Journal of Web Engineering, Vol. 8, No. 4 (2009) 371–404

SYSTEMATIC IMPROVEMENT OF WEB APPLICATIONS DESIGN

ALEJANDRA GARRIDO GUSTAVO ROSSI
LIFIA, Fac. de Informática, Universidad Nacional de La Plata
50 y 115, La Plata, CP 1900, Buenos Aires, Argentina
{*garrido, gustavo*}*@lifia.info.unlp.edu.ar*

DAMIANO DISTANTE
Faculty of Economics, Tel.M.A. University
Via Santa Caterina da Siena 57, 00186 Rome, Italy
damiano.distante@unitelma.it

Received February 15, 2008
Revised August 31, 2009

Web applications are usually characterized by a rapid development process and continuous evolution. This evolution implies growth of the graph of pages and navigation paths, as well as new functionality and new data made available to the user. Measurement of the system usability, periodic or after a time of growth, is crucial to the system's evolution in the line of design maturity and to prevent the system from decay. This kind of evolution is one of the main practices of agile methods, in which design improvement occurs daily during development and often during maintenance.

In this paper we propose a list of changes for the design models of a Web application and when they may be applied, in order to improve the application's usability while preserving its functionality. The purpose of the proposed changes is to improve the maturity of the design instead of adding new features; we call them *Web design improvements*. This paper starts a catalogue of Web design improvements for the navigation and presentation models of a Web application. Since changing these models has direct impact on the user experience with the application, our Web design improvements aim at improving the external quality and user experience.

Keywords: Web applications, Web patterns, agile methods, navigation model, presentation model, usability.

1 Introduction

It is not a novelty that Web applications must evolve fast; their evolution is driven by a myriad of different factors: new emerging requirements, such as adding services or information types; old requirements evolving according to users' feedback; new technology giving the chance to change the application's look and feel or interaction style, etc.

Many times, however, evolution is only driven by the developers' reflection on the application structure, behavior and/or code [1]. This kind of evolution is crucial to prevent an application from becoming unmaintainable and buggy, as supported by Lehman's law of "Declining Quality" ("Programs will be perceived as of declining quality unless rigorously maintained and adapted to a changing operational environment") [2]. In these cases, the application is modified not to add new functionality but to improve its maintainability and extensibility for future and eventual additions.

Most Web application design methodologies formalize the design of a Web application by means of three models: the *application* (or *content*) *model*, the *navigation model*, and the *presentation model* [3, 4, 5]. The application model defines the contents of the application and its behavior. The navigation model defines the information units of consumption (nodes) for the user and the navigation paths (links) between units. The presentation model defines the abstract user interface. The implementation phase that follows the design will aim to realize a Web application reflecting the design choices defined by these three models. As such, it is not surprising that the quality attributes, both internal and external, of the final application depend on these design choices, and that by properly changing the design of the application we can improve those quality attributes.

We come to the focus of our work: in this paper we are interested in defining small and systematic changes to the navigation and presentation design models of a Web application that can improve external quality attributes, particularly usability and user experience, while preserving behavior. We call these changes *Web design improvements* (*WDIs*). Particularly, WDIs for the navigation model may modify the navigation structure of the application while preserving the reachability of the information and operations available in existing nodes. Similarly, WDIs for the presentation model may modify the look-and-feel of pages but preserve the available operations and their semantics. WDIs may also cause the application of a Web pattern like those described in [6]. Moreover, WDIs not only produce internal changes to Web models but may affect the user experience with the application. In our research, we are particularly interested in these kind of changes that, even preserving the behavior of the application model and the navigability of its features, may systematically improve its *external* qualities such as usability.

As an example that we will elaborate later in the paper, we show in Figure 1 and Figure 2, two different versions of an index from the Amazon bookstore. The first index points to a set of CDs by just exhibiting their titles and artists; meanwhile, the second gives much more information on each CD. The reader may notice that the main intended functionality of the two indexes, i.e., collecting a set of CDs and enabling navigation to pages that give details on each of them, remains unaltered. However, the second index can be considered as an extension of the first one, obtained by applying some changes (enhancements) to it (enriching its contents). This is the kind of design improvement in which we are interested.

Fig. 1. A simple index.

In this paper we characterize WDIs and present a set of them to illustrate our ideas. Particularly, our research is aimed at:

Fig. 2. An enriched index.

- Defining a catalogue of navigation and presentation WDIs and when they may be adopted to improve the external quality of a Web application.
- Analyzing the impact of WDIs on the application's usability, particularly when they introduce Web patterns.
- Analyzing dependencies between navigation WDIs and presentation WDIs.

The remainder of the paper is structured as follows. In Section 2 we review some background concepts and related work. In Section 3 we describe, using the OOHDM notation, an electronic-store that will serve as a running example to illustrate our WDIs. In Section 4 we motivate and characterize WDIs. Section 5 defines navigation WDIs and describes in detail some of them. Section 6 does its part for presentation WDIs. Section 7 shows how WDIs can be composed to create an improvement with a larger impact, and Section 8 provides some concluding remarks and further work. We also include two appendixes: Appendix A provides a quick reference for the Web patterns that we cite, and Appendix B lists all WDIs of the navigation and presentation catalogs together with their purpose, which serves as a quick reference and also presents those WDIs that are not discussed in the paper.

2 Background and Related Work

In this section we discuss some related work on Web design patterns (since they are the target of most WDIs), on refactoring and refactoring to patterns (because they are also changes that preserve behavior), and explore some background concepts on Web applications' design.

2.1 Web design patterns

Design patterns came out in the early 90s as elegant solutions that experts would apply to solve a recurrent problem in software design [7]. Since then, we have seen plenty of catalogues of design patterns, code patterns, interface patterns, and even hypermedia and Web patterns. The concept of Web patterns emerged from the application of hypermedia patterns developed

in the late 90s [8, 9] to the Web. Web patterns are similar to design patterns because they address a recurrent (Web) design problem with a general solution that can be instantiated according to the specific application being solved. The reader can find catalogues of Web patterns at [6, 10, 11]. We include in Appendix A a quick reference to the Web patterns cited in this article. The most extensive catalogue of Web patterns can be found in the work of van Duyne et al. [6]. The patterns of that catalogue all go around the concept of customer-centered design. The book even proposes involving customers in the design process, much like agile methods. However, unlike agile practices, maintenance is considered a separate phase after completion of the system, and patterns do not consider an existent design to start with and evolve from. Another characteristic of the patterns in [6] is that they have a large and comprehensive motivation (which is called the "Problem" section) but the proposed solution is usually short and very general, sketched with hand-drawings of the set of pages. In other words, the Problem section of these patterns is very valuable, but the authors do not address the Solution in detail, probably because they did not want to stick to a particular Web methodology, Web design model or technology. Lastly, the patterns in [6] focus mostly on what the customer perceives, i.e., the graphical interface, but are not usually conscious of the navigation model behind it.

2.2 Refactoring

Refactoring was conceived as "*a program transformation that reorganizes a program without changing its behavior*" [12] and was originally applied to object-oriented code [13, 14]. Refactoring was aimed to eliminate "code smells" which are implementation structures that negatively affect system life cycle properties, such as understandability, testability, extensibility, and reusability; that is, code smells ultimately result in maintainability problems. Refactoring was then extended to some imperative and functional languages (e.g., Fortran [15], C [16] and Lisp [17]) as a technique for restructuring programs. Following the direction of code refactoring, Ricca and Tonella have worked on code restructuring for Web applications [18]. They define different categories of restructuring, like syntax update, internal page improvement and dynamic page construction, and specify transformations that apply over HTML, PHP and JavaScript, which are some of the languages used for Web development. They have some of their restructuring rules implemented in the DMS software reengineering toolkit [19]. Similarly, Harold's book on refactoring of HTML code [20] focuses on refactorings that allow upgrading a site to web standards like XHTML and cascading style sheets.

The idea of refactoring has also been recently applied at the level of design models, giving rise to the concept of model refactoring [21, 22, 23]. New refactorings can appear at the model level that are not apparent from the code [21]. In model refactoring, changes are usually applied on static UML class diagrams, but they have also been proposed for state diagrams and activity diagrams [21, 24]. In each case, the refactorings are applied to different artifacts and the behavior to be preserved is defined for each particular model, or not defined at all. In general, there is no consensus about the semantics associated to the word "behavior" and how it is proven that "behavior" is unchanged by refactoring, since it is usually very hard or too expensive to prove [25].

Model refactoring has also been proposed for the navigation model of a Web application [26]. In their work, Cabot and Gomez present refactorings over links, pages and navigation

paths, considering a navigation model constructed with the WebML methodology [27]. In WebML, the application model is an entity-relationship diagram, and the navigation model based on it is constructed with pages and links that can have associated read or modification (insert/update/delete) operations over the entities. Cabot and Gomez propose refactorings that preserve that kind of operations. Examples of their refactorings are "Add Link", "Parallel Merge" (of pages) and "Head-Merge of Navigation Paths". A downside of this work is that, other than reads and writes over a database, it does not support complex operations as those found when an underlying object-oriented application model is used (like in UWE [3], UWA [4] and OOHDM [5] methodologies - see Section 2.4).

In an earlier paper ([28]), we have described how the ideas behind refactoring could be also applied to the models of a Web application. Here we extend those ideas, generalizing also from the concept of refactoring to include changes that not only improve internal structure but that are mainly driven by the improvement on usability. Related to the improvement on usability, we have also shown how to integrate our changes with a well-known measurement and evaluation framework to demonstrate external quality improvement [29]. Moreover, each WDI might emerge from an evolution cycle assessing the quality of the site, for example using the strategy of [30].

2.3 Refactoring to patterns

With the outcome of agile methodologies, developers move away from over-engineering an application and from applying design patterns whenever possible. Instead, they start with simple designs and apply patterns only when their solution adds flexibility and not complexity [1, 31]. This does not mean that there is a tendency to under-engineering, which is also very dangerous, but to prevent from creating overly complex designs that are too difficult to maintain [1, 31].

Refactoring comes to help keeping the balance between over and under-engineering [31]. The development process starts with a simple design and, when more flexibility is later needed, this design is modified to incorporate the patterns that solve the specific problem. A well-known example of a refactoring that introduces a pattern is "Form Template Method" [31], which explains how to change the existent code to introduce a "Template Method" [7].

With the same spirit of "Refactoring to Patterns" [31], we propose WDIs to introduce Web patterns in an existent Web model only when they are needed [28], e.g. because a problem in the usability of the Web application was found. Paraphrasing the Gang-of-Four [7] and Fowler [14], Web patterns provide the targets for our WDIs; i.e., they represent the solution we want to arrive at, starting from somewhere else. For example, "*Add Processing Page*"(6.2.3) is the WDI that introduces the pattern "Processing Page" [10]. We describe the Web patterns that direct each WDI in its Motivation part. We then describe in the Mechanics the steps necessary to introduce the pattern in an existent navigation or presentation model.

2.4 Web engineering life-cycle

In this paper we adhere to a model-based approach for Web applications development. This approach is the one basically agreed on by most mature development methodologies, like WebML [27], UWE [3], UWA [4], WSDM [32], OOWS [33] and OOHDM [5]. After requirement elicitation, a Web application is usually designed in a three stage process that defines an *application model*, a *navigation model* and a *presentation model*. Particularly, we base our

work on the models produced by the OOHDM methodology [5]. In this section, we briefly describe OOHDM models, particularly the navigation and presentation models, since they will be the targets of our WDIs.

The *application model* (also known as domain or content model) describes the structure of the application's contents, i.e., the content types and their relationships, as well as the possible operations on these contents. In OOHDM, the application model is an object-oriented model comprised of classes and their associations, and it is described with a UML class diagram.

The *navigation model*, which is essential for Web software, specifies the units of consumption of the application contents and operations (i.e., navigation nodes), and the navigation paths through contents, (i.e., links, indexes, etc.). In OOHDM, the navigation model is defined as a view over the application model, with nodes as logical windows on application model classes, and links as the hypermedia realization of application model associations [34]. Moreover, the navigation model is composed of two diagrams: the *navigational contexts diagram* and the *navigational class diagram* [5]. The former defines navigational contexts: sets of nodes and links that organize the navigational space to help complete the intended uses of the system. The latter describes navigational classes: nodes and links. Section 3 shows examples of both OOHDM navigational diagrams.

Nodes contain perceivable data (attributes), available operations over the data, and anchors that allow triggering links. Links connect node classes and reflect the associations of the application model that are intended to be explored by the final user. Indexes are a particular type of composite node defined to enable one-to-many navigation. Each entry in an index may be defined to contain some attributes of the target node and/or an anchor to navigate to it. Alternatively, each entry may be a full-fledged node. Navigation topologies like guided tours can also be defined.

Since nodes are views over application model classes, node contents are derived from application class attributes and operations. Moreover, it may be necessary to combine features from different application classes to describe a node class. The mapping from application class attributes to node attributes is specified in the navigation class diagram with a query language similar to the one in [35], and the application model remains unaware of the mapping. For example, suppose we have two application classes, `Client` and `Order`, defined as shown at the top of Figure 3. In the navigation model (bottom half of Figure 3), there is a single node class named `Order` with attributes combining information from both application classes. There will be a node `Order` for each order '`o`' in the application, with node attributes `number`, `date` and `amount` showing the values for those attributes in '`o`'. In the case of the node attribute `name` (and similarly for `address`), the query statement specifies that it is derived from the attribute of the same name of the Client '`cl`' associated with '`o`'. We do not describe the query language further because the WDIs proposed in this paper do not apply changes to the mapping. The interested readers can refer to [5].

Referring back to the navigational context diagram, each navigational context in the diagram represents a set of nodes related by some concept, e.g., Books by Author. The concept that relates the nodes may be a node attribute or a related class. The representation of a navigational context is depicted in Figure 4. The grey box to the left represents the elements in the set, while the inner white box represents the concept that relates them. A navigational context card (shown in the right-hand side of Figure 4) is also associated to describe the

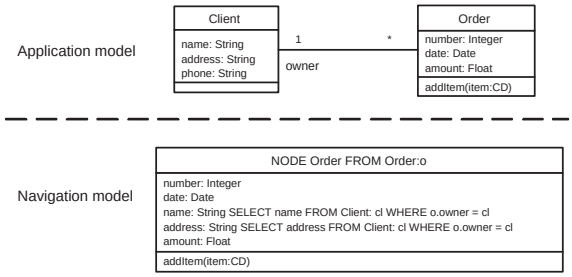

Fig. 3. Example of a mapping from application to navigation model

attributes that will appear while navigating the nodes inside this context. Section 3 presents concrete examples of navigational contexts.

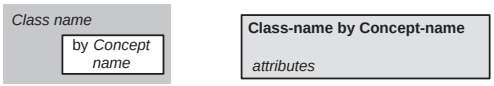

Fig. 4. Representation of a navigational context

A concept similar to navigational contexts is also provided by UWA, namely, the concept of Navigation Cluster [36]. A cluster identifies a set of nodes that can be navigated across, the accessibility relations that support navigation between them, and the navigation pattern describing the actual way navigation across nodes is allowed. Clusters of nodes are generated from Entities (application classes), from Semantic Associations (Entities involved in some relationships) or Collections (subsets of the instances of one or more Entities, selected according to a certain criteria). Particularly, contexts in OOHDM are equivalent to clusters generated from collections in UWA.

Finally, the *presentation model* is defined as a layer on top of the navigation model, mapping nodes to pages and defining for each page the interface objects that display information or facilitate user actions and navigation. In OOHDM, this model is also called Abstract Interface Design, and is described with a graphical notation called Abstract Data View (ADV) [37]. ADVs contain different types of abstract widgets that either display node attributes (e.g. multimedia fields), trigger node operations or enable navigation [34], and may also contain other ADVs. OOHDM uses ADVs to express: i) the layout of pages and page sections in terms of abstract widgets and aggregation of ADVs, ii) the connection with the navigation model, and iii) the interface transformations that occur as a result of user interaction [5].

In terms of the connection with the navigation model, a page in an OOHDM presentation model may map one or more navigation model nodes, and conversely, a node may be mapped (not split) into several pages. Section 3 also shows an example of an ADV.

3 Example

The WDIs that we propose are generic, but in this paper we describe them in the context of an electronic store like Amazon and Barnes&Noble. We chose this kind of application for various reasons: first, Amazon is a very good example of a website that is frequently improved;

second the changes they apply inspired many of our WDIs; third, they have been previously analyzed and their design decisions well documented (e.g., in [6]).

In electronic stores, users browse and search for articles and add them to a shopping cart. Figure 5 shows a reduced application class diagram and the associated navigation class diagram for a CD-store. The navigation diagram uses the OOHDM notation [34], in which node attributes are specified with query statements, optional values (between square brackets), indexes (using the keyword `Idx`) or lists of items (specified with the keyword `List`).

The example assumes a basic application model with classes `Client`, `Order`, `CD`, `Track`, `Artist` and `OrderItem` as an association class. The navigation diagram shows four node classes: 'Order' (mapping attributes from application classes `Order` and `Client`), 'OrderItem' (showing attributes from `CD`s in an `Order` and providing anchors to them), 'CD' (mapping application class `CD` and holding an index to the participating `Artist`s) and 'Artist' (mapping class `Artist`).

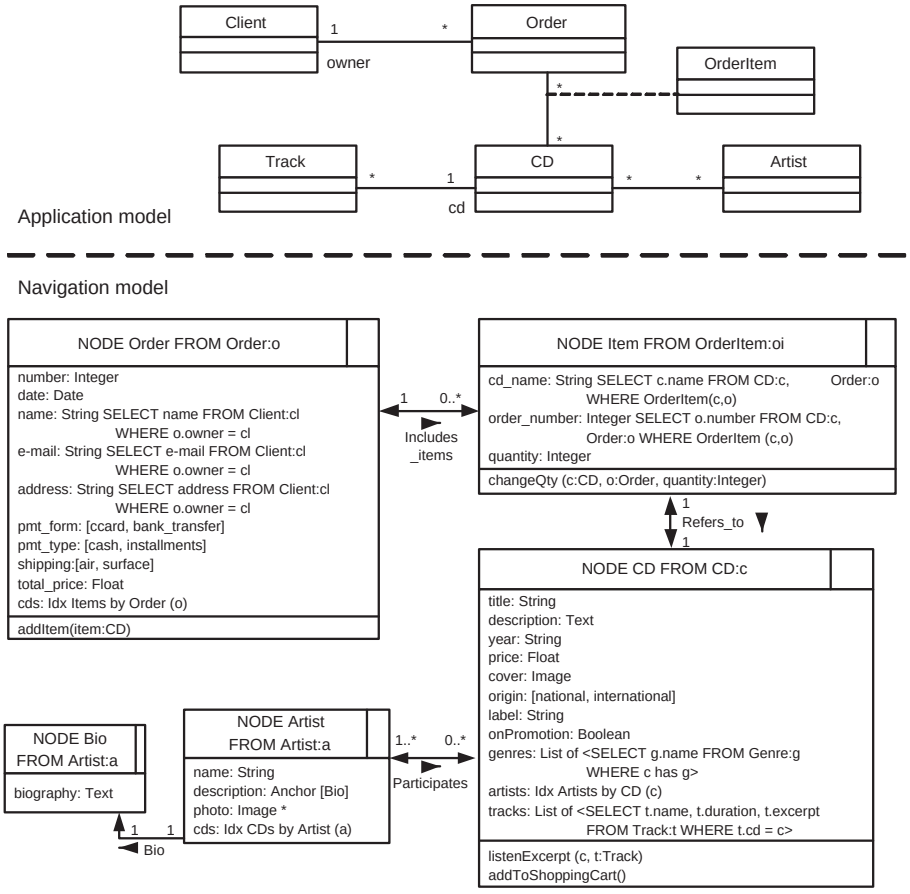

Fig. 5. Navigational class diagram of a CD store

Figure 6 shows the partial context diagram for this example. Both 'CD by Genre' and 'Artist by CD' are contexts, which correspond to sets of elements [34]. The elements making

up each set are described in the grey boxes.

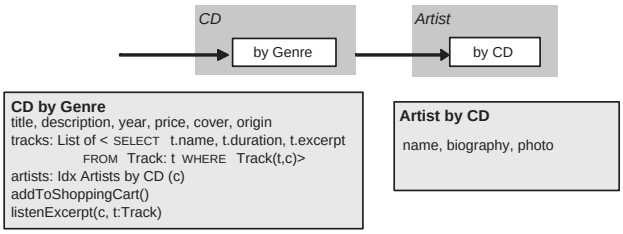

Fig. 6. Partial navigational context diagram of a CD store

Finally, Figure 7 shows a sample ADV for the presentation of a CD. The ADV is composed of a widget for the picture, an anchor to execute the Buy action and two component ADVs: one that presents several details of the CD and the other displaying an index to the Artists that participate in the CD.

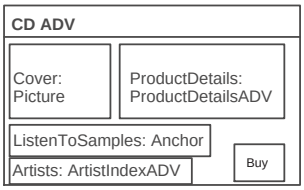

Fig. 7. Abstract interface of a CD

The WDIs that we describe in this paper deal with some features that we consider important for the usability and efficacy of Web applications, particularly, e-stores, which developers should keep improving. We describe just three of these features and the actions needed to maintain them while the application evolves.

1. E-stores should provide meaningful and diverse browsing categories of products. As the site grows, it is important to keep revising categories and subcategories, by creating new subcategories, splitting and/or merging the existing ones.

2. Most users like to have recommendations and customers' rating at hand, but as more information is added to a product and to product indexes, pages get easily cluttered and become confusing, so the interface should be redesigned accordingly.

3. As new services are incorporated to an e-store, operations become longer and need more steps. The Web application should grow accordingly to provide feedback on the current step, backtracking and navigability from/to the shopping cart. At the same time, short-cuts and "buy with one-click" options become important for experienced users.

4 Improving the Design of a Web Application

There are basically two levels of abstractions or approaches in which the design of a Web application may be improved while preserving its functionality. An approach is to apply

changes at the code level in order to increase maintainability and extensibility of the application. This is the approach of code restructuring taken by Ricca and Tonella [18] and that of HTML refactoring [20] (see Section 2). Refactoring a Web application at the code level is similar in intent and structure to conventional code refactoring [14], though it deals not only with object-oriented code but also with code in typical Web languages such as HTML, XML, JavaScript, etc.

A second approach to design improvement of a Web application is to apply changes at the model level. According to the generic development approach for a Web application described in Section 2.4, this means to apply changes to the application, navigation and/or presentation models. For object-oriented Web design methods like OOHDM, the application model of a Web application is an object-oriented model and as such, changes at this level are described by model refactorings like those in [21, 22], which impact mainly on internal qualities such as maintainability.

On the navigation model we could also apply changes targeted at improving internal qualities. Examples of this kind of changes are the refactorings described by Cabot and Gómez like "Remove redundant navigation paths", "Head-Merge of navigation paths" and "Tail-Merge of navigation paths" [26]. The goal of our WDIs is instead to apply changes to the navigation model that impact on the user experience with the application, thus *improving external qualities* such as usability. As in object-oriented refactorings, we aim to eliminate those "bad smells" in the navigational and interface structure of Web applications which influence negatively in usability.

An example of the kind of changes we propose for the navigation model is '*Move Node Attribute*' from a source node to a destination node. The motivation for applying this change arises when the information provided by the attribute is no longer useful or is unrelated to the operations provided by the source node, and therefore the attribute is moved to where the user will find it more useful, thus improving usability.

The same idea applies to WDIs of the presentation model. At this level it is easier to see that a change cannot be internal, since this model deals with the appearance of pages. Changes at this level include replacing a widget for a more appropriate one, or introducing "bread crumbs" [6].

Note that our goal is to *change* Web design models that *already exist* in order to *improve their usability*. Improvement gain will always depend on the good judgment of the developer to select the most advantageous changes, i.e., in his ability to detect "bad smells". Nevertheless, the improvement gain can be measured by integrating a mature quality measurement and evaluation method like WebQEM [38] before and after applying WDIs, as we have proposed elsewhere [29] as an approach to systematically find those "bad smells".

Web design improvements preserve the behavior defined in the application model, such as user operations and their operational semantics. Moreover, WDIs for the navigation model (*navigation WDIs*) preserve the reachability of the information and operations made available from existing nodes; meanwhile, WDIs for the presentation model (*presentation WDIs*) must also preserve the visibility of available features.

We have identified a group of concrete navigation and presentation WDIs by considering the properties/aspects defined for a Web application in the navigation and the presentation models, and by examining how successful Web applications usually evolve. Section 5 describes

navigation WDIs and Section 6 describes presentation WDIs. In both sections, we first provide a precise definition of what navigation and presentation WDIs are allowed to change, and then list some WDIs. Each WDI is described using a common template comprising *Motivation*, *Mechanics*, *Examples* and *Related WDIs*. The *Motivation* presents a design problem that the WDI helps solve and/or describes a good solution that the WDI can help reach, and particularly indicating which "bad smells" it avoids. For the sake of conciseness and understanding, we indicate the "bad smell" with a simple explanation, presenting them with a coarse-grained description and do not explain how smells can be classified. We emphasize the description of bad smells for navigational WDIs. The *Mechanics* describe the small steps that will change the design into the target one. Some steps may be the application of another (possibly atomic) WDI, and in that case we are describing a *composite* WDI. The *Examples* show how the WDI could be applied to our running example. Lastly, *Related WDIs* shows dependencies or relationships with other WDIs that are part of the current one or are its consequences.

While Sections 5 and 6 present small or simple WDIs, Section 7 describes more complex WDIs spanning both the navigation and presentation layers and built as composition of smaller WDIs.

5 Navigation Web Design Improvements

5.1 Definition

Navigation Web design improvements apply changes to the navigation model, i.e., according to the OOHDM method, they may change the design elements defined in the navigational class diagram or the navigational context diagram, such as:

- the set of available nodes;
- the contents of a node (including attributes, operations and anchors);
- the set of links between nodes;
- the navigation topology associated to the set of nodes in a context diagram;
- the available navigational contexts;
- the links between navigational contexts.

We consider as navigation WDIs only those changes to the navigation model that preserve the application's behavior. At the navigation model level, the behavior of a Web application is defined by two aspects: (*i*) the set of operations of the application model for which there is a view or mapping to the navigation model, and their associated semantics; (*ii*) the "reachability" of these operations through the available links (i.e., the existence of a path from the home page to the node containing the operation). Accordingly, we provide the following definition.

Definition. *A navigation design improvement may change the elements defined in the navigational class diagram or the navigational context diagram of a Web application but must preserve:*

- *The set of operations defined in the application model that are made available from nodes, and the semantics of each operation;*
- *The "reachability" of the operations through the available links.*

Therefore, navigation WDIs are changes applied to the navigation model that preserve both the behavior defined in the application model (at least the behavior that is visible to the user from the navigation model) and the navigability of the navigation model. Moreover, navigation WDIs should not introduce information, relationships or operations that are not already available in the application model.

5.2 Catalogue

Table 1 shows a list of basic navigation WDIs and some of them, marked with a '*', are described in detail below. Appendix B repeats the list in Table 1, adding the purpose of each WDI as a quick reference.

Table 1. Navigation Design Improvements

Split Node Class *
Merge Node Classes *
Remove Unreachable Node
Move Node Attribute
Turn Attribute Into Link *
Add Node Operation *
Move Node Operation
Add Link *
Remove Redundant Link
Add Category *
Split Category *
Merge Categories
Relate Categories *
Add Index
Change Navigation Topology

5.2.1 Split Node Class

Motivation: The purpose of this WDI is to break a node class that has become too cluttered with information, anchors for links or functionality, by extracting some of the node's attributes into a new node. It may be the case that a node does not present "Clean Product Details" [6] to help customers in their buying decisions. You may also want to introduce the pattern "Overview by detail" [10]. Moreover, this WDI may also be motivated by the application of the refactoring "Extract Class" [14] in the application model. Another motivation is to allow design evolution of the navigation model: a node that has been defined to map more than one application class may become too complex and need to be partitioned, or a node associated to a single application class may be split in order to provide partial but more cohesive views of the class attributes. In this case the bad smell is node cluttering.
Mechanics: The WDI is performed in four steps:

(i) Add a new empty node class.
(ii) For each attribute you decide to move from the original node class to the new one, use the WDI '*Move Node Attribute*' to add the attribute to the new node class. You may also copy some attributes that should be present in both nodes (e.g., the CD title in the example below).

(iii) For each operation you decide to move from the original to the new node class, use '*Move Node Operation*'. You may also decide to copy some operations (e.g., the operation `addToShoppingCart()` in the example below).
(iv) Use '*Add Link*'(5.2.5) to add a bidirectional link (or two links back and forth) between the original and the new node class, in order for the user to be able to access the original overall information and set of operations.

Example: In an e-commerce website, the node class associated to products of the catalogue may be split into a node class with basic or "Above the Fold" [6] information, and a node class with detailed or "technical information". In our running example, we could for example decide to split the node class `CD` defined in Figure 5 in two node classes: `CD_Basic` and `CD_Details`. Figure 8 below shows the original node class to the left of the grey arrow and the resulting node classes to the right. We have first created the new node `CD_Details`, we copied the `title` and moved to the new node the attributes `origin`, `label`, `onPromotion`, `genres`, `artists` and `tracks`. We also copied the operation `addToShoppingCart()` and moved the operation `listenToExcerpt()`. We added two links between the nodes: `Details` and `Basic` with their corresponding anchors. In this case we additionally renamed `CD` to `CD_Basic`.

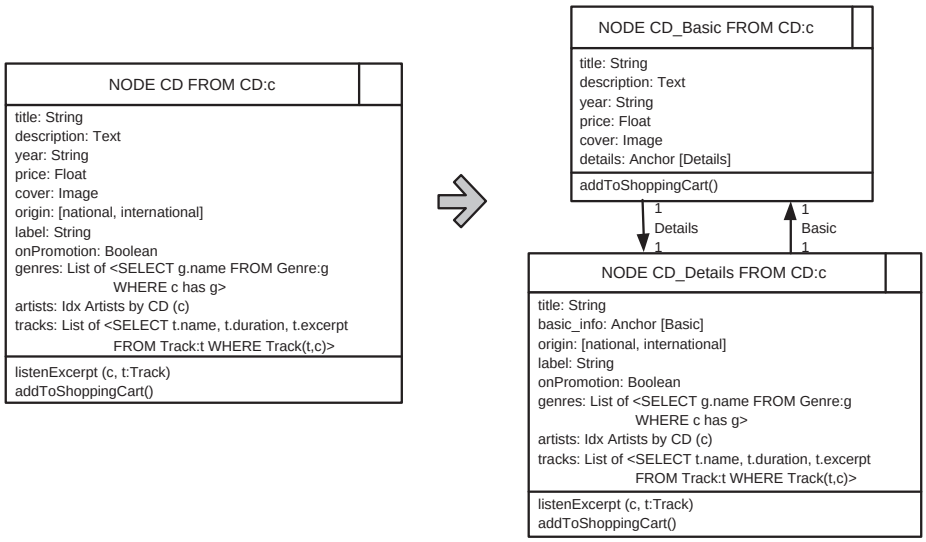

Fig. 8. Example of Split Node Class

Related WDIs: This WDI is a composition of atomic ones like *'Move Node Attribute'*, *'Move Node Operation'* and '*Add Link*'. It may be followed by the application of the presentation WDI *'Split Page'* in the presentation model.

5.2.2 Merge Node Classes

Motivation: This WDI is the inverse of the previous one, and therefore its motivation may be the opposite: to join two nodes that present scattered information. The bad smell here is information scattering. This WDI should be used when evaluations of the Web application show that most users accessing the contents of a node A need to navigate to a node B to

complete their task, such as getting all the available information on a product. If the node A is not already too cluttered, it may be convenient to merge nodes A and B to shorten the navigation path, at least for experienced users. Furthermore, a node B may be also absorbed by another one after the refactoring 'Inline Class' [14] has been applied to the application model class that B was derived from.
Mechanics: This WDI is performed as follows:

(i) Choose the node class (`A`) into which you will merge the other one (`B`).
(ii) Move each attribute from node class `B` to `A` using the WDI '*Move Node Attribute*'.
(iii) Move each operation from node class `B` to `A` using '*Move Node Operation*'.
(iv) If there is a link from `B` to `A`, delete the link and its anchor definition.
(v) For all other links departing from `B`, move to `A` both: their source and the corresponding anchor definition.
(vi) Delete the link from `A` to `B` and its anchor definition (the existence of this link is implied by the motivation of this WDI, since you would not want to merge two unrelated nodes).
(vii) For all other links arriving to `B`, change their target so that they arrive to `A`.
(viii) Consider revising the name of `A` in case it needs to include the name of `B`.
(ix) Delete the node class `B`.

Example: In the example of Figure 5, the node class `Bio` could be merged into the node class `Artist`. Figure 9 shows the nodes before and after the application of this WDI.

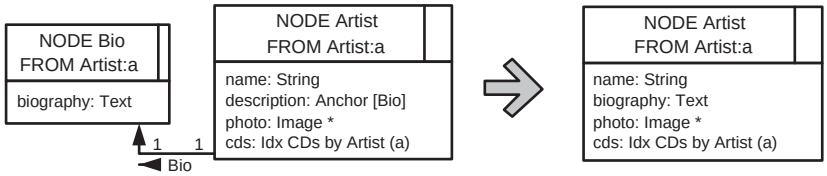

Fig. 9. Example of Merge Node Classes

Related WDIs: This WDI is a composition of atomic ones like '*Move Node Attribute*' and '*Move Node Operation*'. It may be followed by the application of the presentation WDI '*Merge Page*'. If the absorbed node (`B`) had its own page in the presentation model, there are two options: (1) merging `B`'s page into `A`'s page by applying '*Merge Page*' or (2) if `A`'s page would become cluttered, leave `B`'s page but update the source of its contents.

5.2.3 Turn Attribute Into Link

Motivation: Studies on the usage of a Web application may show a need to provide navigation from a piece of displayed information to another. For example, during the process of completing a business transaction, some Web pages may show intermediate results or a succinct review of the information gathered until a certain point (like in "Shopping Cart" [6]). Such Web pages should provide the user with the chance to review the information associated to the intermediate results by means of direct links to the pages showing details on them. One possible bad smell to motivate this WDI is the lack of needed navigation facilities.

Another motivation for this improvement may be to provide "Context-Sensitive Help" [6] or "Embedded Links" [6].

Mechanics: Select the attribute in the source node that better distinguishes the target node. In the navigation class diagram, perform the next two steps:

(i) Use '*Add Link*'(5.2.5) to add a new link with name `link-class-name` from the source to the target node, if the link does not already exists;
(ii) In the source node, replace the definition of the attribute by an anchor definition.

The syntax of an anchor definition in OOHDM navigation model is the following:

```
anchor-name: [link-class-name]
```

Example: A common example occurs when checking the status of the shopping cart during the process of buying some products in an e-commerce website. This WDI may be used to add links from names of products in the shopping cart to the pages showing detailed information about the products. In our running example, we apply this WDI to change the attribute `cd_name` in node `Item` to be an anchor. Figure 10 shows node `Item` before and after applying this WDI, highlighting the new anchor in bold. Note that step (i) was not necessary since the link `Refers_to` already existed.

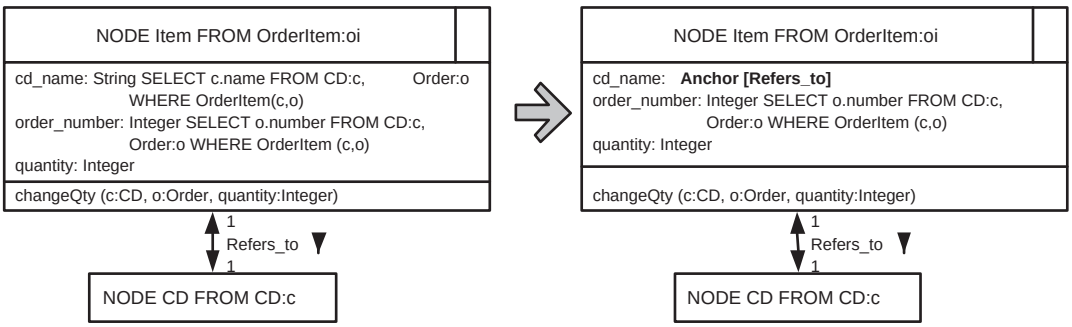

Fig. 10. Example of Turn Attribute into Link

Related WDIs: In the presentation model, the '*Add Interface Anchor*' WDI is required to map the navigation level anchor that was introduced in the node definition.

5.2.4 Add Node Operation

Motivation: Operations should always appear close to the data on which they operate, and Web applications should be designed with that in mind. However, operations may be added later to a Web page for various reasons: as the result of an operation added to the application model because of a new requirement; to speed-up the check-out process of an e-commerce Website; to provide "Printable Pages" [6], etc. The operations may be also added to each entry on an index, so that the user will not need to navigate to the node describing the entry to operate on it. With this WDI we attack the bad smell produced when operations are far from the information on which they work.
Mechanics: The precondition for adding an operation to a node is that the operation should be already available in the application model. If the precondition holds, the operation is added to the appropriate node class in the bottom part of the node definition.
Example: In the Amazon bookstore, the check-out process has evolved over time to allow a considerable speed-up. When an item is added to the shopping cart, the cart has an extra

button that says: "Buy now with 1-Click". Pressing that button allows users to login and retrieve all the information they have specified in a previous order, avoiding the need to re-enter it for every purchase. Figure 11 shows the change in the node `CD_Basic` of our example.

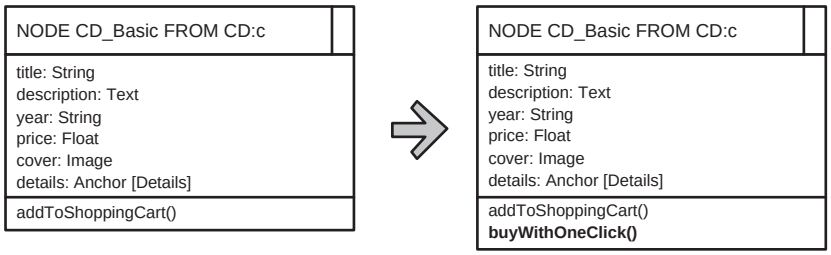

Fig. 11. Example of Add Node Operation

Related WDIs: Adding an operation to the navigation model requires the presentation model to be augmented with an extra widget to dispatch the operation. This may require the ADV for the corresponding page to be restructured, for example using '*Add Page Section*' to group related operations. The addition of an operation that takes some time to process may require the application of '*Add Processing Page*'(6.2.3).

5.2.5 Add Link

Motivation: Adding a link between two nodes provides the user with new navigation possibilities. This could be useful in different cases: to shorten the navigation path between two nodes already reachable from each other; to make reachable a new node added to the navigation model; after applying '*Split Node Class*'(5.2.1) to add a link between the resulting halves. This WDI also solves the bad smell produced by the lack of navigation facilities.
Mechanics: A new link class with name `To-target` between two node classes `sourceNC` and `targetNC` can only be added if the following preconditions hold: (1) `sourceNC` and `targetNC` exist as node classes in the navigation model and (2) `To-target` is not already used in the navigation model. If these preconditions are met, a link class is added in two steps:

(i) Add an association labelled `To-target` between `sourceNC` and `targetNC` in the navigational class diagram; apply the appropriate cardinality and direction to the association.
(ii) Add a suitable anchor in source node class `sourceNC` able to provide access to the new link.

Figure 12 shows a sketch of a portion of the navigational class diagram affected by this WDI, with the old version to the left of the grey arrow, and the new version to the right.
Example: We have used this WDI in the example of the '*Split Node Class*'(5.2.1) to add links between the nodes `CD_Basic` and `CD_Details`.
Related WDIs: A link can be added to a node by turning one of the node attributes into a link; in this sense this WDI can be realized by the '*Turn Attribute into Link*'(5.2.3) one. If the new link is introduced to shorten the navigation path between two existing nodes, it could be that one or more of the intermediate nodes become useless and have to be removed by means of the WDI '*Remove Unreachable Node*'. Finally, if we want to render the new link

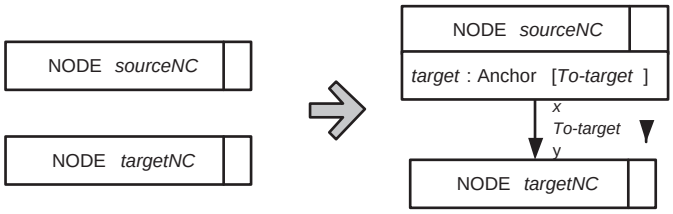

Fig. 12. Application of 'Add Link'

with a graphical widget, the WDI '*Add Interface Anchor*' has to be applied to extend the presentation model.

5.2.6 Add Category

Motivation: When the number of pages in a website increases, users may find it more difficult to find what they need by just following available links. Some navigation structure becomes necessary, to organize related pages in sets or categories. The pattern "Browsable Content" [6] prescribes organizing content in categories that are meaningful to users. Moreover, different ways of categorizing nodes have been proposed: "Hierarchical Organization", "Task-Based Organization", "Chronological Organization" and "Popularity-based Organization" [6]. The involved bad smell in this WDI is bad information organization.
Mechanics: The purpose of this WDI is to add a new way of categorizing navigational nodes, for example, organize the books in an e-bookstore by author. In OOHDM, a new category is created by defining a *navigational context* in the navigational context diagram [5] (see Subsection 2.4).

The precondition of this WDI is that all parties involved in the navigational context must be already present in the application model, namely, the class of elements in the set, the concept (either attribute or class) that relates the elements, and the attributes displayed while navigating the elements.

A new navigational context over a set of elements S related by a concept C is added to the navigational context diagram in two steps:

(i) If there is not an existing navigational context relating the elements in S, add a new grey box to the navigational context diagram labelled *'S'* with an inner white box labelled *'by C'*. If there is an existing navigational context relating the elements in S, enlarge the box labelled *'S'* to contain a new inner white boy labelled *'by C'* and separate the new white box from the previous ones with a dashed line.
(ii) Add a context card labelled *'S by C'* to describe the attributes of elements in S that will be displayed while navigating them in this context.

Example: : In our CD store example, we add other criteria to traverse CDs: 'by Artist' and 'by Order'. Figure 13 shows the resulting context diagram, although we omit the context cards for space reasons.
Related WDIs: The way in which a navigational context is reached is usually through an access structure like an index. Therefore, a WDI that will be usually needed after '*Add Category*' is '*Add Index*'.

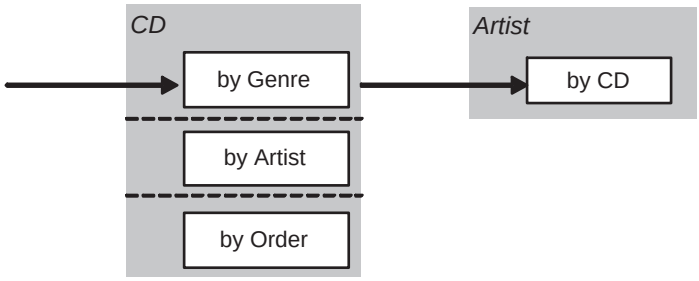

Fig. 13. Example of Add Category

5.2.7 Split Category

Motivation: As a website grows and new nodes are added, categories may become too large. The pattern "Hierarchical Organization" [6] recommends keeping the number of elements in a category under 50. When the number of nodes in a category grows above that, it is convenient to split the category into a number of subcategories. Similarly to the previous WDI, the bad smell is bad information organization.
Mechanics: Similarly to '*Split Node Class*'(5.2.1), this is a composed WDI since splitting a category involves adding new subcategories through the WDI '*Add Category*'. Moreover, indexes to the subcategories may be added through '*Add Index*'. Alternatively, if the refinement occurs in the contents of the application model, the definition of the navigational context remains the same, but the associated index is defined to be hierarchical; this is done in OOHDM by appending a colon at the end [5].
Example: Products of an e-shop should be easy to reach navigating through categories. At the same time, the list of products in a category should not grow too large because it becomes difficult to read and customers get easily frustrated. Barnes&Noble has long subcategory chains to allow easy access to the large book base. While applying this WDI, the developer should keep in mind that adding a new level of subcategories while structuring navigation lengthens the navigation paths towards the final content of the site. Thus, the right balance between navigation structuring and short navigation paths has to be decided.

Continuing with our running example, consider the navigational context 'CD by Genre'. In this context, we have genres like 'Rock', 'Country', 'Classical', etc. We could refine or divide these genres in subsets, for example, dividing 'Rock' in 'Funk Rock', 'Latin Rock', 'Oldies&Retro', and the many types of Rock one can find in Amazon when browsing music. In this case, the definition of the navigational context 'CD by Genre' does not change, but the associated index ('Genres') is defined to be hierarchical by renaming it to 'Genres:'.
Related WDIs: In the presentation model, it may be necessary to '*Provide Breadcrumbs*' while going deep in a chain of subcategories.

5.2.8 Relate Categories

Motivation: If two categories have elements in common, customers should be able to navigate from one to the other and vice versa. Using this WDI we provide "Multiple Ways to Navigate" [6], so as to prevent the site from becoming tedious to use. The bad smell here is the lack of navigation facilities applied to information categories.

Mechanics: The precondition of this WDI is that the navigation model already has the navigational contexts defining the categories to be related. Then, the WDI is performed by using '*Add Link*'(5.2.5) to create a bidirectional link between the navigational contexts.
Example: Continuing with the example presented in Figure 13, we could add a link between the context 'Artist' and 'CD by Artist'. The resulting context diagram appears in Figure 14.

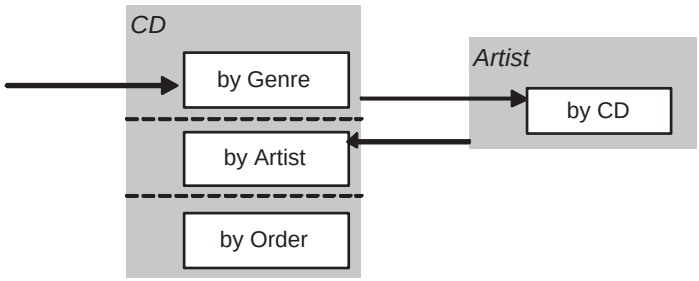

Fig. 14. Example of Relate Categories

Related WDIs: As described in the mechanics, this WDI is realized by means of '*Add Link*'(5.2.5) between navigational contexts. At the presentation layer, it requires the WDI '*Add Interface Anchor*' to be applied to show the new link in both pages.

6 Presentation Web Design Improvements

6.1 Definition

The presentation model includes the following elements that may be the target of our presentation WDIs:

- the type of interface objects (abstract widgets) that compose a page (ADV);
- the arrangement and composition of abstract widgets in an ADV;
- the mapping of nodes into pages;
- the interface transformations occurring as the result of user interaction.

The behavior of a Web application as specified by its presentation model is given by the set of operations that users may trigger in a page and the set of links they can navigate from a page. This means that if a node operation or link anchor has a "view" from the presentation model (i.e., an abstract widget defined in an ADV to trigger it), a presentation WDI should not remove this view. Therefore, we provide the following definition of "legal" presentation WDIs.
Definition. *A presentation design improvement may change the elements defined in the presentation model but must preserve:*

- *The set of available operations and their semantics as defined in the application and navigation models;*
- *The availability of an abstract interface for navigation model features.*

Preserving the "availability" of an abstract interface means that a presentation WDI may not remove abstract widgets from an ADV or an ADV itself, leaving a node operation, link

anchor or a whole node without its presentation view. Nevertheless, ADVs may be split or merged, and some widgets may be replaced by others. Therefore, presentation WDIs may:

- change the type of widgets if the new type preserves the underlying functionality;
- change the arrangement of widgets in an ADV;
- split or merge ADVs;
- add information or operations to a page if they are available in the underlying navigation and application models;
- add or change the available interface effects.

Notice that most navigation WDIs that we enumerated in Section 5.2 will cause other WDIs in the presentation model. For example, '*Add Link*'(5.2.5) requires its corresponding origin (source anchor) to be displayed (using '*Add Interface Anchor*'); in turn, '*Add Category*'(5.2.6) may require to show the current subcategory in deep hierarchies (applying '*Provide Breadcrumbs*'). Meanwhile, presentation WDIs should not change the navigational structure, i.e., splitting a page should not change the navigability of nodes.

6.2 Catalogue

Table 2 shows a list of presentation WDIs and some of them, marked with a '*', are described afterwards.

Table 2. Presentation Web Design Improvements.

Split Page *
Merge Pages
Add Page Section
Merge Page Sections
Move Widget
Replace Widget *
Add Interface Anchor
Add Processing Page *
Provide Breadcrumbs
Allow Changing Category *
Introduce Information on Demand
Introduce Link Destination Announcement
Introduce Scrolling
Split List

6.2.1 Split Page

Motivation: A web page that is found to be too cluttered with information or that requires too much scrolling discourages users and may cause some of its contents not being viewed by users. When this happens, it may be useful to split the page in two or more pages or page sections, as suggested by the pattern "Clean Product Details" [6].
Mechanics: Let us suppose we want to split the page in two. The steps to follow are:

(i) Add a new page defining a new empty ADV for it.
(ii) Use '*Move Widget*' to move the selected interface widgets (both information and interaction widgets) from the original page to the new one;

(iii) Use '*Add Interface Anchor*' in each page to add anchors for links between the pages, to enable the user to access the content and the operations available in the original page.

If we want to split a page in different sections (thus maintaining a single page), the steps to follow are:

(i) Identify the boundaries of the different sections in which you want to split the page, and use '*Add Page Section*' for each one (i.e., defining inner ADVs for each section).
(ii) Use '*Move Widget*' to move the interface widgets from the ADV of the page to the ADVs of the corresponding section.
(iii) Use '*Add Interface Anchor*' in the ADV that describes the page, as many times as new sections have been defined, so to allow navigating from the top of the page to each section.
(iv) Use '*Add Interface Anchor*' at the end of each section to allow navigating back to the top of the page.

Example: A common situation in which this design improvement becomes useful is when dealing with pages displaying a text which is too long. In this case, we can split the page in a sequence of sub pages, distribute the text in these pages, and provide links from each new page to the following and backwards. It is worth to note that from the navigation model point of view the node underlying the original page is left unchanged, while it changes the way its content is presented, i.e., splitting it in more pages instead of a single one. As an example, let us consider the page "Novedades" (Novelties) in the Cuspide.com Web site. This page shows a list of links to recently published books, by category (Figure 15). Below this list, in the same page, another navigation index is showed. This index enables navigation to best sellers books, by category (Figure 16). Since the first index is very long, the second cannot be seen unless the user scrolls the page to half of its length. Additionally to this, given the page title (Novedades), it is also likely that the user does not expect to find the second list of links. In order to emphasize the second index and better support navigation to best sellers books, the '*Split Page*' WDI can be applied and the original page split in two, one showing the navigation index to "Novelties" and the second that to "Best Sellers". Figure 17 shows the ADVs representing the original page and the two pages resulting from splitting it. Navigation between the two new pages is enabled by the links represented with the edges 1 and 2 in the figure.
Related WDIs: Splitting a page into two or more pages may be the consequence of applying the '*Split Node Class*'(5.2.1) WDI in the navigation model. Moreover, it uses other WDIs in the presentation model, such as '*Add Interface Anchor*' and '*Add Page Section*'.

6.2.2 Replace Widget

Motivation: Inspection of usage of the site may show that the type of widget used to display some information item or to activate an operation, should be changed by a more appropriate one, to improve operability, usability or accessibility.
Mechanics: The precondition for this WDI is that, for the case of widgets that activate an operation, the new type of widget does not change the underlying functionality of the operation. For example, a single-selection list should not be changed by a multiple-selection list (unless changes in the application model requires it). If the precondition holds, identify

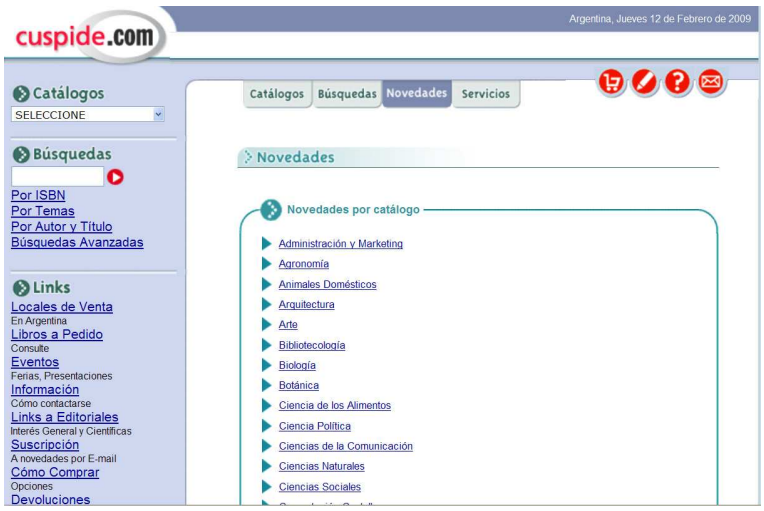

Fig. 15. The top portion of the page "Novedades" (Novelties) at Cuspide.com

the ADV for the page of the presentation model that contains the unsuitable widget, and replace the current widget by a more appropriate one.

Example: Check-boxes are best suited to enable users select one or more items from a list to perform an operation on them. A typical example is that of an email reader that allows selecting individual emails by means of check-boxes in order to apply, afterwards, operations like "delete" or "mark as read". Thus, users do not expect check-boxes to dispatch an operation when they are clicked but just to show a check-mark in the box. In the case of the Cuspide.com shopping cart [www.cuspide.com], which appears in Figure 18, checking any box under the title "Borrar" (Delete) automatically deletes the item from the cart, which is an unexpected behavior in terms of user experience. In this case, a more suitable widget would be a button with label "Borrar" or the usual trash can icon. Figure 19 shows on the left part the ADV for the Cuspide's webpage of Figure 18, and on the right part, the new ADV after '*Replace Widget*'.

Related WDIs: The WDI '*Turn Attribute into Link*'(5.2.3) in the navigation model may require the replacement of the widget displaying the node's attribute. The presentation WDI '*Introduce Information on Demand*', '*Introduce Link Destination Announcement*' and '*Introduce Scrolling*' all use '*Replace Widget*' in one or more of their steps.

6.2.3 Add Processing Page

Motivation: This WDI allows introducing the pattern "Processing Page" [10]. The purpose of a processing page is to provide feedback to users about the progress of their transaction, usually with a progress bar or a sand glass showing the progress of the execution of an operation triggered by the user.

Mechanics: The WDI is performed in three basic steps:

(i) Create a new page adding an empty ADV;
(ii) Add widgets into the new ADV for a progress bar and probably some text;

Fig. 16. The index "Best Sellers" in the bottom portion of the page "Novedades" at Cuspide.com

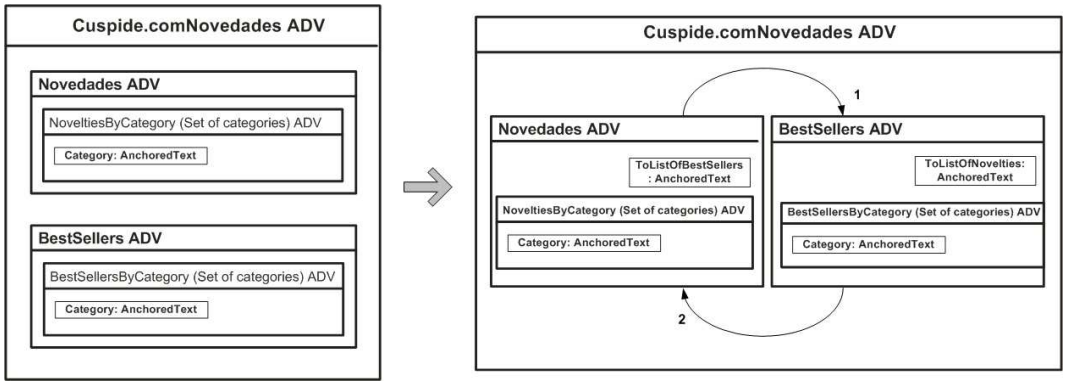

Fig. 17. ADVs representing the page "Novedades" from the Cuspide.com Web site, prior and after applying the '*Split Page*' WDI

(iii) Change the interface effect associated with the widget that triggers the operation, so that it also navigates to the processing page. OOHDM uses ADV-charts, a variant of state-charts, to specify the dynamic aspects of ADVs [5], although we do not describe them for space reasons.

It is worth noting that this WDI could also be applied by replacing inner ADVs in the same results page. This format is more common nowadays with RIA applications, and it is easily defined with aggregation of ADVs and ADV-charts.

Example: This solution is well visible, for example, in practically all airline operator websites or traveling brokers. In Figure 20, we show a processing page from Expedia.com [www.expedia.com].

In the case of our running example, during the check-out process in websites like Amazon, contacting the credit card service takes some time and a progress bar may provide feedback of the transaction stages like "communicating with bank", "checking card", "getting authorization", "ending", etc.

Fig. 18. Screenshot of the Cuspide's shopping cart

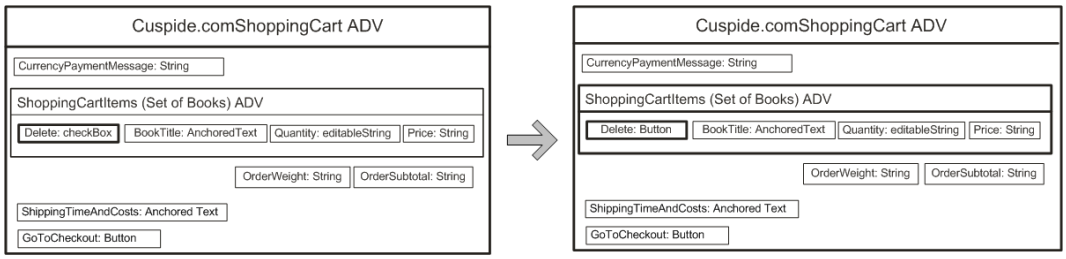

Fig. 19. ADVs representing the application of the Replace Widget WDI on the Cuspide's shopping cart page

Related WDIs: Changing the interface effect associated with a widget may require using '*Replace Widget*'(6.2.2) if the type of widget is no longer appropriate.

6.2.4 Allow Changing Category

Motivation: In a website that allows searching or browsing items by category, and in which there are hundreds of categories and subcategories, it is convenient to allow navigating to related subcategories that may be in a separate hierarchy. This improvement is intended to provide "Multiple Ways to Navigate" [6].

Mechanics: This WDI involves adding multiple widgets, like checkboxes or combo-boxes, to select the categories to change to, and labels for them. All widgets may be framed or enclosed in a different section by applying '*Add Page Section*'.

Example: Barnes&Noble allows changing categories while browsing books by selecting the corresponding checkboxes included in a section of the page entitled "Modify Your Selection". For example, if a reader follows the following chain of subcategories: 'Spanish' → 'Children' → 'Animals' → 'Birds' → 'Children's Nonfiction', arrives at the page shown in Figure 21. At this point, the reader may for example select 'Children', 'Animal' and 'Birds' in the section "Modify Your Selection", so as to browse non-Spanish books on that subject. However, Barnes&Noble does not appear to provide the same functionality for CDs, so the website could be improved in that respect.

Fig. 20. Example of Add Processing Page

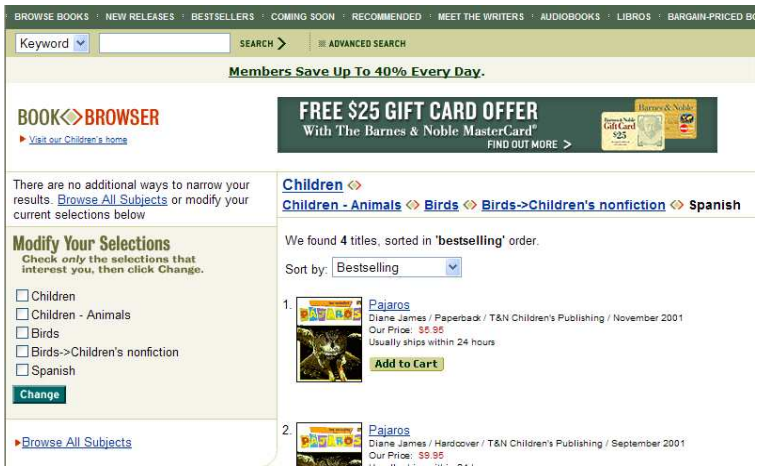

Fig. 21. Example of Allow Changing Category

Related WDIs: This WDI may be the consequence of applying '*Relate Categories*' at the navigation model. As described in the mechanics, the WDI '*Add Page Section*' may be applied to define a different section for the new widgets that allow to change the category.

7 Composite Web Design Improvements Involving Different Layers

The WDIs presented in the previous sections are self-contained and their application may produce an improvement in usability. However, a wise composition of these WDIs may yield a non trivial transformation with a higher impact on usability, which is nonetheless described as a step-by-step sequence of small WDIs.

To illustrate how WDIs can be composed into complex ones, we present two examples: '*Anticipate Target*', which may be realized as simpler WDIs at the navigation and/or presentation models, and '*Enrich Index*', a complex change using conjunctive and disjunctive compositions of atomic WDIs.

7.1 Anticipate Target

Motivation: Analysis of application's usage may show that users repeatedly backtrack after a forward link activation. The reason for this may be that the target of the link is not what the user expected. Too much false link activations (going forward and backward) will rapidly

lead to frustration and users leaving the site. To prevent this, the target of the link should be somehow "explained" better, e.g., by providing a preview of the target node or page, i.e., by anticipating the target. Other times anticipation of target is used to provide users with a summary of the information they would find visiting the linked page, so that if they find the summary satisfactory, they may avoid visiting the page resulting in faster navigation.

Mechanics: There are at least two different ways to anticipate the target of the link:

- *Show the Target.* Add a script to the anchor of the link so that when a mouse is rolled over it, it displays a small version of the target page. This is an interface transformation that can be specified with ADV-charts and it is usually implemented using some advanced scripting technology (such as Ajax). This solution is recommended for external links.

- *Add Target Information.* This solution involves changes in both the navigation and the presentation models. In the navigation class diagram, carefully select a few important attributes of the target node that best describe it (so as to reduce false link activations), and copy these attributes into the source node. In the ADV of the presentation model that is associated to the source node, add widgets to display the new attributes in the area of the link's anchor. If target information is added to all entries in an index, use '*Introduce Information on Demand*', to keep the index from growing excessively.

Example: An example of the results that could be obtained by applying this WDI is represented by Google Previews, which displays the target of each search result as an embedded image (instead of a pop-up). Figure 22 shows two other examples of target anticipation from the Amazon website. The screenshot on the left shows the pop-up window providing a summary of customer reviews on the product in the underlying page. The pop-up window shows up when rolling the mouse over a specific widget of the page (a small button with a downwards arrow). The same widget is used to provide a link towards the page showing the reviews summary and the details on each review. Similarly, the screenshot on the right shows the pop-up window that is used in Amazon to anticipate the page on a user/customer profile.

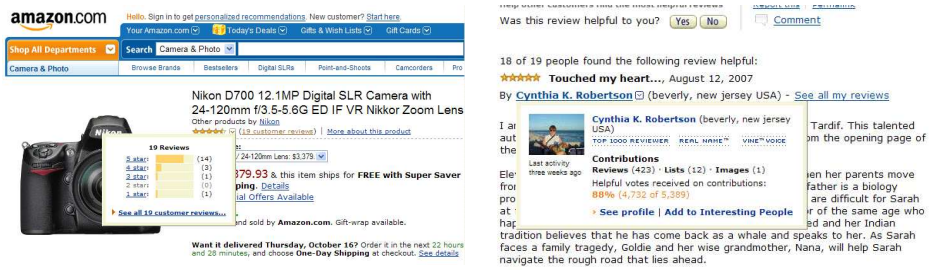

Fig. 22. Example of Anticipate Target

Related WDIs: Adding target information to each entry in an index works better when using '*Introduce Information on Demand*'.

7.2 Enrich Index

Motivation: Most Web applications are not only intended to publish content and data but also to enable the user to operate on them. While applying '*Anticipate Target*' gives some cues on the target of a link to decide whether or not to navigate to it, many times this cue is not sufficient to make decisions, for example when we have many different links as is typical in indexes. Moreover, once the desired target is selected (e.g., a product in an e-store), we might want to reduce the number of navigation steps necessary for users to achieve their goal (e.g. buy the product). One solution is to enrich the index in such a way that it contains more information and eventually operations on the target. However, since the space reserved for each index entry is usually scarce, special considerations should be taken at the presentation layer.

Mechanics: Suppose that we begin with a simple index like the one in Figure 1, which provides access to products. The following steps use smaller WDIs to enrich the index:

(i) Apply '*Add Target Information*' to '*Anticipate Target*' for each element of the index. The information is obtained from the target node of the corresponding index entry. The concrete steps, in terms of the OOHDM navigational diagram, are to change the index selectors (clickable anchors to navigate to the target) into complex structures (in fact component nodes) and writing a query on the target of the link to get the information. The final result would look as shown in Figure 2.

(ii) Apply '*Add Node Operation*'(5.2.4) to each element of the index. Add those operations belonging to the target element which should be made immediately available to the user. The final result would look as in the example of Figure 23.

(iii) After applying the previous steps, the index will have probably grown too large. There are different alternative WDIs that may be applied at the presentation layer to make a better use of the space:

- *Introduce Information on Demand.* This solution is directed by the pattern "Information on Demand" [8]. Using this pattern, the same section of the page is devoted to show information of all different index entries.
- *Introduce Link Destination Announcement.* This solution is directed by the pattern "Link Destination Announcement" [9]. In this case, a control is added to each index entry or a particular widget in the entry, so that when the mouse is rolled over the anchor, a pop-up appears with the information and operations added in steps i) and ii). The drawback of this solution is that pop-ups may be blocked or may be annoying for some users. An example of this solution appears in Figure 24.
- *Introduce Scrolling.* Use vertical or horizontal scrolling to make the index co-exist with other widgets in the page (see Figure 24 for an example of horizontal scrolling).
- *Split List.* Divide the entries of an index into several pages, allowing the user to navigate them sequentially. There are plenty of examples of this, like Google search results.

Example: E-commerce applications usually provide recommendations for their products as an effective way of advertising. This has become a feature that customers usually seek. On an emerging website, recommendations may be just a list of the products with a title and

a link to the product's page, i.e., a simple index as shown in Figure 1. We can see the intermediate results of applying each step of this design improvement in different versions of Amazon's "recommendations". Starting from Figure 1, we can apply Step (i), adding information about price, rating, a picture, etc., arriving at the page shown in Figure 2. Then we apply Step (ii) to add user operations "Add to cart" and "Add to wish list", arriving at the page shown in Figure 23. Here the index grew into a very long list. Finally, the space devoted to each entry index is reduced by applying '*Introduce Information on Demand*' and '*Introduce Scrolling*', as indicated in Step (iii). The result appears in Figure 24.

Fig. 23. Result of *Add Node Operation* to the index of Figure 2

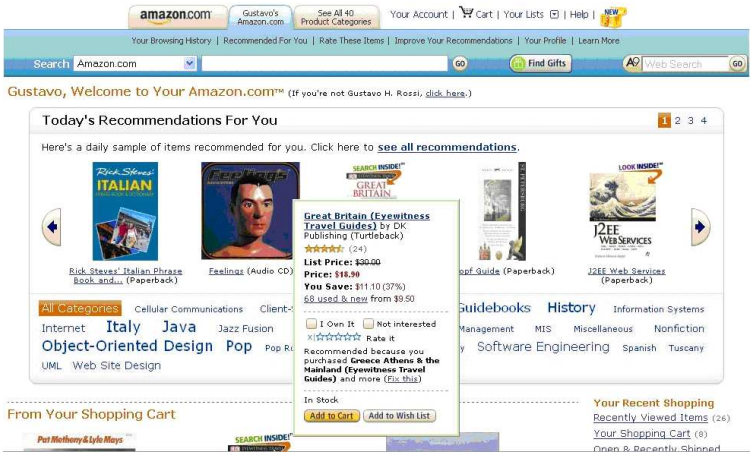

Fig. 24. Recommendation List after applying *Enrich Index*

Related WDIs: *Enrich Index* is composed of WDIs at both the navigation and the presentation levels. It shows two kind of composition: a conjunctive composition, where all improvements are applied in a particular order, and a disjunctive composition, where improvements are applied alternatively.

8 Conclusions and Future Work

In this paper we presented our approach for design improvements in Web applications. It is based on the view of modern Web engineering methods and it considers changes to the navigation and presentation design models.

As discussed in the introduction, Web applications have to be developed fast and, to make matters worse, must also evolve fast. Evolution might be driven by different factors such as change in requirements, but we also claim that evolution is strongly related to incremental and iterative development, now typical of agile methods, which has characterized Web applications development since early times. While agile methods have incorporated a number of auxiliary techniques, such as refactoring, to assure design and code quality, the problem of improving usability has been always considered as a separate issue regarding the development process. Moreover, while there are many texts on Web applications usability, e.g. considering desirable navigation and presentation properties, it might not be clear for developers how to find a good compromise between keeping the development pace fast and fulfilling these properties. A step by step process, based on small design changes as shown in Sections 5 and 6, allows dealing with the "permanent beta" state of this kind of software, introducing a kind of "agile" style of adding new features without complete re-engineering. Approaches for the automatic analysis and transformation of Web applications, such as that recently presented in [39] and focused on navigation structure, are also useful to keep the usability of a Web application high as it evolves during its life-cycle.

This paper shows, as a primary contribution, that it is possible to apply "small" design improvements to the navigation and presentation models (which are critical to usability) without changing the basic application's requirements. This implies focusing on simple situations and incrementally improving the application's quality. In this sense, our proposal impacts on some quality-in-use attributes in an equivalent way as refactoring impacts on code and design quality. We have demonstrated that our proposal allows a fine-grained characterization of the different kinds of design improvements, together with their impact in the various design models involved in the development life-cycle. We also showed that it is possible to identify the "bad smells" in usability to systematically apply one or more of our catalogue of WDIs. We have also shown some of those bad smells for navigational WDIs; by applying our approach combining WDIs with quality evaluation [29], the task of bad smell identification and deodorization can be related to the improvement of specific quality attributes.

We have exemplified how WDIs can help Web applications evolve by showing some simple, "atomic" improvements (such as '*Turn attribute into Link*' or '*Replace Widget*'), which introduce some basic concepts related with navigability or presentation design; additionally we have presented other WDIs, which require more reflection and introduce more elaborated navigation and interface structures. These improvements allow introducing well-known Web patterns. We have also shown how simple design improvements can be combined to achieve a more complex transformation, as one WDI unleashes others in the same or other design models.

Though some of the WDIs we propose may be suggested by common sense and not require a formal description, precisely describing them and the steps to be carried out for their application enable: 1) making sure the changes do not bring the application to an inconsistent state, 2) highlighting other WDIs implied by a given one, 3) suggesting practical situations

when each of the WDIs may be useful. Though we have described our WDIs using the syntax and semantics of the OOHDM design framework, the approach can be easily used in any of the well known Web design approaches, as the idea of systematically applying WDIs to a Web design is orthogonal to the way this design is expressed. Moreover, describing WDIs at the model level is a key issue for model-driven approaches, as it raises the abstraction level in which quality-in-use properties are discussed.

We are currently working to complete our catalogue of WDIs to cover other aspects of Web application design, such as business process or transaction design, and analyze more extensively their interaction and composition.

As technology evolves and new development possibilities emerge, there is more room to find WDIs. As an example, we are also researching on a specific sub-catalogue devoted to Rich Internet Applications. In these applications, it is possible to present the user with richer interaction, interface and also navigation possibilities; some of these improvements have been already reported [40].

Once these issues are addressed, a further research issue is to map WDIs to the implementation level, particularly, using a Web application framework (such as Struts, Shale, Tapestry, etc.). We are also studying how to extend the tools provided with Model-Driven Web engineering methods to support our catalogue of WDIs.

References

1. Beck, K. and Andres, C. (2005) *Extreme Programming Explained. Embrace Change*. Addison-Wesley.
2. Lehman, N. M. (1996) Laws of software evolution revisited. *Proceedings of EWSPT'96*, Nancy.
3. Koch, N. and Kraus, A. (2002) The expressive power of UML-based web engineering. *Proc. 2nd Int. Workshop on Web Oriented Software Technology (IWWOST02)*, Málaga, Spain.
4. UWA Consortium (2002) Ubiquitous web applications. *Proceedings of the eBusiness and eWork Conference e2002*, Prague, Czech Republic.
5. Schwabe, D. and Rossi, G. (1998) An object oriented approach to web-based application design. *Theory and Practice of Object Systems*, **4**, wiley and Sons.
6. Duyne, D. V., Landay, J., and Hong, J. (2003) *The Design of Sites*. Addison-Wesley.
7. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995) *Design Patterns. Elements of reusable object-oriented software*. Addison Wesley.
8. Rossi, G., Schwabe, D., and Garrido, A. (1997) Design reuse in hypermedia applications development. *Proceedings of Hypertext'97*, Southampton, UK.
9. Nanard, M., Nanard, J., and Kahn, P. (1998) Pushing reuse in hypermedia design: Golden rules, design patterns and constructive templates. *Proc. of Hypertext'98*, Pittsburgh, USA.
10. Web Design Patterns. http://www.welie.com/ patterns/.
11. Web Patterns. http://webpatterns.org.
12. Johnson, R. and Opdyke, W. (1993) Refactoring and aggregation. *Object Technologies for Advanced Software, First JSSST International Symposium*, vol. 742, pp. 264–278, Springer-Verlag.
13. Opdyke, W. (1992) *Refactoring Object-Oriented Frameworks*. Ph.D. thesis, University of Illinois at Urbana-Champaign.
14. Fowler, M. (1999) *Refactoring. Improving the Design of Existing Code*. Addison-Wesley.
15. Overbey, J., Xanthos, S., Johnson, R., and Foote, B. (2005) Refactorings for Fortran and High-Performance Computing. *2nd. Int. Workshop on Software Engineering for High Performance Computing System Applications*, St. Louis, MO.
16. Garrido, A. (2005) *Program Refactoring in the Presence of Preprocessor Directives*. Ph.D. thesis, University of Illinois at Urbana-Champaign. Tech.Rep.No. UIUCDCS-R-2005-2617.

17. Leitão, A. M. (2002) A formal pattern language for refactoring of Lisp programs. *CSMR'02: Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*, Washington, DC, USA, p. 186, IEEE Computer Society.
18. Ricca, F. and Tonella, P. (2005) Program transformations for web application restructuring. *Web Engineering: Principles and Techniques. Woojong Suh (eds.)*, chap. XI, pp. 242–260, Idea Group Publishing.
19. Ricca, F., Tonella, P., and Baxter, I. D. (2001) Restructuring web applications via transformation rules. *SCAM 2001*, pp. 150–160.
20. Harold, E. R. (2008) *Refactoring HTML: Improving the Design of Existing Web Applications*. Addison Wesley.
21. Boger, M., Sturm, T., and Fragemann, P. (2003) Refactoring browser for UML. *Objects, Components, Architectures, Services, and Applications for a NetworkedWorld: International Conference NetObjectDays, NODe 2002, Erfurt, Germany*, vol. LNCS 2591/2003, pp. 366–377, Springer Berlin.
22. Zhang, J., Lin, Y., and Gray, J. (2005) Generic and domain-specific model refactoring using a model transformation engine. *Model-driven Software Development*, chap. 9, Springer.
23. Gorp, P. V., Stenten, H., Mens, T., and Demeyer, S. (2003) Towards automating source-consistent UML refactorings. *Proceedings of the 6th Int. Conference on UML*.
24. Straeten, R. V. D., Simmonds, J., and Mens, T. (2003) Detecting Inconsistencies between UML Models Using Description Logic. *Description Logics*.
25. Mens, T. and Tourwé, T. (2004) A Survey of Software Refactoring. *IEEE Transactions on Software Engineering*, **30**.
26. Cabot, J. and Gomez, C. (2008) A catalogue of refactorings for navigation models. *Proc. of the 8th. Int. Conference on Web Engineering: ICWE'08*, Yorktown Heights, New York.
27. Ceri, S., Fraternali, P., and Bongio, A. (2000) Web modeling language (WebML): a modeling language for designing web sites. *Proceedings of the 9th World Wide Web Conference WWW9*, Amsterdam, NL.
28. Garrido, A., Rossi, G., and Distante, D. (2007) Model refactoring in web applications. *Proceedings of the 9th IEEE Int. Symposium on Web Site Evolution (WSE2007)*, Paris, France, October.
29. Olsina, L., Garrido, A., Rossi, G., Distante, D., and Canfora, G. (2008) Web application evaluation and refactoring: A quality-oriented improvement approach. *Journal on Web Engineering*, **7**.
30. Scharl, A. (2000) *Evolutionary Web development*. Springer.
31. Kerievsky, J. (2005) *Refactoring to Patterns*. Addison-Wesley.
32. De Troyer, O. and Leune, C. (1998) WSDM: A user-centered design method for web sites. *Computer Networks and ISDN systems, Proc. of the 7th Int. WWW Conf.*, Elsevier.
33. Pastor, O., Abraho, S. M., and Fons, J. (2001) An object-oriented approach to automate web applications development. *Proceedings of EC-Web*, Munich, Germany.
34. Rossi, G. and Schwabe, D. (2006) Model-based web application development. *Web Engineering*, chap. 10, Springer.
35. Kim, W. (1994) *Advanced Database systems*. ACM Press.
36. UWA Consortium, Deliverable d7: Hypermedia and operation design: model and tool architecture. www.uwaproject.org.
37. Cowan, D. D. and Lucena, C. J. P. (1995) Abstract data views: An interface specification concept to enhance design for reuse. *IEEE Trans. Softw. Eng.*, **21**, 229–243.
38. Olsina, L. and Rossi, G. (2002) Measuring web application quality with WebQEM. *IEEE Multimedia*, **9**, 20–29.
39. Scanniello, G., Distante, D., and Risi, M. (2008) Using semantic clustering to enhance the navigation structure of web sites. *Proc. of the 10th. Int. Symposium on Web Site Evolution: WSE2008*, Beijing, China.
40. Rossi, G., Urbieta, M. M., Ginzburg, J., Distante, D., and Garrido, A. (2008) Refactoring to rich internet applications. a model-driven approach. *Proc. of the 8th. Int. Conference on Web Engineering: ICWE'08*, Yorktown Heights, New York.

Appendix A

The following is a list of the Web patterns cited in this article, in order of appearance. We only provide a quick reference that includes a digest of the problem statement and solution proposed.

Table A.1. Some Web patterns cited in the paper.

Web pattern name	Problem	Solution
Clean Product Details[6]	When shopping, customers want to see product details to help inform their buying decisions.	Keep important items above the fold and put secondary items below the fold.
Overview by detail[10]	Users need to inspect objects that are part of a set.	Show an overview of all objects and display object details of a selected object on a new page.
Above the Fold[6]	Customers often miss navigation elements and content if they have to scroll down to see them.	Make sure that the most important material is at the top of the Web page.
Shopping Cart[6]	Customers want to collect and purchase several items in one transaction	Give customers easy access to the shopping cart from every page of your site
Context-Sensitive Help[6]	Customers sometimes need highly specific help to complete a task	Help your customers by pacing context-sensitive text and links near where they are needed on a page
Embedded Links[6]	Links off to the side or at the end of the text may lack the context readers need to understand how they relate to content	Embed links within a text passage to allow more free-form exploration
Printable Pages[6]	Customers become frustrated if a Web page does not offer a "printer-friendly" version	Create a printer-friendly page template by removing frames, additional columns and navigation bars
Browsable Content[6]	Browsing content can be difficult if the information is not organized, or if there are no clear and consistent navigation cues	Organize your content in several ways, in categories that make sense to your customers and in the intuitive ways
Hierarchical Organization[6]	Organizing information in a hierarchy of categories can help customers find things. Building an effective hierarchy is not easy	Build a hierarchy of categories with input from customers or experts in the area
Task-Based Organization[6]	Completing tasks is not fast and easy unless related tasks are linked together	Build relationships between tasks and link them so that the completion of one task is followed by the start of the next
Chronological Organization[6]	Chronologically organizing content helps visitors understand the order of content in time	Display chronological lists with 50 items maximum
Popularity-based Organization[6]	Some customers want to see which content or products are the most popular	Build lists of popular content from customer usage, ratings or acquired outside lists
Processing Page[10]	Users need feedback that their action is being performed but may take a while to complete	Provide a feedback page with animation
Multiple Ways to Navigate[6]	Customers navigate Web sites in many ways.	Put search and browse navigation tools at the top of the page.

Appendix B

This appendix provides a short description of the purpose of each WDI in the catalogs. Table B.1 lists all navigation design improvements and Table B.2 lists all presentation design improvements.

Table B.1. Brief description of navigation design improvements

Navigation WDI	**Purpose**
Split Node Class	Break a node class that has become too cluttered with information, anchors for links or functionality.
Merge Node Classes	Join two nodes that present scattered information.
Remove Unreachable Node	Clean the navigation model from nodes which are no longer reachable and thus are useless.
Move Node Attribute	Move an attribute from one node to another. Used in combination with '*Split Node Class*' in order to lighten a node which is found too cluttered with information.
Turn Attribute into Link	Provide navigation from a piece of displayed information to intermediate results of a transaction or context-sensitive data.
Add Node Operation	Allow adding an operation to a node to fulfill a new requirement such as speed-up a transaction, provide printing services or allow operations in index entries.
Move Node Operation	Move an operation from one node to another. Used in combination with '*Split Node Class*'.
Add Link	Provide new navigation possibilities.
Remove Redundant Link	Allow cleaning the navigation model from links that depart from the same source and go to the same target that another one that is selected most often.
Add Category	Organize related nodes in sets or categories that are meaningful to users, thus improving the website structure.
Split Category	Upon a large number of nodes in a category, split it in subcategories to improve the organization of the website content.
Merge Categories	Allow merging categories that have been found containing few items or that can be considered clones. This may help shortening the navigation path to the base content (e.g., products in an e-store) of the application.
Relate Categories	Allow navigation between categories.
Add Index	Allow defining a new node which serves as index to navigate towards a set of related nodes. It is used in combination with '*Add Category*' and '*Split Category*' to enable navigation to nodes in a context.
Change Navigation Topology	Allow modifying the navigation path between a set of nodes, while preserving their reachability. An example is that of transforming an access to a set of nodes by index, in which each node of the set is reachable starting from an index node, in an access by guided tour, in which there is a predefined order in which the nodes in the set are suggested to be visited.

Table B.2. Brief description of presentation design improvements

Presentation WDI	**Purpose**
Split Page	Divide a page in two or more pages or page sections.
Merge Pages	Merge two or more pages into a single page.
Add Page Section	Allow adding a section (portion of a page which can be represented by an inner ADV in the OOHDM presentation model) to an existing page, for example, to show the information of an additional node introduced by '*Add Node Class*' in the navigation model.
Merge Page Sections	Allow merging two or more page sections into a single one.
Move Widget	Allow moving a presentation/interaction widget from one page to another, e.g., when '*Split Page*' is applied and some widgets from the original page have to be moved into the new resulting page.
Replace Widget	Change an inappropriate widget to improve operability, usability or accessibility.
Add Interface Anchor	Needed, for example, when a new link is added to a given node, in order for the link to be showed.
Add Processing Page	Provide feedback to users about the progress of their transaction.
Provide Breadcrumbs	Help users keep track of their navigation path up to the current page and allow them to quickly navigate back to one of the previous navigation steps.
Allow Changing Category	Add widgets to allow navigating to related subcategories of items in a separate hierarchy of a hierarchical organization of contents.
Introduce Information on Demand	Replace an index with large entries by an index with smaller entries and an area to the side that is used to display details of the selected index entry
Introduce Link Destination Announcement	Enrich the anchor of a link with additional widgets to present information of the target page.
Introduce Scrolling	Allow to fix the width/height of a presentation widget (such as a text area) while showing a large amount of text/other widgets.
Split List	Similarly to '*Split Page*', it enables dividing a list of items that became too long, into more lists.