Model Refactoring in Web Applications

Alejandra Garrido[1], Gustavo Rossi[1] and Damiano Distante[2]
[1]*LIFIA, Universidad Nacional de La Plata, Argentina*
[2]*Research Centre on Software Technology (RCOST), University of Sannio, Italy*
{garrido, gustavo}@lifia.info.unlp.edu.ar, distante@unisannio.it

Abstract

Refactoring has been growing in importance with recent software engineering approaches, particularly agile methodologies, which promote continuous improvement of an application's code and design. Web applications are especially suitable for refactoring because of their rapid development and continuous evolution. Refactoring is about applying transformations that preserve program behavior. Code refactorings apply transformations to the source code while model refactorings apply to design models, both with the purpose of increasing internal qualities like maintainability and extensibility. In this paper we propose Web model refactorings as transformations that apply to the design models of a Web application. Particularly, we define refactorings on the navigation and presentation models, and present examples. Since changing these models impacts on what the user perceives, the intent of Web model refactorings is to improve external qualities like usability. They may also help to introduce Web patterns in a Web application.

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 as a consequence of users' feedback; new technological possibilities giving the chance to change the application's look and feel or interaction styles, etc.

Many times, however, evolution is only driven by the developers' reflection on the application structure, behavior and/or code; in these cases, the application is modified not to add new functionality but to improve its maintainability and extensibility for future and eventual additions. It is at this point where refactoring shows up. Refactoring was introduced some years ago in the context of object-oriented applications [11]. A refactoring was defined as a syntactic transformation of source code that improves its internal structure while preserving external behavior, i.e., the mapping of input to output values. A refactoring is performed in small steps, thus reducing the risks of breaking the system. Nevertheless, refactorings are usually composable [16], yielding larger transformations that improve readability, reusability and maintainability of a system.

With the upcoming of eXtreme Programming and Agile Methodologies, refactoring became quite popular as a disciplined process of continuous improvement of design and code [1, 5]. Well-known refactoring techniques have been catalogued as step-by-step recipes to help developers with a manual process, and many refactoring tools have been developed to automate these transformations [5]. The concept of refactoring has been applied to non-object-oriented languages [7] and to the level of design models (as UML class diagrams) [21, 24].

In the field of Web applications, we are interested in model refactorings that apply at the navigation and presentation models. In essence, navigation model refactorings modify the navigation topology while preserving the reachability of every node, and presentation model refactorings modify the look-and-feel of pages but preserve the operations available at each page. In this way, and similarly to traditional refactorings, Web model refactorings improve the internal structure of Web models while preserving the behavior as defined by these models. What differs from traditional refactoring is that modifying the navigation and presentation of a Web application impacts directly on what the user perceives. As a consequence, in our research we have analyzed those kinds of refactorings that, even preserving the application's functionality, may improve external qualities such as usability. These refactorings may also derive in the application of a Web pattern like those described in [20].

As an example that we will elaborate later in the paper, we show in Fig. 1 and Fig. 2, two different indexes of the Amazon bookstore. The first index points to a set of CDs by just exhibiting CD titles and interpreters. Meanwhile, the index in Fig. 2 gives much more

[1] Partially funded by project PICT 13623. Also CONICET.

information on each CD. Notice that the basic intended functionality, i.e. collecting a set of CDs and enabling navigation to pages that give details on each of them, did not change. However, the second index can be considered as an extension of the first one, obtained by applying some transformations to it (adding more information). This is the kind of atomic refactorings in which we are interested.

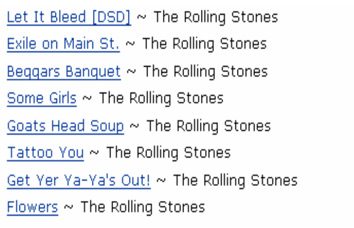
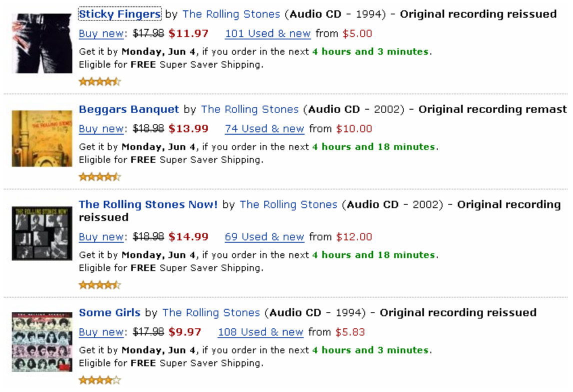

Let It Bleed [DSD] ~ The Rolling Stones
Exile on Main St. ~ The Rolling Stones
Beggars Banquet ~ The Rolling Stones
Some Girls ~ The Rolling Stones
Goats Head Soup ~ The Rolling Stones
Tattoo You ~ The Rolling Stones
Get Yer Ya-Ya's Out! ~ The Rolling Stones
Flowers ~ The Rolling Stones

Figure 1. A simple index


Sticky Fingers by The Rolling Stones (**Audio CD** - 1994) - **Original recording reissued**
Buy new: ~~$17.98~~ **$11.97** 101 Used & new from $5.00
Get it by **Monday, Jun 4**, if you order in the next **4 hours and 3 minutes**.
Eligible for **FREE** Super Saver Shipping.

Beggars Banquet by The Rolling Stones (**Audio CD** - 2002) - **Original recording remast**
Buy new: ~~$18.98~~ **$13.99** 74 Used & new from $10.00
Get it by **Monday, Jun 4**, if you order in the next **4 hours and 18 minutes**.
Eligible for **FREE** Super Saver Shipping.

The Rolling Stones Now! by The Rolling Stones (**Audio CD** - 2002) - **Original recording reissued**
Buy new: ~~$18.98~~ **$14.99** 69 Used & new from $12.00
Get it by **Monday, Jun 4**, if you order in the next **4 hours and 18 minutes**.
Eligible for **FREE** Super Saver Shipping.

Some Girls by The Rolling Stones (**Audio CD** - 1994) - **Original recording reissued**
Buy new: ~~$17.98~~ **$9.97** 108 Used & new from $5.83
Get it by **Monday, Jun 4**, if you order in the next **4 hours and 3 minutes**.
Eligible for **FREE** Super Saver Shipping.

Figure 2. An enriched version of the index in Figure 1

In this paper we characterize model refactoring in Web applications and present a set of concrete refactorings to illustrate our ideas. Particularly, our research is aimed at:

- Defining the subject of transformations of Web model refactoring and the semantics that they preserve.
- Analyzing the impact of Web model refactorings on the application's usability, particularly when they introduce Web patterns.
- Analyzing dependencies between navigation refactorings and presentation refactorings.

The remainder of the paper is structured as follows. In Section 2 we review some background concepts on refactoring and Web application development. In Section 3 we define and characterize Web model refactoring. In Section 4 we show examples of refactorings, analyzing their motivation and mechanics; we also indicate how atomic refactorings can be composed to create a refactoring with a larger impact. In Section 5 we discuss some related work and finally in Section 6, we provide some concluding remarks and discuss further research.

2. Background

2.1 An introduction to refactoring

Refactoring was defined in [11] as the technique that applies syntactic transformations to the source code of an application without changing its semantics, i.e., on a given input, the application produces the same output before and after refactoring. Refactoring differs from restructuring in that transformations are usually small and interactive. An example of refactoring is the extraction of a method or a component from a class.

Refactoring has also spread to the level of design models, giving rise to the concept of model refactoring [21, 24]. Typical model refactorings are applied on UML class diagrams and include the transformations of a class hierarchy, like pushing up/down methods and instance variables, and creating an abstract superclass by factoring out common features.

2.2 Refactoring to patterns

Design patterns came out in the early nineties as elegant solutions that experts would apply to solve a general problem [6]. Since then, we have seen plenty of catalogues of design patterns, code patterns, interface patterns, and even hypermedia and Web patterns.

One of the drawbacks of applying design patterns is the risk of over-engineer an application by applying patterns even when there might be a much simpler solution [8]. Moreover, it is usually difficult to understand when and how to apply a pattern. With the outcome of agile methodologies, developers move away from over-engineering and towards simpler designs [1]. However, under-engineering is as much or even more dangerous than over-engineering.

Refactoring comes to help keeping the balance between over and under-engineering [8]. The development process might start with a simple design and, if the need for more flexibility is later discovered, this design is refactored to incorporate the patterns that solve the specific problem.

2.3 The Web engineering life-cycle

In this paper we adhere to a model-based approach for Web applications development, which has been basically agreed by most mature development approaches like WebML [2], UWE [9], UWA [19], WSDM [3], OOWS [12], OOHDM [18], etc.

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*. A running implementation is then derived from these models either by applying to them a set of heuristics or by using a transformation tool.

The *application model* (also known as content model) describes the structure of the application's data, i.e., the contents it will provide to users, their associations and the possible operations on these data.

The *navigation model*, which is essential for Web software, specifies the units of consumption of the application contents (i.e., navigation nodes), the navigation paths through contents, (i.e., links, indexes, etc.) and the operations each node will enable.

Finally, the *presentation model* (also known as user-interface model) defines the mapping from nodes to pages, their look and feel, the interface objects needed to facilitate navigation or other user actions, and the interface transformations that occur as the result of the user interaction.

Each Web application development method uses its own armory of primitives to represent the above design models. We based our work on common design primitives which we describe in Section 3.1, using the OOHDM terminology.

3. Defining and characterizing model refactoring in Web applications

Refactorings can be applied to Web applications both at the implementation and the model level. Refactorings at the implementation level are similar in intent and structure to conventional code level refactorings [5], though they might deal not only with object-oriented code but also with code in typical Web languages such as HTML, XML, JavaScript, etc. This paper does not address Web code refactorings, but they are mentioned in the related work section.

At the model level, refactorings can be applied to any of the design models of a Web application, i.e., according to the generic development approach described in Section 2.3, to the application, navigation and presentation models. Since the application model of a Web application is, for most methods, similar to a UML class diagram, refactorings that may be applied to this model are basically the same described in [21, 24]. Therefore, we will focus on refactorings that can be applied to the navigation and presentation models, which we call *Web model refactorings*, or specifically, *navigation model refactorings* and *presentation model refactorings*, respectively.

3.1 Characterizing navigation and presentation model refactorings

Navigation and presentation model refactorings affect the way the application presents contents, enables navigation through contents, and provides interaction capabilities. In order to identify the possible navigation and presentation refactorings that may be applied to a Web application, it is important to define the subject of transformation and the behavior these refactorings should preserve.

The navigation model produced with the OOHDM method is the *navigational class diagram*, a UML class diagram where classes represent navigation nodes, associations represent navigation links, and indexes are a particular type of node defined to enable one-to-many navigation. Nodes are derived from classes in the application model. They are described with anchors for links and also content attributes and methods, the latter mapping application class attributes and operations. Indexes are a kind of composite node containing a set of entries. Each entry 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. Access structures like guided tours can also be defined.

Navigation model refactorings, having the navigational class diagram as the subject of change, may involve changes to any of the properties defined by this model, such as:

- The contents of a node (including index nodes);
- The outgoing links in a node;
- The navigation topology associated to a set of nodes (guided tour, index, etc.);
- The user operations enabled by a node.

The behavior of a Web application, as specified at the navigation level, is given both by the set of operations available at each node but also by the set of links that allow the user to navigate through the set of nodes. Thus a refactoring at this level should preserve:

- The set of possible operations and the semantics of each operation;
- The "navigability", which is defined as the set of nodes the user can navigate.

We are now able to define navigation model refactorings precisely: they are transformations applied on the navigational class diagram that preserve operational semantics and navigability. Preserving the navigability of the set of nodes means that existing nodes may not become unreachable though the set may be augmented (e.g. by splitting a node). Moreover, these refactorings should not introduce information, relationships or operations that are not in the application model. Examples of these refactorings are:

1. Add contents or operations to a node, provided they are available in the application model.
2. Add new links between existing nodes.
3. Remove a link between nodes if that does not make a node unreachable.
4. Add a node or remove an unreachable node.

The OOHDM presentation model describes the "abstract interface" of the application by specifying how the navigation objects and the application functionality will be perceived by the user. Moreover, it focuses on the various types of functionality that can be played by interface elements, either displaying the node's data (e.g. multimedia fields) or triggering its associated operations. The presentation model is composed of pages and widgets describing the look and feel of pages. Specifically, it includes the following elements that may be subject to change:

- The general layout of a page;
- The graphical widgets that compose a page, with their type and position;
- The nodes grouped and presented into a page;
- The interface transformations occurring as the result of user interaction.

The behavior at the presentation level is given by the operations that the user may trigger, including both operations of the underlying node or link activations. Therefore, legal presentation model refactorings may not remove available user operations, though they may:

1. Change the look and feel of the page by moving widgets around;
2. Add information or operations available in the underlying node;
3. Add or change the available interface effects.

Notice that most navigation refactorings will imply some change in the user interface and thus in the presentation model. For example, if we add some content to a node, we should add the corresponding interface object to make this information perceivable; if we add a link, its corresponding origin (anchor) should be made visible, etc. Meanwhile, presentation model refactorings should not change the navigational structure, i.e., if we add some interface object or interface effect (e.g. scrolling), the navigation topology should be the same after the addition.

3.2 Refactoring to Web patterns

The concept of Web patterns emerged as the application to the Web of the hypermedia patterns concepts developed in the late 90s [17, 10]. Web patterns are similar to design patterns because they address a recurrent (Web) design problem with a generic solution that can be instantiated according to the specific application being solved. The reader can find catalogues of Web patterns at [20, 22, 23].

With the same spirit of "Refactoring to Patterns" [8], we propose model refactorings on web applications to introduce Web patterns in their architecture. Kerievsky also calls them "*pattern-directed refactorings*". Similarly to them, Web-pattern-directed refactorings discuss the problems that each Web pattern helps solve and the pros and cons of applying it.

4. Towards a catalogue of navigation and presentation model refactorings

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, we have identified a group of concrete navigation and presentation model refactorings that we present in this section. Similarly to the well-known refactorings described in [5, 8], each one is described using a common template comprising *Name*, *Motivation*, *Mechanics, Examples* and *Impact*. The last element of the template is added here to describe the model levels affected directly or indirectly by the refactoring. Web patterns involved in the refactorings are referenced in italics, while references to other refactorings are distinguished with a capitalized font. We first present a list of atomic (i.e., basic) refactorings and then an example of a composite refactoring that uses atomic refactorings as individual steps of its mechanics.

4.1 Atomic refactorings

4.1.1. Add Information

Motivation: Application usability studies may show the need to display more information than what is currently on a page. The kind of information may come from different sources or have different purposes: it may be data extracted from the application model; it may be information added with the purpose of attracting customers or advertising; or it may be data introduced to help during navigation (obtained from the navigation model itself).

This refactoring may be used to introduce patterns like *Clean Product Details* [20] to add details about products in an e-commerce website. With the purpose of attracting customers, we may introduce *Personalized Recommendations* [20] or rating information. Data that helps during navigation may be added to introduce *Active Reference* [17], i.e., to provide a reference about the current status of navigation.

Mechanics: The mechanics may vary according to the different sub-intents above. In the most general case: add an attribute to a node class in the navigation model where the information is to be added. If the information is extracted from the application model, attach to the attribute the statement describing the mapping to the application model [18].

If the refactoring is used to introduce *Active Reference* [17], add an index to the node class such that the current page is highlighted in the index.

Example: This refactoring is applied to transform the page that appears in Fig. 1 into the one that appears in Fig. 2. In this case the information is added to each entry of the index; the added information includes the CD picture, price, rating, sale information, links to list of sellers, year of edition, etc.

Impact: This is a navigation model refactoring since it involves adding an attribute to a node class in the navigation model. However, it will also produce a refactoring on the presentation model (ADD WIDGET), to display the new information.

4.1.2. ADD 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* [20], etc. The operation may be also added to each entry on an index, so that the user does not need to navigate to the node describing the entry to operate on it.

Mechanics: Add an operation to the appropriate node class in the navigation model. Note that the operation should be already available in the application model.

Example: In the Amazon bookstore, the check-out process has evolved 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"*. Selecting that button allows the user to login and retrieve all previous information so she does not have to re-enter the information for every purchase.

Impact: Adding an operation to the navigational model requires the interface to be augmented with an extra widget to dispatch it. Note that there is an associated refactoring at the presentation model layer (ADD BUTTON or ADD INTERFACE OPERATION). The latter would be the case of including operations that turn on accessibility features or customize the interface to the user.

4.1.3. ANTICIPATE TARGET

Motivation: Analysis of application's usage may show that users repeatedly backtrack after a forward link activation. The reason for this is usually 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 customers 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 or anticipating the target.

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 refactoring and it is usually implemented using some advanced scripting language (such as Ajax). This solution is recommended for external links.
- Add Target Information. This solution is a variation on Add Information; in this case the information that is added to a node is derived from the target of a link, in the anchor's area. The information displayed about the target should be carefully chosen to reduce false link activations. This refactoring applies at the navigation model and will therefore unleash refactorings at the presentation layer to reflect it. Here we can use different alternatives, from adding widgets around the anchor to augment it with target information, to introducing patterns like *Information on Demand* [17], which are usually found in indexes as discussed in Section 4.2.

Examples: Google Previews displays the target of each search result as an embedded image (instead of a pop-up). Figure 3 shows an example from Dr. Dobb's Portal (http://www.ddj.com) that adds target information for different links ("Webinars", "Architecture", etc.) in the same area, using *Information on Demand* [17] at the presentation level.

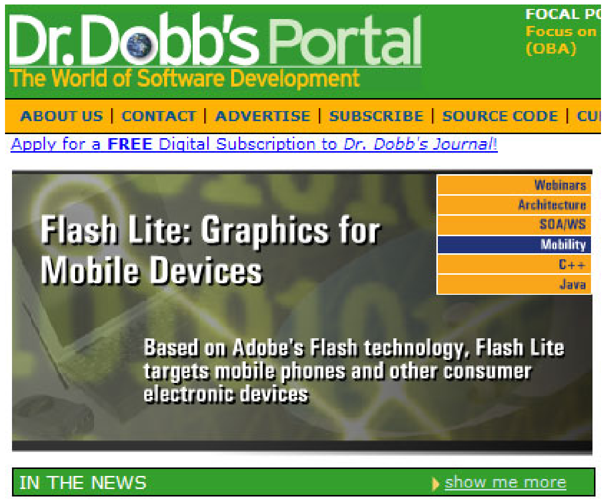

Dr.Dobb's Portal
The World of Software Development
FOCAL P
Focus on
(OBA)
ABOUT US | CONTACT | ADVERTISE | SUBSCRIBE | SOURCE CODE | CU
Apply for a FREE Digital Subscription to Dr. Dobb's Journal!
Webinars
Architecture
SOA/WS
Mobility
C++
Java
Flash Lite: Graphics for Mobile Devices
Based on Adobe's Flash technology, Flash Lite targets mobile phones and other consumer electronic devices
IN THE NEWS
show me more

Figure 3. Showing different targets in the same area.

Impact: Notice that this refactoring has two alternative mechanics, one at the presentation level, and the second at the navigation level. Applying the second solution, i.e., Add Target Information, on the navigation model will trigger other refactorings at the presentation layer to reflect the transformation.

4.2 Composite refactorings

Each of the refactorings presented in 4.1 is self-contained and its application may yield an improvement in usability. However, a wise composition of atomic refactorings may imply a non trivial transformation with a higher impact on usability, which is nonetheless described as a step-by-step sequence. To illustrate, we present ENRICH INDEX, a composite refactoring that is accomplished in three steps, using the atomic refactorings presented in the previous section.

4.2.1. ENRICH INDEX

Motivation: In many Web applications we navigate not only to get information about objects but also to operate on them. While applying ANTICIPATE TARGET gives some cues on the target of a link to decide whether to navigate or not, 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 to achieve our 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 providing access to products. The following steps show how to use atomic refactorings to improve the index:

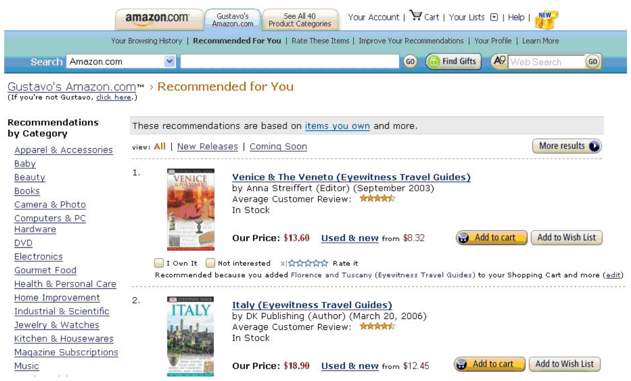
1) Apply Add Target Information to ANTICIPATE TARGET of each element of the index. The information is obtained from the target node of the corresponding index entry. The concrete transformation, in terms of the OOHDM navigational diagram, is to change the index selectors (clickable anchors to navigate 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 end result after mapping this step to the interface would look as shown in Figure 2.

2) Apply ADD OPERATION to each element of the index. Add those operations belonging to the target element which should be made immediately available to the user. The end result would look as in the example of Figure 4.

3) After applying the previous steps, the index will have probably grown too large. There are different alternative refactorings 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* [17]. Using this pattern, the same section of the page is devoted to show information of all different index entries, as shown in Figure 3.
- INTRODUCE LINK DESTINATION ANNOUNCEMENT. This solution is directed by the pattern *Link Destination Announcement* [10]. 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 1) and 2). The drawback of applying this solution is that pop-ups may be blocked or may be annoying for some users. An example of this solution appears in Figure 5.
- INTRODUCE SCROLLING. Use vertical or horizontal scrolling to make the index co-exist with other widgets in the page (see Figure 5).
- 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. It has become a feature that customers usually seek. On an emerging web site, 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 Fig. 1. We can see the intermediate results of applying each step of this refactoring in different versions of Amazon's "recommendations". Starting from Fig. 1, we can apply step 1, adding information about price, rating, a picture, etc., arriving at the page shown in Fig. 2. Then we apply step 2, to add operations "Add to cart" and "Add to wish list", arriving at the page shown in Fig. 4. 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 3. The result appears in Figure 5.


amazon.com
Gustavo's Amazon.com
See All 40 Product Categories
Your Account | Cart | Your Lists | Help
Your Browsing History | Recommended For You | Rate These Items | Improve Your Recommendations | Your Profile | Learn More
Search Amazon.com
GO
Find Gifts
Web Search
GO
More results
Add to cart
Add to Wish List
I Own It
Not interested
Rate it

Gustavo's Amazon.com™ › Recommended for You
(If you're not Gustavo, click here.)

Recommendations by Category
Apparel & Accessories
Baby
Beauty
Books
Camera & Photo
Computers & PC Hardware
DVD
Electronics
Gourmet Food
Health & Personal Care
Home Improvement
Industrial & Scientific
Jewelry & Watches
Kitchen & Housewares
Magazine Subscriptions
Music

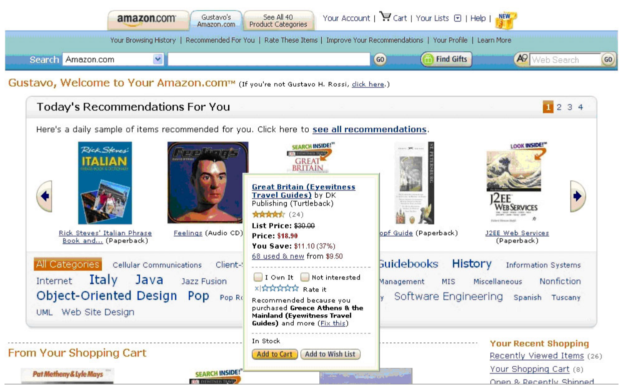
These recommendations are based on items you own and more.

view: All | New Releases | Coming Soon

1. Venice & The Veneto (Eyewitness Travel Guides)
by Anna Streiffert (Editor) (September 2003)
Average Customer Review:
In Stock
Our Price: $13.60 Used & new from $8.32
Recommended because you added Florence and Tuscany (Eyewitness Travel Guides) to your Shopping Cart and more (edit)

2. Italy (Eyewitness Travel Guides)
by DK Publishing (Author) (March 20, 2006)
Average Customer Review:
In Stock
Our Price: $18.90 Used & new from $12.45

Figure 4. Result of Add Operation refactoring on the index in Figure 1



Gustavo, Welcome to Your Amazon.com™ (If you're not Gustavo H. Rossi, click here.)

Here's a daily sample of items recommended for you. Click here to see all recommendations.

Great Britain (Eyewitness Travel Guides) by DK Publishing (Turtleback)
(24)
List Price: ~~$30.00~~
Price: $18.90
You Save: $11.10 (37%)
68 used & new from $9.50

Recommended because you purchased **Greece Athens & the Mainland (Eyewitness Travel Guides)** and more (Fix this)

In Stock

Figure 5. Recommendation List after Enrich Index

Impact: Enrich Index is composed of refactorings at both the navigation and the presentation levels. Moreover, it shows that we can compose refactorings in different ways: a conjunctive composition, where all refactorings are applied in a particular order, or a disjunctive composition, where we can apply refactorings alternatively.

5. Related work

Our research differs from existing work in the Refactoring field in two aspects: the *subject* and the *intent* of refactoring. The subjects of the refactorings we propose are the navigational and interface models of Web design methods. Meanwhile, most of the existing literature on refactoring is targeted at the code level. Our work also differs from the work on model refactoring [21, 24], even for those Web design methods entirely based on UML [9], since our target is not the application model but the navigation and presentation models.

Regarding the intent of refactoring, Web model refactorings are aimed at improving the users' experience with the Web application. The same intent (improving usability) is shared by the work on Web transaction reengineering [4], although their subject are business transactions and their transformations are not necessarily refactorings.

Ricca and Tonella have worked on code restructuring for Web applications [14]. They define different categories of restructuring, like Syntax update, Internal page improvement and Dynamic page construction. As explained before, refactoring differs from restructuring in that the latter implies larger transformations that are usually run in batch mode by applying certain rules. Instead, refactorings are smaller and applied interactively. However, the main difference with our work is that their transformations apply on the source code, in this case, HTML, PHP and/or Javascript. They have some of their restructuring rules implemented in the DMS reengineering tool [15].

Ping and Kontogiannis propose an algorithm to cluster links into several types and group web pages according to these link types [13]. Applying this technique should provide a roadmap for the identification of controller components of a controller-centric architecture. Although their target is the navigational structure of a Web application, they do not provide the mechanics to apply the transformation, but only a first step of recognizing where to apply it.

6. Conclusions and future work

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

Our proposal allows a fine-grained characterization of the different kinds of refactorings, together with their impact in the various design models involved in the development life-cycle. We have demonstrated how refactorings can help Web applications evolve by applying well-known Web patterns into their design, in order to improve quality in use properties, such as usability. Moreover, we have shown how refactorings can be combined to achieve a more complex transformation, as one refactoring unleashes others in the same or other design models.

We are currently working to augment our catalogue of refactorings to a full set of navigation and presentation model refactorings, showing their interaction and composition. Though not discussed in this paper, we are also researching on how to apply the same concepts in the process of evolution to Rich Internet Applications, in which more sophisticated interface transformations are available. A step by step process, based on small refactorings as shown in Section 4.2, allows progressing from the "permanent beta" state of this kind of software.

Once these issues are addressed, a further research issue is to map refactorings to the implementation level, upgrading Web design tools to support refactoring of navigational structures, and also mapping our model refactorings to code refactorings of applications built with a particular web application framework (like Struts, Shale, Tapestry, etc). A refactoring tool should also be very helpful in synchronizing changes at different layers, i.e., from the navigation model to the presentation model and to the implementation level.

7. References

[1] Kent Beck. *Test-Driven Development*. Addison-Wesley. 2002.

[2] Ceri, S., Fraternali, P., Bongio, A. Web Modeling Language (WebML): a Modeling Language for Designing Web Sites.

Proc WWW9 Conference, Amsterdam, NL, May 2000 (also in Computer Networks, 33 (2000), pp. 137-157).

[3] De Troyer O., Leune C. WSDM: A User-Centered Design Method for Web Sites. Computer Networks and ISDN systems, Proc. of the 7th Int. WWW Conf. Elsevier. 1998.

[4] Distante, D., Tilley, S., Huang, S. "Web Site Evolution via Transaction Reengineering". In Proc. of the 6th Int. Workshop on Web Site Evolution: WSE'04. Chicago, 2004.

[5] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 2000.

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns. Elements of reusable object-oriented software*, Addison Wesley 1995.

[7] Alejandra Garrido. Program Refactoring in the Presence of Preprocessor Directives. Ph.D.Thesis, Univ. of Illinois at Urbana-Champaign, 2005.

[8] Joshua Kerievsky. *Refactoring to Patterns*. Addison-Wesley, 2005.

[9] Koch, N., Kraus, A. The Expressive Power of UML-based Web Engineering. In *Proc. of 2nd Int. Workshop on Web Oriented Software Technology* (IWWOST´02) at ECOOP´02. 2002. Málaga, Spain. pp. 105-119.

[10] M. Nananr, J. Nanard, P.Kahn. Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates. In *Proc. of Hypertext'98*. Pittsburgh, USA. 1998.

[11] William Opdyke. Refactoring Object-Oriented Frameworks. Ph.D.Thesis, Univ. of Illinois at Urbana-Champaign, 1992.

[12] Pastor, O., Abrahão, S.M., Fons, J. An Object-Oriented Approach to Automate Web Applications Development in Proceedings of EC-Web 2001. Munich, Germany, 2001.

[13] Y.Ping and K.Kontogiannis. Refactoring Web sites to the Controller-Centric Architecture. In *Proc. of the European Conf. on Software Maintenance and Reengineering (CSMR'04)*. Tampere, Finland, 2004.

[14] F. Ricca and P. Tonella. Program Transformations for Web Application Restructuring. In *Web Engineering: Principles and Techniques*. Woojong Suh (eds.). chapter XI: pp. 242-260. 2005.

[15] Filippo Ricca, Paolo Tonella, Ira D. Baxter: Restructuring Web Applications via Transformation Rules. SCAM 2001: pp. 150-160, Firenze, Italy, 2001.

[16] Donald Roberts. Eliminating Analysis in Refactoring. Ph.D.Thesis, Univ. of Illinois at Urbana-Champaign, 1999.

[17] G.Rossi, D.Schwabe, A.Garrido. Design Reuse in Hypermedia Applications Development. In *Proceedings of Hypertext'97*. Southampton, UK, 1997.

[18] Schwabe, D., Rossi, G. (1998) An Object Oriented Approach to Web-Based Application Design. Theory and Practice of Object Systems 4(4), Wiley and Sons, 1998.

[19] UWA Consortium, Ubiquitous Web Applications. Proceedings of the eBusiness and eWork Conference e2002: 2002; Prague, Czech Republic.

[20] D. Van Duyne, J. Landay, J. Hong. *The Design of Sites*. Addison-Wesley 2003.

[21] P. Van Gorp, H. Stenten, T. Mens, S. Demeyer. Towards automating source-consistent UML Refactorings. In *Proceedings of the 6th Int. Conference on UML*, 2003.

[22] Web Design Patterns. http://www.welie.com/ patterns/

[23] Web Patterns. http://webpatterns.org.

[24] J. Zhang, Y. Lin, J. Gray. Generic and Domain-Specific Model Refactoring using a Model Transformation Engine. In *Model-driven Software Development*, Springer, Ch. 9, 2005.