in this domain.

To generalize, the first hot-spot in a Web design frameworks is defined by establishing the constraints on possible mappings between Conceptual an Navigation Classes, stating among other things that certain elements of the Navigation Class Schema are optional. These constraints must also address the issue of consistency during framework instantiation, which will be discussed later.

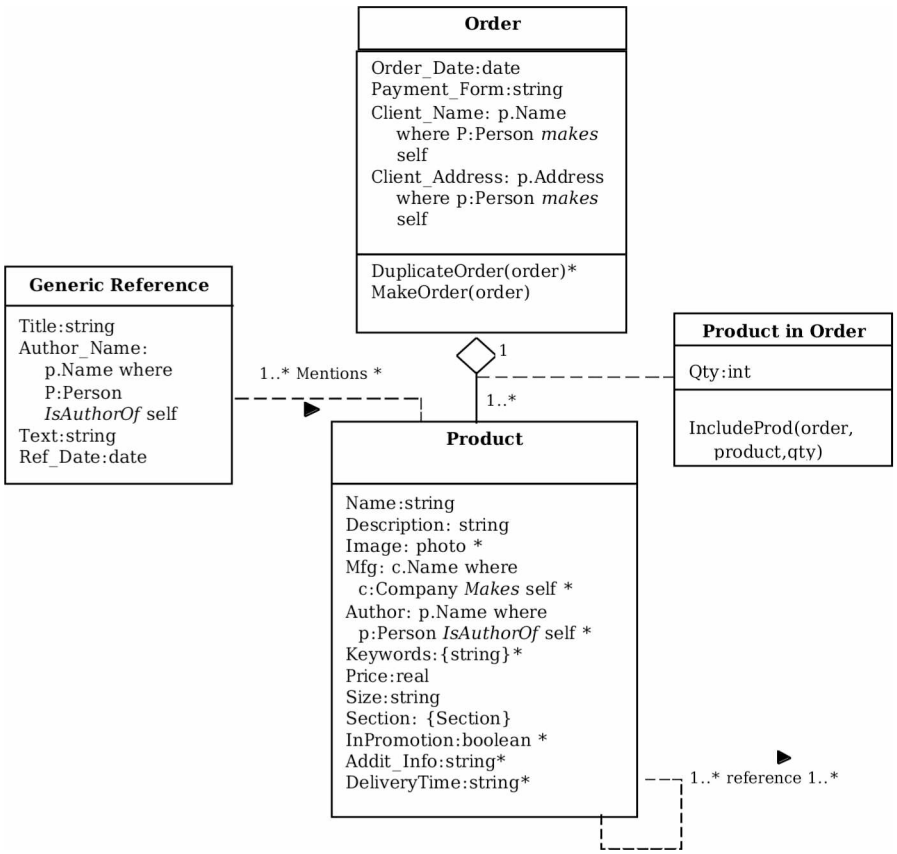


Figure 2 - An example Navigation Class Schema for an electronic online store. Dashed lines indicate optional links.

The second component of the Navigational model is the Navigation Context Diagram. For Web design frameworks, it is necessary to generalize the concept of Navigation Context, replacing it with the concept of Generic Navigation Context. A Generic Navigation Context is the specification of a *set* of possible Navigation Contexts, subject to some constraints. This is another type of hot-spot in a Web design framework, which is exercised during instantiation by substituting the Generic Context by actual Navigation Contexts, all of them satisfying the Generic Context's constraints. The framework's Context Navigation Diagram is supplemented with a set of instantiation rules, which further constrain possible Context Diagrams that may be derived from it during the framework instantiation process.

Consider for example an e-commerce application, where one is designing a product catalog section. One may define several contexts that group instances of Navigation Class "Product" in different ways, for instance, "Product by Category", "Product by Price", etc. Each of these defines a Navigation Context. A Generic Navigation Context can be "Any Class-Derived Context based on Product"; it is generic because it does not specify the particular property that is used to define each derived context. In addition, it constrains its concrete instances by requiring that all of them be based on some property over the attributes of Navigation Class "Product". When a given framework's Context Diagram includes concrete Navigation Contexts (i.e., non-generic), it means that all applications derived from this framework must include such contexts as defined in the specification, without variations.

Besides Navigation Contexts, a Context Diagram also contains the specification of Access Structures (indexes). By analogy, the framework's Context Diagram will also contain the definition of Generic Access Structures, which are generalizations of Access Structures. In Generic Access Structures, the criteria that may be varied are which elements are included, their ordering, whether the index is static or dynamic, and so on. Again, a Generic Access Structure may be substituted by several actual access structures, all of which satisfy its instantiation rules. The inclusion of concrete Access Structures in the framework's Context Diagram determines that all instantiations include that access structure; a common example is the "Main Menu" access structure, present in most frameworks.

Summarizing, a framework's Navigation Context diagram is made up of Generic Navigation Contexts and (regular) Navigation Contexts, plus a set of instantiation rules. It defines all valid Context Diagrams of actual applications that can be obtained from the framework instantiation.

4 The OOHDM-Frame notation

In order to specify Web design frameworks, we have defined a new set of models, called OOHDM-Frame. A framework specification in OOHDM-Frame is comprised of a Conceptual Model specification and a Navigation Model specification, together with instantiation rules. We have already discussed the Conceptual Model in section 3.2; it uses the same primitives and notation as OOHDM Conceptual Models plus hot-spots with the notation in [Pree94]. Whereas generic behaviors are specified in the conceptual model, generic navigation architectures are specified in the navigational model.

The Navigation Model in OOHDM-Frame is made up of a Navigation Class Schema, a Context Diagram, and a set of mapping and instantiation rules, controlling how the Navigation Class Schema is mapped onto the Conceptual Class Schema. The Navigation Class schema is similar to the Conceptual Class schema, except for the fact that Class attributes may be optional (marked with an "*") and Relations (links) can be optional (drawn with a dashed line). In addition, a Navigation Class in the Navigational Class Schema may also be sub-classed during instantiation, thus implementing at least in part the "traditional" hot-spot mechanism in OO frameworks. For example, class "Product" in Figure 2 may be sub-classed in actual applications, which will allow the definition of the real products in each case.

Let us briefly examine the notation used to represent generic contexts, and their respective context cards.

The first kind is the Generic Simple Context. It may be substituted during instantiation by any simple context (class or link derived). The context card will detail the instantiation restrictions, such as whether there can be from 0 to n instances (cardinality); whether the resulting contexts are communicating (i.e., it is possible to switch from one to the other at any moment during navigation within either one of them); and the allowed types with respect to persistence.

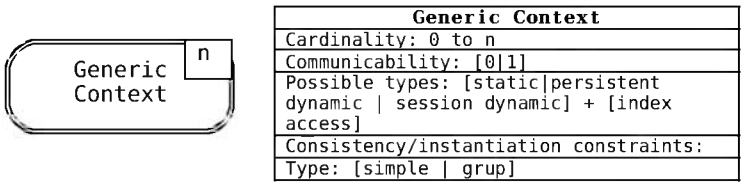


Figure 3 – A Generic Context, and its specification card.

The cardinality specification may be used to indicate that a generic context must have at least one instantiation, e.g., 1 to n. Arbitrary contexts, i.e., those whose elements are enumerated, are also represented with the same notation.

Figure 4 shows the representation of other generic contexts. An instance creation or modification context is a dynamic context that allows the creation of new object instances, or changing the attributes of an existing object. In this case, the only hot spot is choosing the cardinality – 0 means it may not be instantiated, 1 means its instantiation is mandatory.

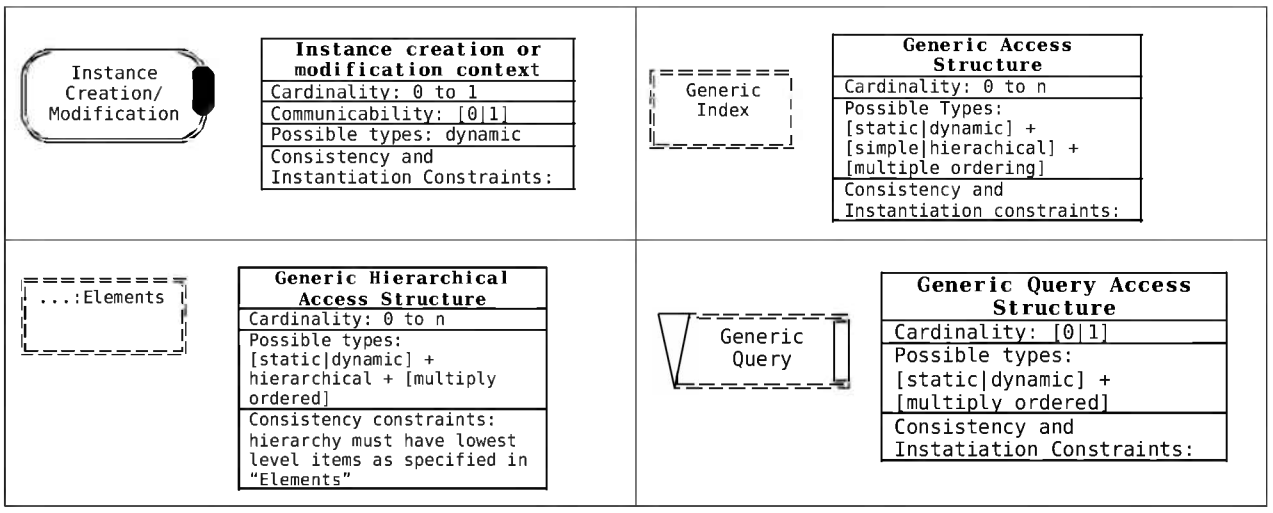


Figure 4– Other generic contexts and access structures and their specification cards

Similarly to contexts, access structures also have a counterpart, generic access structures. The only restriction placed on index instantiations is that it must be compatible with the context instantiations they point to. Since indexes may be hierarchic, it is also possible to make the actual hierarchy a hot spot. In this case, we use the notation shown in Figure 4.

Another type of hot spot is the property of a context of being protected or not. It should be recalled that contexts may have access restrictions for users of a certain type. This hot spot is denoted in the diagram with a double-edged oval next to the corresponding generic context or index

In the following section we will examine in more detail the actual process of framework instantiation, by looking at an example.

5 Instantiating a framework

Let us look at an example of a framework for an online store. The conceptual model has been shown in Figure 1and Figure 2.

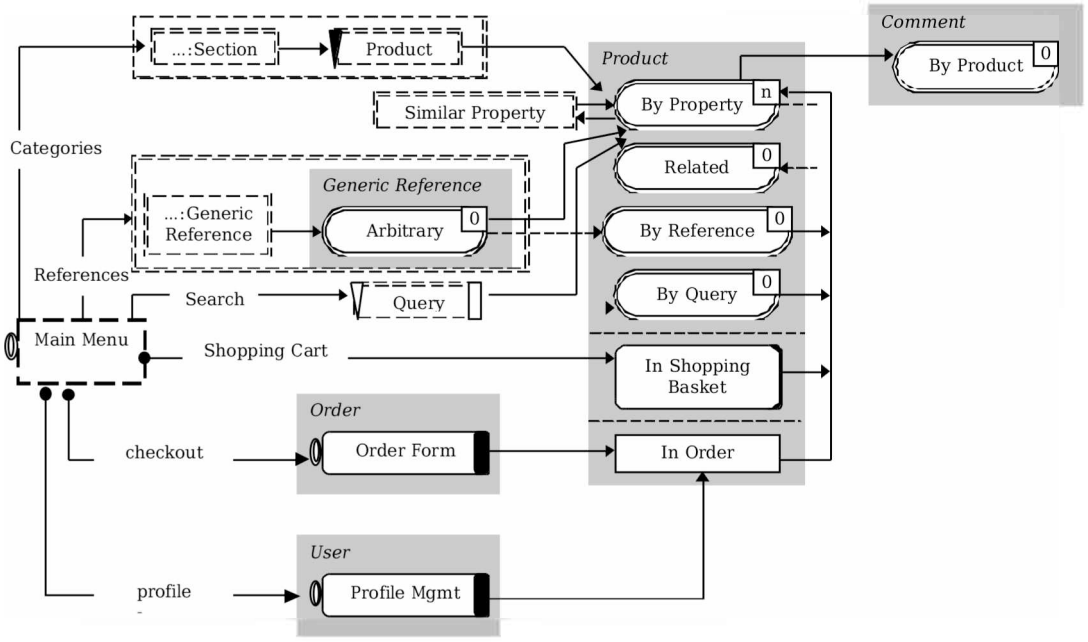


Figure 5 – The Context Diagram for an Online Store application framework.

This "generic" diagram is at an abstraction level that allows reasoning closer to the domain of online stores. The main goal of such sites is, evidently, to sell products. Therefore, it must allow as many paths as possible leading to products; furthermore, once the reader ("consumer") has reached a product, he should be lead (or shown) as many additional products as possible.

Generic context "Product by Property" is a either simple our group class derived context, which will be typically instantiated into one or more contexts that allow navigation among products according to certain properties (e.g., "Product by price"; "Product by size"; "Product by Color"; etc…). Once within any of these, it is normally possible to navigate to other "Related Products" (e.g., accessories, matching products, etc…). There are several access structures that lead the reader into these contexts; typically, these are hierarchical access structures that reflect product sections (departments) in a real world store.

Additional paths leading to products can be offered by opportunistically grouping products according to some (arbitrary) criteria, such as "N.Y. Times Bestsellers List", "John's Recommendations", or "Promotions". Such groupings are modeled by the generic "Products by Reference" context, which can be reached through the generic hierarchical index "…: Generic Reference".

The shopping basket and the order itself are modeled as concrete dynamic contexts, which are always present in any instantiated application (note the absence of dashed lines). In addition, the access to order is normally protected by some identification process (notice the double oval next to the "Order Form" context). Finally, some online stores may require customer identification at the entrance, which is modeled by having the "Main Menu" access structure is optionally protected.

Let us now look at how this framework may be instantiated into the "book" section of Amazon.com's website (http://www.amazon.com). We have deliberately chosen to exemplify with only a portion of that website, for reasons of clarity and space; for instance, we have not included user profile management that appears in the framework. Figure 6 shows the context diagram for this section of Amazon.com's website.

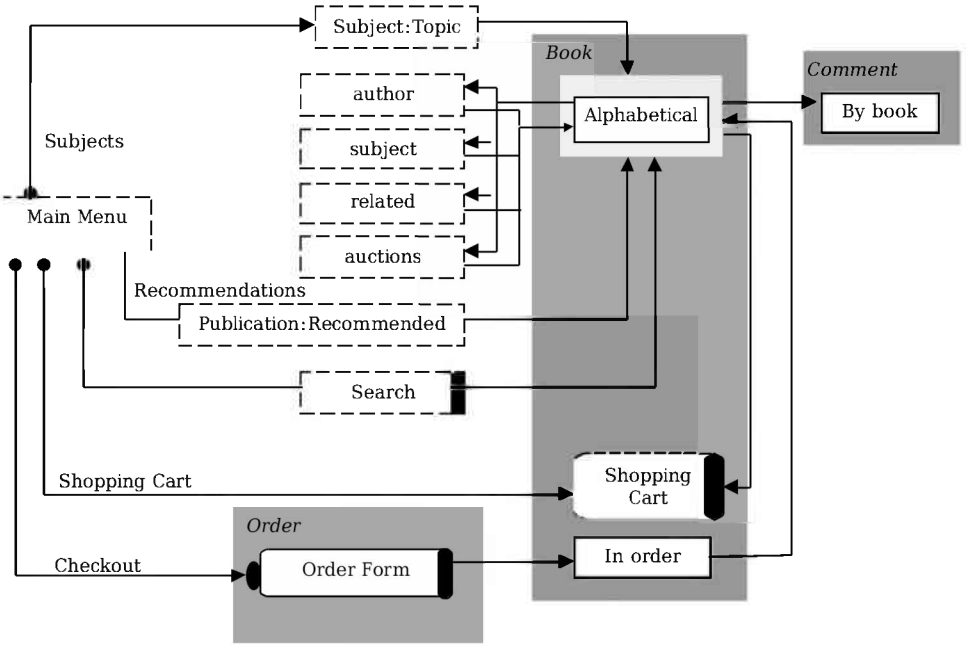


Figure 6 – The Context Diagram for http://www.amazon.com, an instantiation of diagram in Figure 5.

It can be readily seen that, as discussed in [Rossi 99c], this site does not explore the "Set Based Navigation" pattern (which is embodied in the "Navigation Context" primitive). When looking at a particular book, it is possible to navigate to several related books, which are accessible through indexes: "books that other customers that have purchased this book have also purchased" ("related" index); "books by the same author") ("author" index); "books with similar subjects" ("subject" index); "books recommended by auction and zShops participants" ("auctions" index). Regardless of which index is used, once the user has navigated from the index to the book node, context information is lost.

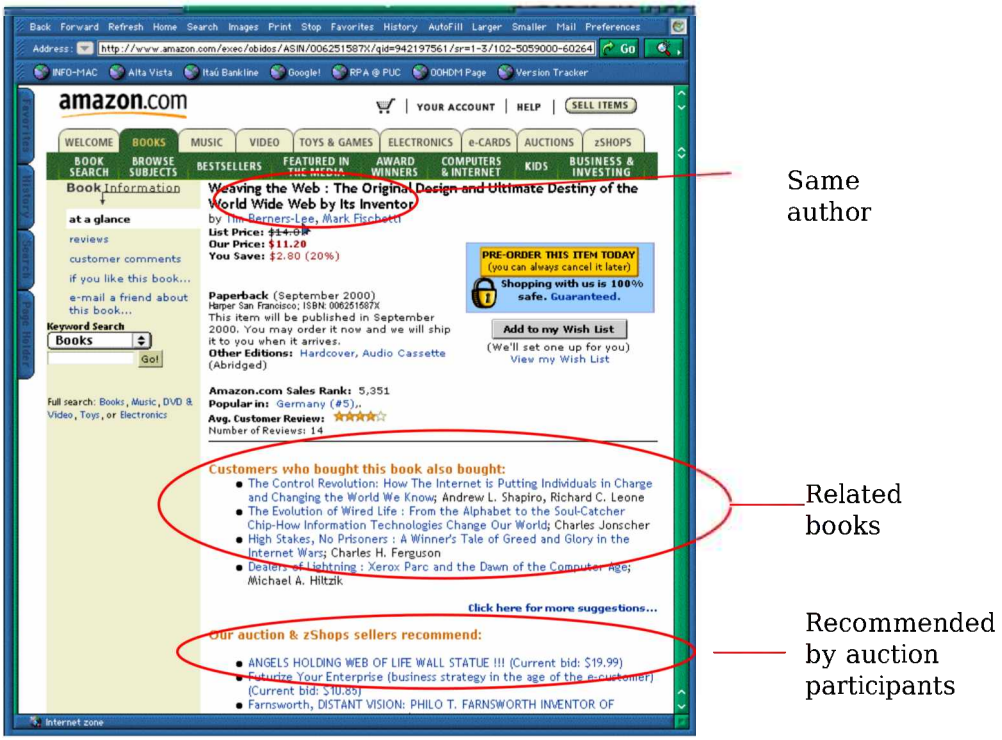


Figure 7 – An example of a "Book by title" instance. The related books are accessible through a variety of indexes, as indicated. Only the top screenful of this (long) page is shown.

The main instantiation mappings for the Amazon.com website are shown in Figure 11. Notice that a single generic index, "similar property", was instantiated into four different indexes, "author", "subject", "related", and "auction". "Generic References" are mapped onto a simple set of indexes "Publication: Recommended", which give the recommended books by several popular publications; the actual text of the recommendations are not included, hence the absence of the "Generic Reference" context itself.

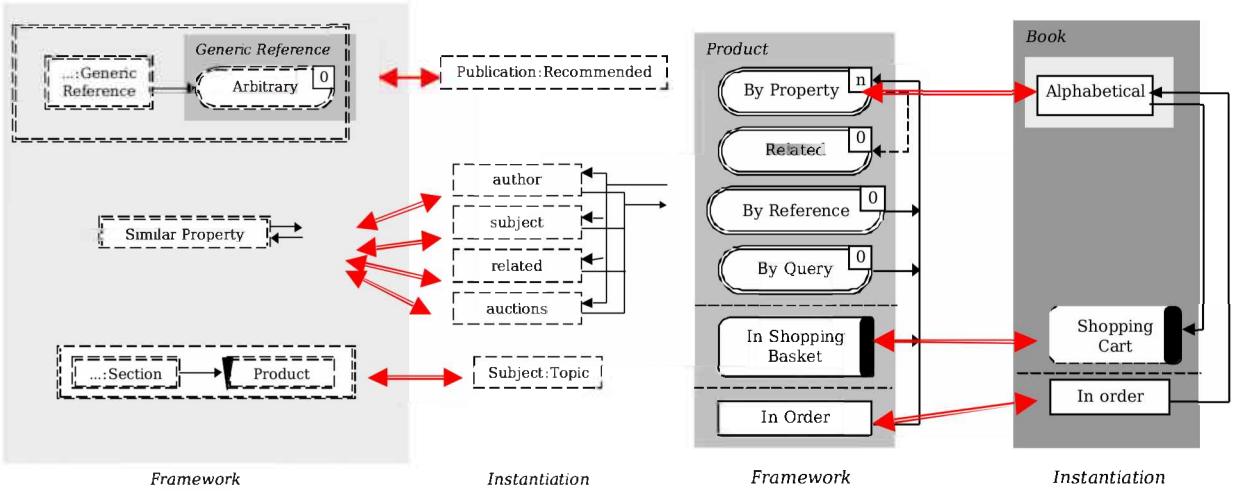


Figure 8 – The instantiation mapping between the application framework context diagram in Figure 5 and the Amazon.com website shown in Figure 6. Only the most interesting mapping for access structures is shown.

For those readers familiar with online stores, it can be noticed that the approach taken in Amazon.com is present in many other online stores; resulting in very similar navigation diagrams; as an example, we cite http://www.etoys.com.

In order to show that this application framework is quite generic, we have also instantiated it for Gap's online store, http://www.gap.com. This site has a straightforward but quite effective navigation architecture, as can be seen in the diagram shown in Figure 9.

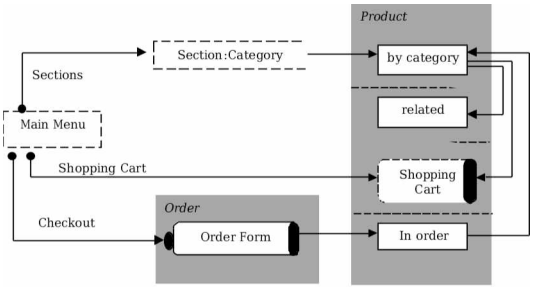
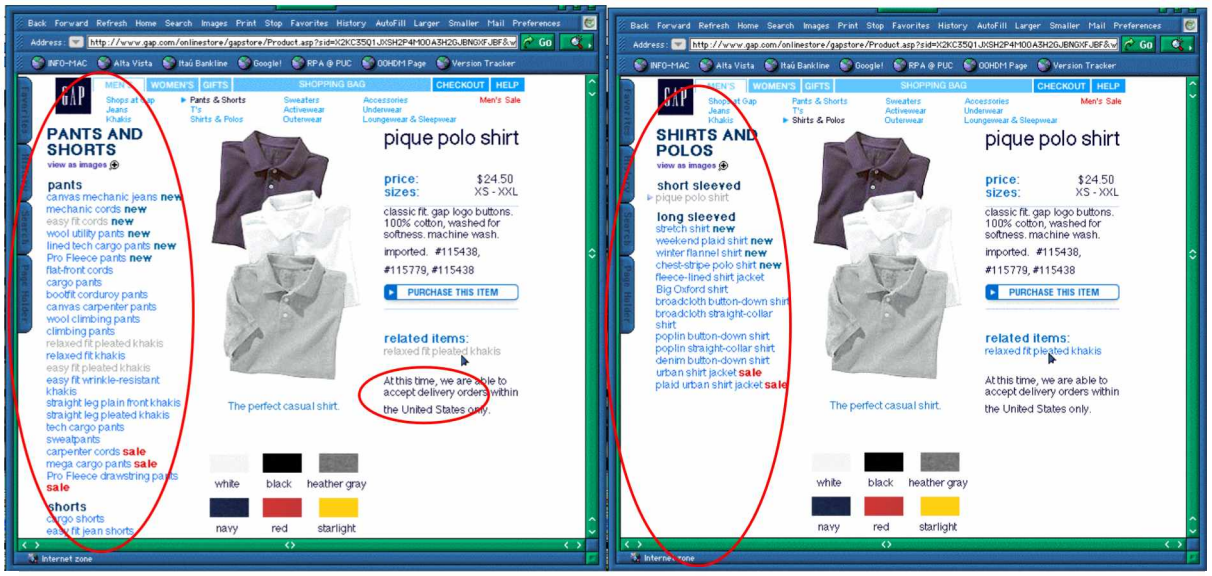


Figure 9 – The context diagram for the Gap Online Store, http://www.gap.com

This site has an interesting use of "Set Based Navigation", as can be observed when navigating in the "related Product" context. Figure 10 shows the same product, "pique polo shirt", in two different contexts: (a) "Related Products", since it is related to "relaxed fit pleated khakis", in category "Pants and Shorts"; and (b) "Product by category". Notice that the index of the original context ("Pants and Shorts" and "Shirts and Polos" remain on the left side of the screen).



(a) (b)

Figure 10- An example of a Product – "pique polo shirt" in two different contexts: (a) related product to "relaxed fit pleated pants"; (b) in the "product by category" context.

A comparison between the framework context diagram in Figure 5 and the instantiation in Figure 9 is shown in Figure 11.

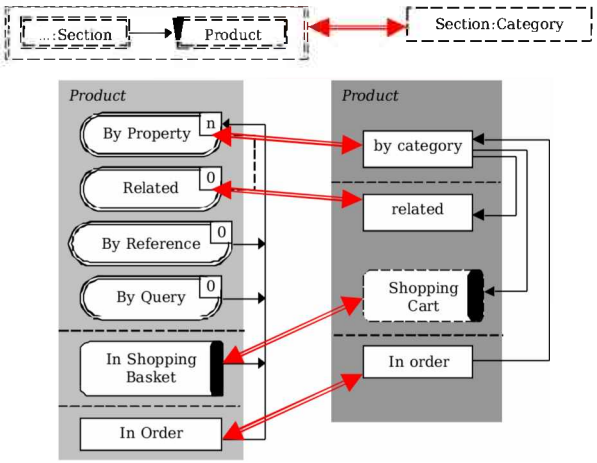


Figure 11 – The instantiation mapping between the application framework context diagram in Figure 5 and the Gap.com website shown in Figure 9. Only the most interesting mapping for access structures is shown at the top.

6 From Design to Application Frameworks and to Web Applications

Web design frameworks help to specify the abstract architecture of a family of Web applications. A framework includes the specification of both the common aspects of applications in the domain and the hot-spots where the specificities of a particular application are accommodated. Web design frameworks are powerful because they are not language or environment-dependent, i.e. we can use the framework to produce Web applications in different settings, including non-object oriented ones.

There are many different alternatives to produce running Web applications for a given framework. In this section we briefly discuss two of them: mapping the design framework onto an application framework (and then instantiate this framework), or instantiating the design framework into an OOHDM model, and then implementing the resulting model in the Web.

6.1 Web Design Frameworks Mapped to Application Frameworks

We have designed an object-oriented architecture that allows designers to implement Web application frameworks for specific domains (an early version is discussed in [Garrido96]; an alternative version is described in [Pizzol 99]). This architecture (and its Java implementation) contains classes that support the core OOHDM primitives (nodes, links, indexes and contexts); these classes can be plugged into domain specific classes (Products, Orders, etc) to improve their behavior with navigation functionality. Using this architecture, a designer should implement the generic conceptual model using an object-oriented programming language (e.g., Java), and for each particular application either sub-class or instantiate the domain classes and connect them with the OOHDM-specific classes that were derived from the generic navigational schema.

In this architecture we decouple the domain and navigational model from the components that

provide dynamic content generation on the Web (ranging from CGI/ISAPI to ASP/JSP). In this way a Web application framework can be designed to be independent of particular industry technologies (and can thus evolve seamlessly).

In this architecture, the OOHDM server is focused on providing the ability to access nodes in different contexts, managing the navigation spaces and linking among nodes. The HTML rendering task is performed on the webserver side by either a custom third-party CGI/ISAPI module or dynamic html pages servers like ASP or JSP.

6.2 Instantiating a Design framework into a Web application using a development environment

Another possible implementation of a particular Web application can be obtained by directly implementing an instantiated Web design framework using standard Web tools. We have been using OOHDM-Web [Schwabe 99] for this purpose. In OOHDM-Web, a complete OOHDM design is represented using special purpose data structures, which are nested lists of attribute-value pairs. These data structures contain class definitions (including InContext classes), navigation context definitions, access structure definitions and interface definitions. These definitions include the description of database entries that store instance data.

Context definitions comprise the query definition that selects the elements that belong to the context; the same is true for access structure definitions. Interface definitions are mixed html templates, one for each class in each context where it appears. The mixed HTML template intersperses pure HTML formatting instructions with function calls to a library of pre-defined functions that are part of the OOHDM-Web environment. These functions allow retrieval of object attributes, or reference to other objects in specified contexts. Reference functions are defined in such a way that, when activated by the user, they cause the exhibition of the destination object in the appropriate context, using the template defined for that context.

Following the same approach, we have defined OOHDM-Frame in a similar way, substituting generic definitions for the concrete ones whenever necessary. The resulting representation describes the generic design of the framework in question. The instantiation process will substitute the generic definitions in the framework by the definitions (using the OOHDM-Web representation) of their corresponding instantiated elements. For example, a generic class-derived context can be substituted by two (or more) class-derived contexts in the instantiated framework; this is achieved by actually replacing, in the data structure that describes de framework, the generic context description by the descriptions of the two concrete contexts, using the OOHDM-Frame format.

At the end of this process, when all hot-spots have been plugged into the corresponding concrete application elements, the resulting data structure is a valid OOHDM-Web representation of the final instantiated application, ready to be used. Our current implementation does not automatically support all constraint verifications, which must be done manually by the designer when instantiating the framework.

7 Concluding remarks and Further Work

In this paper we have introduced Web Design frameworks as a novel technology to further design reuse in Web applications. A Web framework contains the specification of both the behavior and navigational structure of a family of Web applications in a particular domain. We have also introduced OOHDM-Frame, a concise syntax that allows expressing generic OOHDM models that conform to a framework specification. We have shown how to instantiate a design framework into a particular application using the domain of online stores as an example. We finally showed that Web design frameworks can be mapped in a straightforward way into application frameworks by showing a specific architecture.

This architecture allows a designer to implement an object-oriented framework for a particular application domain in much the same way he would do it if the applications were not supposed to run in the Web. He could then plug his application classes into Web specific components (implementing nodes, links and contexts) in order to deploy a running Web application. Web design frameworks can be also directly mapped into an application by using the OOHDM-Web environment

One of the most (if not the most) important architectural components in Web Design Frameworks are (generic) Navigational Contexts. Contexts are recurrent patterns in Web applications as they usually deal with sets of similar objects (products in a store, paintings in a museum's room, etc). The notion that patterns contribute to define the architecture of complex applications is not new [Johnson94] though it is just being perceived in the Web community. We are now incorporating other navigation patterns into OOHDM-Frame to enhance its expressive power. In particular, we are defining a notation for Landmarks and News [Rossi99b].

Landmarks help to indicate well-known entry-points for navigation. For example, in an online electronic store such as Amazon.com each sub-store (Music, Video, Zshops, Auctions) are accessed from everywhere in the site. An OOHDM-Frame specification for Landmarks will allow defining both generic and specific Landmarks to be used by the implementers of a particular online store.

The News pattern meanwhile shows how to deal with sites in which new information (products,

services) are constantly added. An OOHDM-Frame specification for News would indicate how those news will be presented, where they come from (in the conceptual schema) and how they can vary in different applications.

We are also improving our support architectures for Web Design Frameworks; we strongly believe that development, delivery and maintenance times in the Web domain require reuse-centric approaches. The systematic reuse of semi-complete design structures, as described by Web design frameworks is a key approach for maximizing reuse in Web application development.

8 References

[Cowan 95] D. D. Cowan, and 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, pp. 229-243, March 1995

[Fayad99] M. Fayad, D. Schmidt and R. Johnson (editors): "Building Application Frameworks", Wiley 1999.

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

[Garrido96] A. Garrido and G. Rossi "A Framework for Extending Object-Oriented Applications with Hypermedia Functionality". The New review of Hypermedia and Multimedia. Taylor Graham, vol. 2, 1996, pp. 25-42.

[Garrido99] A. Garrido and G. Rossi: "*Capturing hypermedia functionality in an object-oriented framework.* " In Object-Oriented Application Frameworks (R. Johnson, M. Fayad editors). John Wiley 1999.

[Johnson88] R. Johnson and B. Foote: "Designing reusable classes". Journal of object-oriented programming" 1(2), 22-35, 1988.

[Johnson94] R. Johnson and K. Beck. "Patterns generate architecture". In Proceedings of the European Conference on Object-Oriented Technology (ECOOP94).

[Pizzol 99] Pizzol, A.M; Schwabe, D., "A Java Framework for Implementing OOHDM Designs", Proceedings of the V Brazilian Symposium on Hypermedia and Multimedia (SBMidia 99), Goiânia, Brazil, May 1999 (In Portuguese).

[Pree94] W. Pree: "Design Patterns for object-oriented software", Addison Wesley, 1994.

[Rossi95] G. Rossi, A. Garrido and S. Carvalho: "Design Patterns for Object-Oriented Hypermedia Applications" , In Program Languages of Program Design 2, pp. 177-191, 1996, Addison Wesley.

[Rossi 95b] G. Rossi, D. Schwabe, C. Lucena and D. Cowan: "An Object-Oriented model for designing the Human-Computer Interface of Hypermedia Applications". Proceedings of IWHD'95 pp. 131-152, Springer Verlag,.

[Rossi96] G. Rossi, A. Garrido and D. Schwabe: "Towards a Pattern Language for hypermedia applications" Proceedings of PLoP'96, Allerton, USA, 1996. Available at: http://www.cs.wustl.edu/~schmidt/PLoP-96/workshops.html.

[Rossi97] G. Rossi, D. Schwabe and A. Garrido: *Design Reuse in Hypermedia Applications.* 8th ACM Conference on Hypertext Technology. Southampton, England 1997 , pp. 57-66, ACM Press

[Rossi 99a] G. Rossi and A. Garrido: "Capturing Hypermedia Functionality in an Object-Oriented Framework", in Building Object-Oriented Application Frameworks, Wiley 1999.

[Rossi99b] G. Rossi, F. Lyardet and D. Schwabe: "Patterns for designing navigable spaces", in *Pattern Languages of Programs 4*, Addison Wesley, 1999.

[Rossi99c] G. Rossi, D. Schwabe, F. Lyardet: "Improving Web information systems with navigational patterns" International Journal of Computer Networks and Applications, May 1999.

[Rossi99d] G. Rossi, D. Schwabe, F. Lyardet: "Web application models are more than conceptual models". Proceedings of the First International Workshop on Conceptual Modeling and the WWW, Paris, France, November 1999.

[Schwabe95b] D. Schwabe and G. Rossi: "The Object-Oriented Hypermedia Design Model (OOHDM)", Comm ACM, pp. 45-46, August 1995, ACM Press.

[Schwabe96] D. Schwabe, G. Rossi and S. Barbosa. "Systematic Hypermedia Application Design with OOHDM", Proceedings of the ACM International
Conference on Hypertext (Hypertext'96), Washington, March, 1996, 116-128, ACM Press.

[Schwabe98] D. Schwabe, G. Rossi: "An object-oriented approach to web-based application design". Theory and Practice of Object Systems (TAPOS), Special Issue on the Internet, v. 4#4, pp.207-225, October, 1998.

[Schwabe 99] Schwabe, D.; Pontes, R. A.; Moura, I.; "OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW", ACM SigWEB Newsletter, Vol. 8, #2, June 1999.

[UML97] Rational Corporation: The unified modeling language. Documentation set. In http://www.rational.com/uml.