Engineering Concern-Sensitive Navigation Structures. Concepts, Tools and Examples

Gustavo Rossi[1], Sergio Firmenich[1], Matias Urbieta[1], Silvia Gordillo[1] , Jocelyne Nanard[2], Marc Nanard[2], Cecilia Challiol[1] and Joao Araujo[3]

[1]Facultad de Informática, Universidad Nacional de La Plata and Conicet Argentina
[gustavo, sergio.firmenich, matias.urbieta, gordillo@lifia.info.unlp.edu.ar]
[2] LIRMM, CNRS/Univ. Montpellier, 161 rue Ada, F34392 Montpellier cedex 5, France
[jnanard, mnanard]@lirmm.fr
[3] Departamento de Informatica, Faculdade de Ciencia e Tecnologia, Universidade Nova de Lisboa, Portugal
ja@di.fct.unl.pt

Abstract. Improving navigability in Web applications is a serious challenge for developers as this quality feature is essential for applications success. In this paper we present the concept of concern-sensitive navigation, a useful conceptual tool to improve navigation by profiting from the nature of application's concerns. Concern sensitive navigation allows enriching Web pages with information, services or links related with the context in which pages are accessed. We show how our ideas are applied during the development process (e.g. by applying wise design strategies for separation of concerns) and can also be used by final users while adapting an application (e.g. by modding). Some examples of Web 2.0 sites are used to illustrate this last possibility. We also compare our research with other similar approaches such as the construction of adaptive Web applications.

Keywords. Separation of concerns, Concern-sensitive navigation, User experience, Web 2.0, Engineering

1 Introduction and Motivation

The rapid evolution of the Web has changed dramatically the way in which we interact with information and services sources. While we are still using popular and "old" Web applications, such as the traditional e-commerce sites, a new generation of Web software has arisen. Interactive encyclopedia like Wikipedia [44], social sites such as Facebook [13], video and photograph servers such as Youtube [45] and Flickr [15] not only changed our way to access the Web but also motivated the traditional sites to introduce new ways to interact and share information, as it is now possible with tagging facilities or forum support in sites like Amazon.com [2]. Users can build mashups, and can also customize their preferred sites either by pre-defined configuration patterns such as in my.yahoo.com, or by programming small scripts which can adapt the application's interface and presentation style in a process called modding [11]. In this way users face themselves to new opportunities and of course new problems.

This rapid evolution has been followed by design approaches such as OOHDM [38], WebML [7], UWE [27] or OOWS [16], which provide conceptual and concrete tools (e.g. model-driven engineering environments [18]) to produce applications quicker and in a correct way. However, there are still open issues regarding the quality of final applications; in Figure 1 we show the page of a product in Amazon.com. This is one application in which users navigate through thousands of products with different concerns (tasks or interests) in mind. In Figure 1.1, for example, the MP3 player page contains a discount banner because the page was accessed from the "Audible device ready" index. Meanwhile in Figure 1.2, the same product page contains information related to its compatible accessories, because the page was accessed from the "Ipod and accessories" index.

Figure 1.1: Ipod Touch view from Audible device ready index

Figure 1.2: Ipod Touch view from Ipod and accessories index

The contents of the MP3 player page is improved by taking into account the "dominant" concern in which it is being accessed; in this case the index in which we selected the product. In both cases, knowing the actual user's concern (what he is looking for) helps to enrich the information on the target page with new contents and links to simplify or clarify the user's task.

To abstract and generalize this simple idea, in [32] we introduced the concept of concern-sensitive navigation (CSN), a strategy to provide a more *flexible* navigational structure, to improve the user navigation experience. Our work aimed at improving the cognitive and rhetoric access to information, which means providing the user with the needed information in each concern, and such that it is organized and presented in a more opportunistic way [24]. By using modern engineering approaches, which favor a good separation of application concerns, we can provide CSN by profiting from the information collected during the design stage to offer richer and less flat navigational structures. CSN can be seen as a kind of Web software adaptation, which does not need the management of user profiles, therefore overcoming the usually discussed privacy and security issues.

In this paper, we elaborate the concept of CSN showing its feasibility in a broad range of Web applications. Specifically we address two different kinds of CSN adaptations: intra and inter-application. We analyze the process of client-based adaptation and present both a disciplined process and a tool to build CSN structures in existing applications. The main contributions of the paper are the following:

- We present a lightweight approach for adaptation by using the application concerns to improve navigability.
- We show that our ideas can be used either by developers of new applications in an intra-application context or by adapting existing applications in a client-based way.
- We present a set of tools to support the process of adding inter-application CSN to existing Web applications.

The structure of the paper is as follows. In Section 2 we introduce CSN and characterize its intent and scope. In Section 3 we discuss the engineering of CSN structures both during developing time and for existing applications. In Section 4 we discuss how to implement CSN in Web applications, and introduce the problem of adapting existing applications; in this context, in Section 5 we discuss several issues related to client-based adaptation; in particular we discuss several design issues to achieve modular adaptation code, and briefly describe a tool to support the process. Section 6 discusses some related work and in Section 7 we conclude and present some ideas for further research.

2 Concern-sensitive navigation

According to [42] we define a concern as a "matter of consideration in a software system". A concern may reflect functional aspects[1] of an application such as uploading Videos in YouTube, Products search or checking out in Amazon, or topic areas such as economy or history in an Encyclopedia. Concerns may be *generic*, when they appear in a broad number of applications (e.g. support for secure login), *domain specific* when they only apply to a set of applications (adding posts or comments in some Web 2.0 software), or even *application specific* when they only show up in a particular kind of software (e.g. routing features in Google Maps). Some kinds of concerns may be defined abstractly during application development (e.g. management of categories in an Encyclopedia) and instantiated by users while using the software (e.g. dealing with a concrete category). A *navigational concern* is an application concern that affects navigation, i.e. it manifests in the navigational structure of the application (i.e. in the exhibited contents, operations and links).

2.1 Concern Sensitive Navigation Functionality: Definition

We say that a Web application supports CSN when the contents, links and operations exhibited by the application pages are not fixed for different navigation paths, but instead can change when accessed in the context of different navigation concerns. To make this definition concrete and practical we assume that users navigate through *navigation objects* which are the realization of hypermedia nodes. Suppose a navigation object Nj which is an instance of a navigation object type N. While in conventional Web navigation, this object will exhibit the same contents and links regardless of how it was reached, in CSN its properties can be slightly adjusted according to the

[1] Non functional concerns such as usability or accessibility though critical in Web software are not relevant for our paper

current user concern as shown in Figure 2. While Nj comprises two attributes (Figure 2.1.), when accessed in a specific concern C it also exhibits an additional attribute and an anchor for a link (Figure 2.2). In Figure 3 we illustrate the idea with the example of Figure 1.

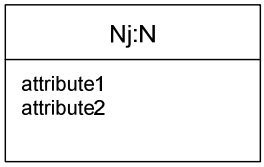

Figure 2.1: Conventional navigation to Nj

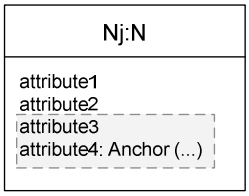

Figure 2.2: Nj accessed in C

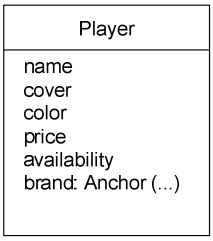

Figure 3.1: A basic product page

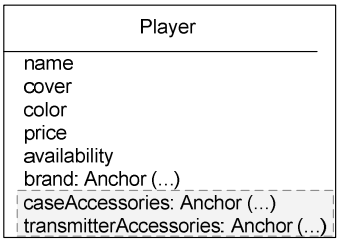

Figure 3.2: Product page enriched with concern information

While the basic definition of CSN does not impose limits to the kind of variations which navigation objects might suffer when accessed in different concerns, a disciplined use of these ideas (i.e. in the context of a solid design approach), have a stronger positive impact on application usability. As explained in the following subsection, the concept of CSN is strongly related to the concept of object roles, as applied to navigation objects.

2.2 Specifying concern sensitive navigation objects

In order to give a deeper theoretical as well as design oriented view of our notion of CSN, we next explain the distinction between the specification of a navigation object and the specification of a concern-sensitive navigation object, i.e. a navigation object that supports being accessed using the ideas in CSN. As we show below this specification is determined by separating the object's properties which correspond to its type and those which correspond to the role it plays in its relationships with other navigation objects. As the role concept has many different semantics in the literature, in the following we tight our discussion to the definitions in [28] which have been then adopted in the work of [36, 5]. A set of precise definitions are given below.

Suppose that we want to specify a navigation object type N, by enumerating its properties (contents, outgoing links and operations). According to [40], N is a natural type when an instance of the type cannot stop belonging to the type without loosing

its identity, and its properties do not depend on any collaboration or relationship with another object. Such properties are called "Intrinsic properties". Using the example of Figure 3.1 above, a particular Mp3 player will always be a player and its base properties (brand, colour, base price) do not depend on any collaboration (in this case navigation path).

Meanwhile [28], a role type characterizes an entity according to the role it plays in relationship with other entities. An object playing a role can stop playing the role without loosing its identity. A role type T then expresses those additional properties that an object of type N exhibit when playing the role T. Given a navigation type N, for each concern C in which N can be accessed we define the N*in*C role type to express how an instance of N will be perceived when navigated in concern C.

In the example of Figure 3.2, the PlayerInIpodList role specifies those properties which pertain to a player when accessed from the list, in other words when being navigated in the concern of the list. Similarly, for each possible concern in which we can access the player and in which we want to use CSN, we need to specify the corresponding role type. For example players show additional properties when accessed from an Audible Device list; for example one can read why the product is advised, and a set of links to similar audible ready products. In this case the PlayerInAudible-Device role type will specify these additional properties.

Notice that roles (similarly to natural types) can be specialized and therefore we can build hierarchies of role types, therefore improving the specification of concern-sensitive properties. A thorough discussion on this subject is outside the scope of this paper and can be read in [28].

Summarizing, a navigation type N (e.g. a Player) exhibits:

- Properties intrinsic to the object type (i.e. which are present regardless the concern in which it is accessed). We call them core or natural or intrinsic properties;
- Properties which, given a concern Ci (e.g. Player*In*IpodList), correspond to the set of perceivable properties of N when accessed in the Concern Ci, i.e. when an instance of N plays the role N*in*Ci. These properties are expressed using the role notation in [28], which we also exercised in [37].

In Figure 4 we show the natural type Player and the two roles Player*In*AudibleDevice and Player*In*IpodList.

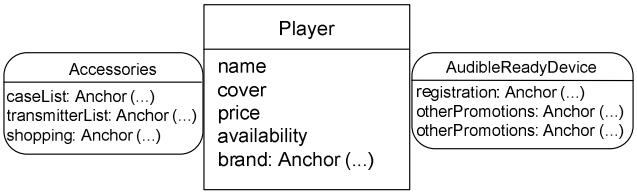

Figure 4: Intrinsic Properties vs. Properties in audible ready concern and accessories concern

By aligning the navigation object's properties to the concern in which it is being accessed, we improve the application's navigational structure, by making contents and links more focused to the actual concern the user is navigating (i.e. his intended task). Figure 5, shows a simplified navigational schema of the exemplar application in which we show how role types are used to indicate concern-sensitive access, e.g. when navigating from *IpodAccessories*, the Player*In*IpodList role is the target of the link.

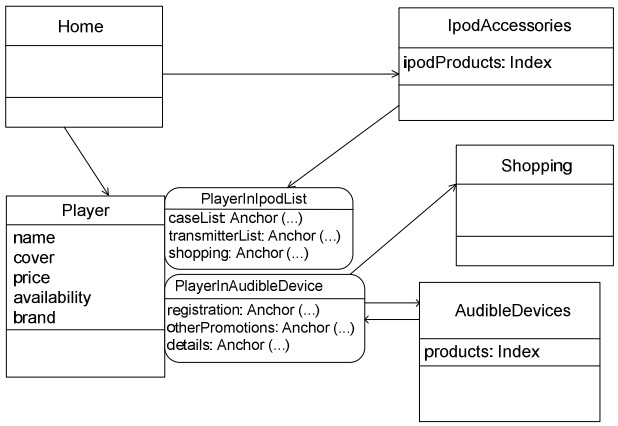

Figure 5: Navigational Diagram with roles

2.3 Enrichment Patterns and Types of Concerns which affect Navigation

In order to use the idea of CSN correctly and to maximize its advantages we need to understand how a navigation object can be enriched when accessed in different concerns. For the sake of understanding we will refer below to the kind of enrichments that can be realized using role objects.

There are basically three different types of enrichments, which can be of course combined:

- *New or modified contents*: A role can add attributes to the core class, or eventually can change the value of a specific attribute. In the role PlayerInAudibleDevice, the node is enhanced with new content describing the term of the discount offer.
- *New or modified Anchors and Links:* A role can add new links (and therefore anchors) to the core navigational class; eventually it can also re-define the target of a link. As an example new outgoing links are provided by PlayerInAudibleDevice role that introduces new navigation path to AudibleDevice and Shopping nodes.
- *New or modified operations*: Roles can improve the repertoire of actions or eventually modify a specific behavior. New operations simplify the user experience by exposing services which pertain to the current concern. An example of this

kind of introduction is found when playing *PlayerInAudibleDevice* role, it is possible to join to Amazon's AudibleListener Gold program by means of a new available operation called "Register".

A disciplined use of this approach and therefore a judicious use of role-based enrichments, would try to limit the core to those attributes, anchors, links and operations that are always valid. When an operation might have different meanings in different concerns, the designer must carefully analyze if it is reasonable to define it once in the core and re-define it for each concern. A more detailed discussion of these issues is outside the scope of this paper.

Though the kind of enrichment we need to add to a navigation object in a particular concern clearly depends on specific application issues, we can characterize the most usual enrichment patterns by analyzing the different types of concerns which arise in Web applications. In this way we can provide guidelines and good practices for a fruitful application of CSN.

As explained before a variety of concerns might arise in the development of a Web application. However, some of them impact on navigation (i.e. the user may access navigation objects in the context of such concerns), and therefore they might be a source of information for concern-sensitive improvement.

We next summarize the most important and recurrent concerns we can find in Web applications and for each one we indicate the most usual types of enrichments we find when accessing navigation objects in these concerns. We call them enrichment patterns as they can be expressed in a generic way and instantiated for each possible case of use. To describe them we use a simple template which consists of a description of the type of concern, a generic description of the enrichment and a discussion on the modeling issues, including a simple example.

- *Tasks concerns.* Most Web applications support the user in performing some tasks: the most usual ones are exploring products, managing the shopping cart, checking out, booking, bidding, entering comments, uploading content, tagging, etc. Some of these tasks might involve traversing different pages to be completed; for example for booking a room in a hotel, or renting a car we might need to enter information of different kinds; while browsing the shopping cart we might want to explore some related products, etc. It is clear that we need to ease the user's progress in the task.
  Enrichment: When the concern is defined by a task or business process (like in [39]), and operating on the target node might conflict with the process, it is advisable either to eliminate operations which collide with the concern or to add specific warnings (e.g. the shopping cart or checkout concerns in Amazon). When the task is performed in an inter-application basis (e.g. navigating from Facebook to Flickr), it is wise to add in the target application an indication of operations related with the source (e.g. adding a photo to the Facebook page). Figure 6 shows an instantiation of this pattern for the checkout process. The node Product is accessed in the check-out concern (e.g. because the user wants to confirm some specific product's properties). The role *InCheckout* contains an attribute with an indication of the concern, an anchor and corresponding link to return to the process and a re-definition of the *addToCart* operation.

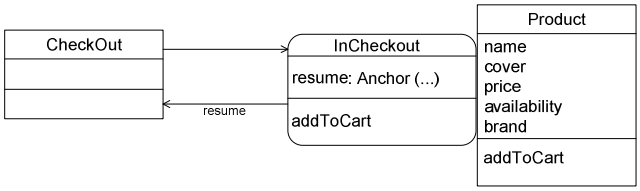

Figure 6: Task-based Enrichment for Check-Out process

- *Topic concerns:* Pure informational sites might introduce even finer-grained concerns; for example topics or themes such as in an Encyclopedia. Topic-based concerns are also present in the context of tasks; for example while searching books in Amazon.com, the genre of the book (thriller, travel, technical) or its theme area (Software Engineering, Programming, etc.) might itself become a concern. In a news site, the same news can be accessed with different optics: for example an article on a new economic measure can be read with a political view therefore adding links to other articles that make sense in this concern (and perhaps not in the "pure" economical one).

  Enrichment: When a node is accessed in that concern, add information and links specific to the topic which is related to the node. For example the Ipod in Figure 1.1 is enriched with links to other audio devices and in Figure 1.2 with links to others Ipod accessories. Figure 5 illustrates an example of a topic-based concern; in this case the topics are: "Audible Devices" and "iPodAccessories" in which each role *InTopic* (the two roles shown in Figure 5) adds different information. Figure 7 shows an example in an interactive tourist guide. The node cathedral can be accessed in three different concerns represented by the roles *History*, *Architecture*(at the right), and *PlaceOfWorship* (at the left). Notice that each of them adds specific information which enriches the core node. For example the *PlaceOfWorkship* role adds the *massSchedules* which only make sense if you access the cathedral specifically to know how to attend to mass.

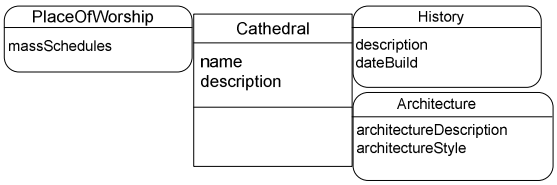

Figure 7: Different Topic Concerns for Tourist Application

- *"Pure" navigational concerns*, like Guided Tours or sets. These are usual abstractions in navigational design and therefore can be considered also as specific concerns. A clear understanding of these concerns allows improving navigation through the nodes belonging to the set.

Enrichment: When the concern can be represented as a set as in OOHDM navigational contexts [38] (e.g. the set of recommendations, etc.), it is wise to enrich the node with links to the index of the current set, and to the previous and next elements of the set. Another example of this kind of enrichment can be found in tag-based navigation like in Flickr (e.g. by providing links to other photos with the same tag). Also, a particular example of this kind of concerns arises when a task can be defined as set of steps which must be followed sequentially and there are no constrains from each step to the following. In Figure 8 we show an example for a guided tour of paintings. The core node Painting with the basic contents is enriched with a role *InGuidedTour* which adds the anchors and corresponding links to the next, previous and to the Guided Tour index.

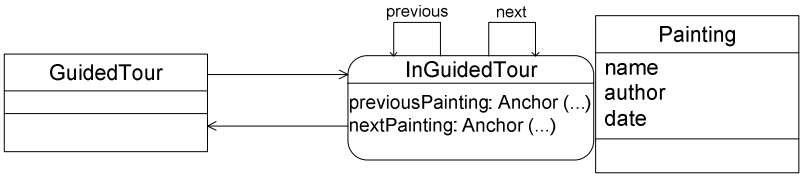

Figure 8: A pure navigational concern for a Guided Tour

2.4 Discussion

We have explained above how to specify concern-sensitive navigation objects. Before we show how to put these ideas in practice there are some additional problems to be solved. CSN provides the user with a specific kind of adaptive navigation experience as contents, links and operations adapt to his/her actual concern. Navigation objects might be enriched according to the concern for example with new information or operations which fit to the actual user task or navigation path. Regarding this initial definition there are some issues to consider:

- Notice that the perceivable properties do not depend on the user profile or identity, which means that CSN is slightly different from adaptive navigation (See the related work section).
- Besides the so called intrinsic properties, there might be properties which pertain to different concerns and which we want to exhibit permanently (e.g. the Add To Wish List operation in a bookstore), i.e. regardless the navigation path. Though this is a design issue not fully related with concern-driven navigation but with concern composition, we only give an overview of it (See Section 3.2).
- Defining the concerns which affect a node type requires a clear understanding of the application concerns, their relationships and the way they are reflected on navigation (See next section).
- There might be examples in which we want a concern to "persist" in much more than a single navigation step; for example, in Figure 4 suppose that the Player node has other (intrinsic) outgoing links, and we want to keep the actual concern (e.g. exploring iPod accessories). The correct way to indicate this in the diagram is to define the corresponding role (inIpodList) in the target nodes and re-define the intrinsic links in the PlayerInIpodList role. In this way we show the differ-

ence in the behavior of the intrinsic link when followed as a pure intrinsic property (e.g. independent of any concern), and when followed in different concerns.

- Obviously, all enrichments in the navigational model have their counterpart in the abstract interface model (since the interface of an enriched node is also enriched). While discussing abstract interface issues is outside the scope of the paper, we have already dealt with interface improvements in a related research [18]. The OOHDM abstract interface model, based on Abstract Data Views [38 can be enriched with roles in a straightforward way and the interface enrichment can be performed using XML transformations to "weave" the core interface with the interface counterpart of the navigational role as discussed in detail in [18].

3 Creating Concern-Sensitive Structures in Web Applications

Our main goal is to improve web application usability by realizing CSN. For the sake of clarity, in the following sub-sections we address this problem from two different points of view: a) how to build a brand new Web application supporting CSN and b) how to adapt an existing application to provide CSN.

3.1 Engineering CSN in a new Web Application

Though the main objective of this paper is to show how to add CSN functionality to existing Web (particularly Web 2.0) applications in a dynamic way, we next summarize how we can build Web applications which provide CSN functionality. The whole development process, as well as the associated implementation, and the architectural issues have been discussed in [32]. Here we present a brief outline of the approach to demonstrate its feasibility. We only focus on intra-application CSN, i.e. we suppose that all the navigation occurs within the boundaries of a single application.

Our approach uses a symmetric style for separation of concerns [8] which implies a separated development process per each navigation concern. This kind of separation of concerns lets to easily enable or disable specific concerns depending on the concern volatility (e.g. functionality for donations which are activated in an e-commerce site after a catastrophe and deactivated after a while). We have developed a light extension of the UML class diagram notation and applied it to the OOHDM design model, by "just" adding some syntactic sugar in order to have separate models for each application concern, and some new modeling artifacts and notations.

First, during the requirement modeling process, we identify the relevant application concerns, and for each of them, we describe the possible interaction and navigation sequences using User Interaction Diagrams [22, 38]. These diagrams give us the first clue about the variants exhibited by each information object when being accessed in different concerns.

Afterwards, and using the information collected from the UIDs we obtain a conceptual model for each application concern. This model can be obtained in two different ways: using model transformation tools which map the UIDs onto classes or by applying heuristics. Considering that many times UIDs are drawn informally (e.g. in a paper-based style) during discussions with customers, the "manual" approach is often

used. In [38] there is a thorough description of the heuristics which intuitively consist in considering each UID "data structure" a potential class in the conceptual model and each transition in the UID diagram as a potential association among classes.

According to OOHDM and the light extension described in [32], each concern is modeled in a different UML package. In these packages, classes with the same name might exhibit slightly different attributes and operations, those which pertain to the corresponding concern.

During the navigational modeling stage, we first build a unique and concern-free navigational diagram, which shows those nodes (including properties and operations) and links that are common to all application concerns. Once again we use the information gathered from the UIDs to build a table showing how each node is enriched when accessed in each concern. This table allows us to build a concern-sensitive diagram, using the role notation as shown in Figure 5 above.

The roles-enriched navigational model can be mapped to an implementation either by following some simple heuristics or in a semi-automatic way using a model-driven development environment. We have used both approaches in our previous research by extending our tool Cazon [18] with functionality to weave navigational concerns dynamically. We have also shown elsewhere [19] that using XSLT transformation engines we can provide a solution which allows to weave concerns at run-time, and also to "undo" the weaving process by disabling concerns which are no longer necessary in the application.

The main disadvantage of the Cazon-based solution is that it requires that the nodes are specified as XML structure instead of working on the role-based navigational diagrams. To solve this problem we implemented our extended UML notation by incorporating the role concept into UML (using some new stereotypes); next we define a corresponding profile and use it in the MagicUWE [6] model-driven development tool. We finally defined the corresponding model to code transformations to generate a running application using a pure model-driven development style.

3.2 Adapting Existing Web Applications with CSN

Even though we could find good examples of CSN in current mainstream Web applications (See the examples in Section 1), most of them only provide a limited set of CSN functionality; in some cases it is possible to find support for set-based navigation in guided tours (See for example The guided tour of the Roman Open Air Museum [43]), following the idea of OOHDM navigational contexts [38] or support for navigation in the context of business processes [39].

However, we have found that it is feasible to incorporate CSN functionality in already built applications, even if they were not conceived with a methodology like the one outlined in Section 3.1. Adaptation to CSN in existing Web applications can be realized in two different ways: a) Server-based adaptation, b) Client-based adaptation. We next discuss both alternatives as an introduction to Section 4.

3.2.1 Server-Based Adaptation

The problem of adapting an existing Web application can be seen as an example of application re-engineering. From this point of view one possible approach is to use the ideas of 3.1, produce a new design model and generate a completely new applica-

tion where the new CSN features are present. However, this alternative might be too costly, moreover taking into account that we will not change application business rules but "only" its navigation and interface features.

More cost-effective alternatives strongly depend on the implementation support. For example the usage of mature Web Model-View-Controller frameworks, which are based on well-known design patterns [17], makes it easier to introduce CSN concept. Frameworks like Struts [41], Django [12], etc. provide the notion of *Filter* which intercepts and decorates web pages. These Filters can be used to implement CSN as decorators. As these frameworks also allow a clever separation and encapsulation of presentation logic using the Command pattern [17], it is possible to add specific presentation logic (e.g. node attributes computation) of a navigation concern in an unobtrusive fashion.

When it is necessary to modify source code to achieve CSN behavior, i.e. when the intervention is "obtrusive", we might create further maintenance problems. For example, if the code becomes tangled, the application evolution will be more complicated. In this case a better and seamless approach can be applied; this consists in decorating the application user interface using a transparent proxy wrapping the whole application like MonkeyGrease [30] does. Each HTTP request done to the decorated application is post processed by the proxy, thereby transforming the interface structure and enriching the content. This approach will be exploited in the next sections when we discuss client-based adaptation.

3.2.2 Client-Based Adaptation

One of the new trends in the Web 2.0 is the possibility to use client scripting to customize the contents, layout and styles of Web applications. This kind of intervention, called "tuning" in [11] is becoming increasingly popular since the emergence of weavers such as *GreaseMonkey* (GM). Basically, it is possible to program simple scripts (e.g. in JavaScript) which run in the browser context (and are realized as browser plug-ins), access the actual page's DOM tree and eventually change it by modifying, adding or deleting nodes of the DOM tree to adapt the contents, links and interface style to the user's wish.

We have found that using this approach, it is more than feasible to extend the scope of CSN to a broader set of applications. For example, with a simple script we could introduce new entries to Wikipedia's sidebar component. In Figure 9, the original page at left is transformed with a GreaseMonkey script resulting in new menu entries as it is shown with a dotted box at right.

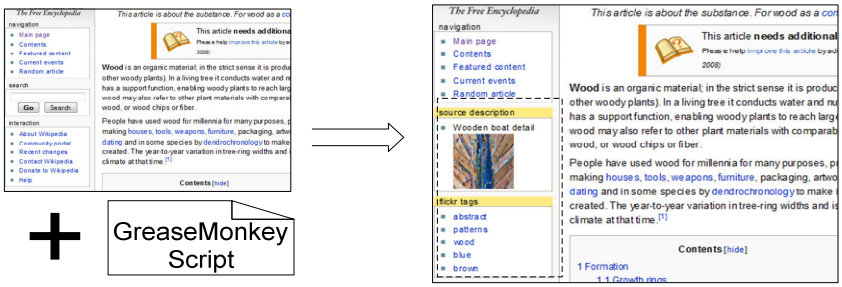

Figure 9: Enhancing Wikipedia with GreaseMonkey

These scripts can be made public and then shared by the community of users that uses the adapted sites. Depending on their popularity, user's scripts have been later incorporated as native features of the application**.** For instance, initially Gmail didn't provide a "delete" button on the mail list's header for deleting selected mails but GreaseMonkey community contributed this feature[2] which later became part of Gmail. This application has included other functionalities, such as Google Calendar integration into Gmail[3] or User interface (UI from now) layout reordering[4], that first were implemented by the users community

While this approach is feasible and powerful, we found that it can be improved to provide inter-application CSN; in the following sections we elaborate this idea and present a framework for CSN in Web 2.0 applications.

4 Broadening the Scope of CSN. Examples and Feasibility

One of the aims of our research is to provide tools to realize CSN in existing applications in a non-intrusive way. For this reason we conducted a thorough study of a large number of popular applications, from typical e-stores such as Amazon.com, to interactive encyclopedia like Wikipedia and sites such as Youtube, Flickr, Picassa, Facebook, LinkedIn [29], etc. We assessed the following issues: does CSN provide substantial improvement when used in these applications? Is it feasible to implement CSN? Which technical problems must we solve? Which are the "boundaries" for realizing CSN? What tools are necessary for this endeavor? For the sake of clarity we separately address each of these questions in the following sub-sections.

4.1 Enriching Navigational Schemas in an intra-application basis

In [32] we showed that CSN provides a more flexible navigation structure therefore improving the user experience by easing the access to information corresponding to the actual concern owing to a more opportunistic organization of information. This is true, regardless of the type of application we are navigating. For instance, Wikipedia allows to group articles within a Book structure for sharing with other users or for easy printing. In practice, the book reviewing process in the current design is not an easy task due to the fact that the book concern is lost after traversing the link from the book index to one of its attached articles. To illustrate this example, Figure 10.1 shows the initial design of a book article of *Jorge Luis Borges* without any feature that helps the books edition. On the contrary, on Figure 10.2, we present an enhanced article interface with a topic concern called *Book article.* This concern introduces components pointed with a dotted box which provides contextual information and allows browsing adjoining articles.

---

[2] Gmail Delete Button script - http://userscripts.org/scripts/review/1345
[3] Gmail + Google Calendar script - http://userscripts.org/scripts/show/9411
[4] Gmail: Chat Right script - http://userscripts.org/scripts/show/4665

Figure 10.1: Borges's wikipedia article

Figure 10.2: Article with Book Edition concern

Figure 11 shows the new navigational model for Wikipedia enriched with a decorator which enhances the *Article* node when the user accesses it from BookEdition.

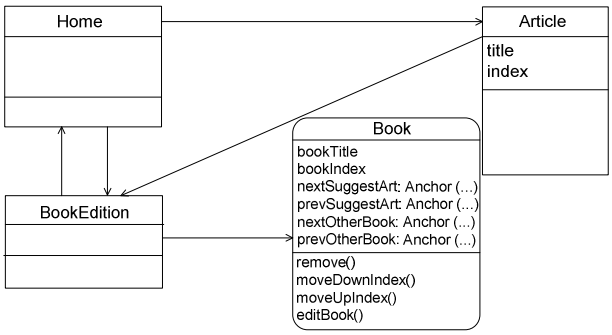

Figure 11: A navigational schema for the Wikipedia Book Concern

The schema above shows how the book concern "contributes" to enrich the information presented in a Wikipedia article to improve usability. In this case the use of CSN is bounded to Wikipedia and its underlying concerns. We define "intra-application" CSN as a CSN navigation occurring within the boundaries of a single application.

4.2 Keeping Navigational Concerns persistent through different applications

In our analysis we have also found that the same concept can be applied with a broader view by analyzing concerns which spam through different applications, i.e. beyond the previously shown "intra-application" examples.

In Figure 12, we show a topic-based CSN that comprises Delicious (as source of navigation) and Wikipedia (as target of navigation). The left of the figure displays tagged links to Wikipedia and the right of the figure displays a Wikipedia page enriched with the Delicious navigational concern.

In order to take advantage of information gathered from tags, when users navigate from Delicious to Wikipedia by traversing one of Delicious's tagged links, the Wikipedia article is enriched with a set of links to similar pages that have the same tags set as current URL does at Delicious. For each tag associated with the URL, a link should also point to its Wikipedia meaning.

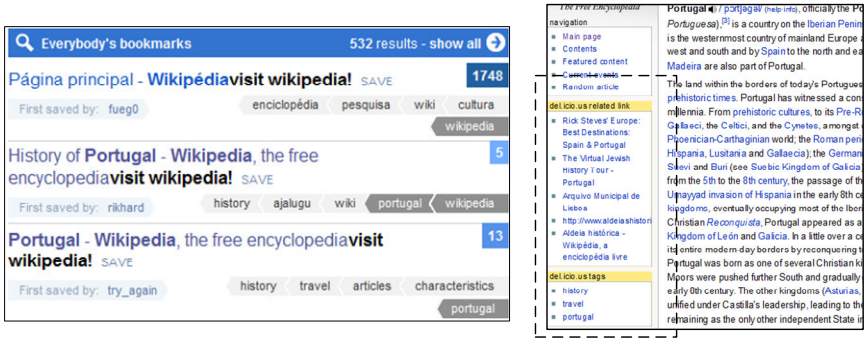

Figure 12: showing inter-application CSN between Delicious and Wikipedia

Notice that Wikipedia has to be enriched with different navigational concerns depending on the user's navigation path. This behavior improves user experience by affording facilities which helps keeping current user concern shared and reused for several applications. In Figure 13, we show the navigational models of the above mentioned example where two application packages demarcate applications boundaries. Note that when the user navigates from Delicious search node to Wikipedia article node, this is enhanced by adding information about the resulting posts. With a dashed line we show the original navigation.

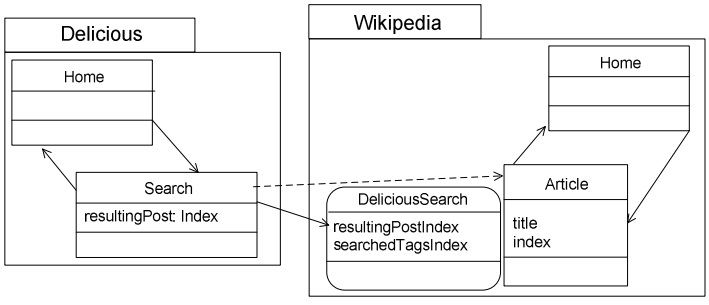

Figure 13: Navigational Schema between Delicious and Wikipedia

We can extend the example above and provide different enrichments to Wikipedia according to the link's origin, as schematically shown in Figure 14. More generally,

we define “inter-application” CSN as CSN navigation where we can use concerns in a source Web application to enrich contents in a target application.

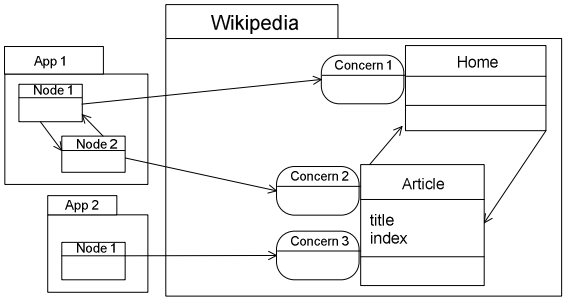

Figure 14: Inter-application CSN for Wikipedia

The principles in intra and inter-application CSN are basically the same and as a consequence the improvements of navigability are somewhat similar. While we have largely discussed intra-application CSN in this paper and in [32], a brief comment is needed to assess inter-application CSN. In all our experiments with users (some of them briefly reported in 5.4), we have found that CSN is more than welcomed in an inter-application basis; the reason is that it is easier to lose the navigation context when departing from an application to another. In this sense when an application’s page is enriched with contents and links which depend on the application from which we navigate, we can keep the navigation concern explicit and also active by offering these new context and links. As discussed in Section 2.4 a navigational concern can be kept active in navigation paths of any length, provided the corresponding roles have been clearly specified in the target nodes. For example if we navigate from Delicious to Wikipedia and then to Flickr, we can specify that the Flickr node opens using either a “FromWikipedia” role, a “FromDelicious” role or none of them by indicating it in the Flickr roles (if any) and in the outgoing links of Wikipedia and the *Delicious-Search* role of Figure 13.

4.3 Feasibility and problems

Considering our idea of realizing CSN on top of existing applications it is important to analyze architectural and design restrictions, particularly when we aim to use CSN in an “inter-application” style.

Some applications provide extensions facilities at the server side (for example Facebook [13] and LinkedIn [29]), while others provide extensions facilities at the client side (for example Youtube [45]), and most of the time none of them allow implementing CSN facilities easily. The obvious alternative to make CSN feasible is to run scripts, implemented as browser’s plug-ins at the client side.

The implantation of CSN functionality depends on mainly two significant factors: concern scenario and application’s extensibility. CSN can be implemented in a single application (intra-application CSN) which just requires understanding the application

underlying model for enriching it. On the other hand, when the navigational concern comprises several applications (inter-application CSN), we must understand the two models, the way the applications are connected and how CSN is realized in the target application with information from the source one. For instance, when analyzing how to improve navigation from Flickr to Wikipedia, the *Tag* concern in Flickr might require a "translation" to Wikipedia's Article.

While the implementation of scripts to adapt an application at the client-side is feasible, there are some problems regarding the needed effort and the stability of the result. Ad-hoc plug-ins which realize a solution for a specific concern need to access directly any low-level page structure like specific HTML tags, or registering standard HTML event listener to DOM. This is usually achieved by directly manipulating the DOM tree; as said before this process can be simplified as some applications like Gmail and YouTube are providing APIs [20, 46] for easier access to already defined UI components and underlying data.

We have done some experiments using the GreaseMonkey engine for weaving concern-specific elements onto the currently navigated page. This engine takes the current page, and appends a JavaScript file which introspects the host page by applying structural changes over it.

In Figure 15, we show a UML activity diagram, corresponding to the process that occurs when the user loads a page into the browser, and a plug-in is defined for it.

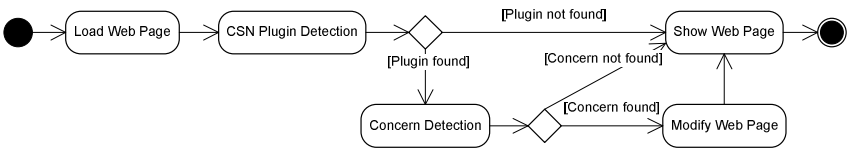

Figure 15: Activity diagram, client-side weaving process

However, when this approach is used, each specific implementation must be done from scratch, thus hindering extensibility and reuse. Additionally if the being adapted (e.g. Wikipedia) changes, we have to face a serious problem of maintenance in our scripts. We analyze this issue in the following sections.

4.4 Improving the realization of CSN

While providing plug-ins for different applications is an important step forward to realize CSN in Web software (even considering the evolution issues mentioned before), we have also found that it is possible to provide better support for CSN by analyzing families of similar applications, such as social sites (Facebook, LinkedIn, MySpace [31]), encyclopedia (Wikipedia, Knol [26]), E-stores (Amazon, Barnes and Noble [3], etc.), image and photo repositories (Flickr, Picassa, etc.)[5]. By using techniques of product-line engineering [10], it is possible to abstract the common con-

[5] We do not pretend to establish a formal classification here as it can be argued that Wikis and repositories are specializations of a more generic type of applications of user generated content

cerns in these applications and build generic tools which provide the basis for CSN without considering the specific applications' look and feels. In this way we can derive concrete plug-ins either by providing the data needed for a member of the family or by just specializing the corresponding scripts.

Using the same ideas we can build support for inter-application CSN first by focusing on two specific applications, e.g. Facebook to Flickr and then generalizing the source and the target and building generic support for CSN between social networks and repositories.

To make this discussion more concrete, in the following section we formalize our approach for client-side adaptation.

5 A High-Level approach for Client-Side CSN Adaptation

As previously explained, an application can be adapted at the client-side with CSN features by means of scripts which act as browser's plug-ins. These scripts are designed to detect that navigation initiated in a source page correspond to a specific concern, and to apply the modifications or adaptations needed to the target node in such a way that the information, operations and all remaining issues related to the concern are introduced.

Even though the task of writing scripts is often considered a minor activity which can be done by final users, we consider that client-side adaptation, specifically to introduce navigations improvements like in CSN, should be tackled with a high level, methodological approach. We have developed a simple process to guide developers in the implementation of CSN at the application's client side and a tool to simplify the development of scripts.

The process is an adaptation of the one explained in Section 3.1, and comprises the following steps:

1. Identify suitable navigational concerns. This stage consists in understanding the main tasks and themes in the underlying application (s).
2. Analyze target node types and specify which enhancements these nodes might have when accessed in each of the corresponding concerns. These enhancements will be realized as decorators; as explained before, might involve low-level tasks which manipulate the DOM for adding new contents or links.
3. Determine those source node types which give origin to a CSN. From these nodes we must extract the information needed to accomplish the transformation in the target node. This information will be *concepts* for a topic CSN or *actions* for a task CSN. Additionally a navigational or interface enhancement may be required, when the source node does not provide links to the target and we aim to have this navigation. Again the DOM will have to be accessed to retrieve some piece of data or creating new document leaf.
4. Implement the corresponding scripts for both the source and target node types. The main responsibilities of these scripts are the following. The source script will check if there is a link between the source and target application and will store the concern information required for realizing CSN in the tar-

get. The target script will introduce the adaptations into the corresponding DOM elements; this script will use the information stored by the source script.

These tasks which require accessing the DOM document are simplified by a visual tool, which also generates a script template that the developer completes with details of the concrete application. As a result of this process, we get two scripts that run respectively at the source node and target nodes. These scripts can be generic and then reused to form others CSN plug-ins, when some good practices are followed as discussed in the following sub-sections. In particular we show how to achieve increasing levels of abstraction in plug-ins development for CSN.

5.1 Adding Abstraction Mechanisms to Script Development

Ad-hoc implementations of client-side CSN, briefly described in the previous subsection can be tedious and error prone as scripts must deal with low level page details. Additionally, and considering the usual "permanent beta" state of Web 2.0 software, when the underlying application changes scripts might become useless. Therefore the need for an abstraction layer becomes mandatory in order to offer the developer a more maintainable and clear platform to help him focus in the development of the extensions, instead of the burden of interpreting how a Web page is constructed and how it can be extended. This new layer decouples the access to low-level UI component by using an adaptor [17]. In Figure 16, we show a schema of the abstraction layer - represented as a dotted box-.

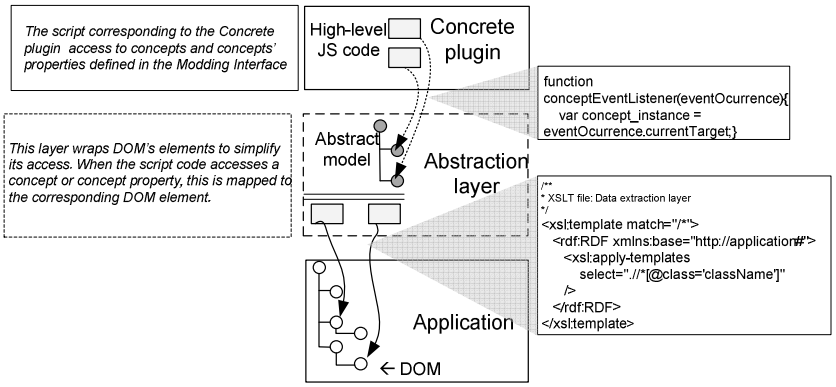

Figure 16: Abstraction layer architecture

To specify the abstraction layer, we have used the Modding Interface (MI) technology presented at [11] which is based on a UI ontology specification. The MI itself is divided in two (sub) layers: (1) a specification containing abstractions about the concepts shown in the page (2) a sub layer to extract these concepts from the UI of the underlying Web page; we call this layer "*data abstraction layer*". The first layer provides an Application Programming Interface (API) that serves as an interface which

allows avoiding references to volatile structures in the CSN code. The second one is a XSLT file that extracts the concepts occurrence from the Web page. When the underlying Web page changes, only the *data extraction layer* must be updated. The main idea behind the MI is to register JavaScript functions as listeners of conceptual events defined for the concepts in the ontology. In our use of the MI all CSN adaptations will be performed using these functions.

According to this idea, when the user opens a web site, the MI specification – identified as a WWW resource with URL- is included in the web page. After processing the specification, a view model is available into the page scope enabling different scripts to access to UI components and manipulating them.

If the underlying application changes its UI structure, we will only need to update the way in which the MI resolves data from the UI, while the CSN scripts remain usable because the interface encapsulates the access to DOM elements.

As a first example, suppose that we want to engineer CSN between Flickr and Wikipedia in such a way that Flickr *tags* can be searched in Wikipedia and that Wikipedia (the target application) will provide additional content and links when accessed from Flickr.

The first step is to create MIs for both Flickr and Wikipedia; the effort in building these interfaces can be rewarded if later we aim to build different CSN structures involving Flickr and Wikipedia (either for sources or targets in navigation).

The MI for Flickr defines the following concepts: *Tag* with the property *name* and the conceptual event *TagLoad*; in the other hand, the concept *UserComment* has two properties named *user* and *comment,* and a conceptual event named *UserComment-Load*. Respectively the MI for Wikipedia (shown in Figure 17) defines the concept of *Menu*, with its properties *title*, and *body* and conceptual events such as *MenuLoad*. With this specification, developers can extend the Wikipedia navigational functionality in an ordered way.

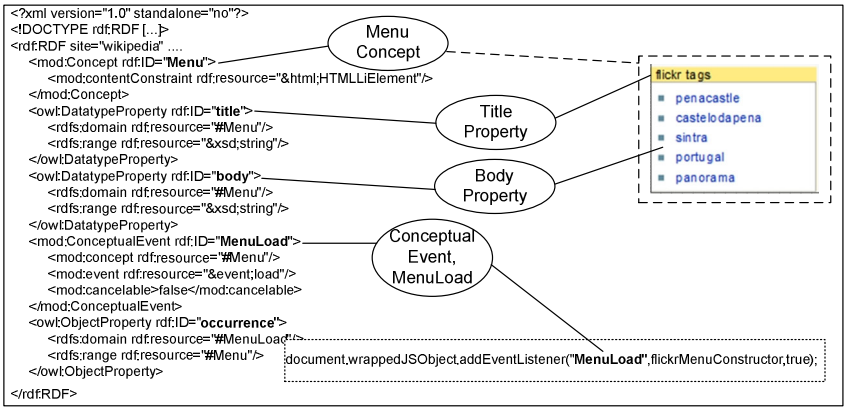

Figure 17: Sketch of Wikipedia Modding Interface

Next we have to create the corresponding scripts. The one running on top of Flickr adds some anchors to allow navigation to Wikipedia, and stores (temporally) the actual user concern and the tags defined in the Flickr page, which will be used by the Wikipedia script. When the user navigates to Wikipedia, the Wikipedia script per-

forms the following actions: when activated (which means that we are navigating from Wikipedia), it restores the information saved by the source script and, using this information, the corresponding links are added into the DOM element wrapped by the *Menu* concept.

In Figure 18 we illustrate the example. The left part displays a picture corresponding to Da Pena's palace hosted by Flickr and, the right part a Wikipedia's page about Portugal – which is reached when users navigate using the Flickr's tag –. In the Flickr image the dotted box highlights the picture's tags which will act as a *Tag navigational concern* for the Wikipedia application. In Wikipedia, a dotted rectangle indicates the navigational concern contributions: contextual information about the source page – in this case information about the palace – and the set of tags that describes the palace where each one is searchable into Wikipedia.

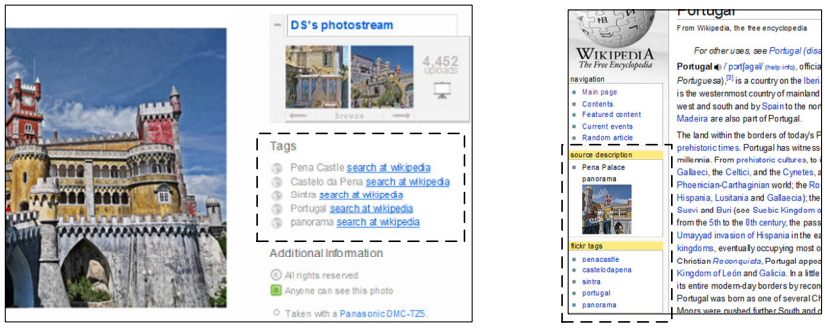

Figure 18: Showing inter-application CSN between Flickr and Wikipedia

If we now aim to go further to add CSN to Wikipedia (e.g. when navigation starts in Delicious) we should either write another script for Wikipedia (which is equivalent to define the corresponding role object as shown in Section 2) which will perform the corresponding tasks to provide the concern enrichment.

When the enrichment is similar (e.g. like the one in Figure 17), the existing script could be reused or specialized depending on (a) what information is necessary for the enrichment (b) how this information is saved by the source script. In the worst case, if the source application model is different i.e. if the concepts defined are different from the ones in Flickr, the developer must change the way in which the Wikipedia script restore the information. On the other hand, if the source application model has defined the same concept (e.g. *Tag* with the property *name*) then, the script can be completely reused.

Though the MI provides a high level abstract view of the underlying application structure, the process of creating the MI might be tedious as well as the development of the corresponding scripts. In the following sub-section we present our approach for simplifying these tasks.

5.2 Visual Specification of CSN Structures

We have developed a simple, visual tool to simplify the process of client-side adaptation of existing Web applications. This tool helps users in the CSN development process, covering steps 2, 3 and 4 as depicted in section 5. Our final aim is to automate the whole process described in the beginning of this section. To make the description of the tool more understandable we exemplify here showing the most significant steps to develop the adaptation in Figure 18. The first step is to specify the MI, which is done by choosing meaningful concepts from the application's UI as shown in Figure 19.1 on the left. On the right of Figure 19.1 we show the dialog used to complete the specification of the concept, including properties and events. This process can be incremental, i.e. the MI can be defined "opportunistically" to be useful for the current adaptation, and completed later when new adaptations require more concepts to be defined. Additionally and even though the script scope (the applications in which it "runs") is configurable from GreaseMonkey, our tool allows defining that scope, by setting a set of URLs or URL patterns as in GreaseMonkey which is useful for applications sharing similar concepts.

The result of this process is the ontology describing the application model and an XSLT file – this is the *data extraction layer* - which is used to extract the occurrences of the concepts defined in the application model (i.e., its ontology), from the concrete application. Figure 19.2 shows the MI Viewer which shows all the defined concepts with their corresponding properties and events.

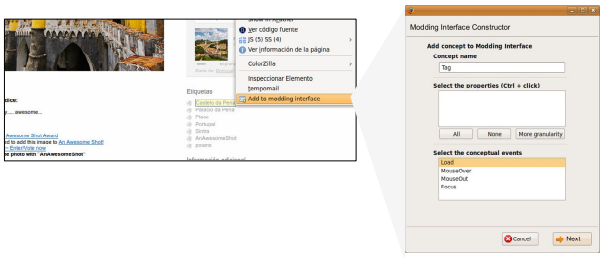

Figure 19.1: Adding concepts to the Modding Interface

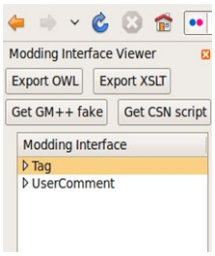

Figure 19.2 MI Viewer

The next step is to create the script for the source application. In our example we first need to specify those anchors which will trigger CSN. As shown in Figure 20.1 this is done by selecting "Create anchor as CSN source…". This option will open a dialog to allow adding a link for each occurrence of the selected concept (in this case a *Tag*), eventually using information of this concept for the anchor's attributes. In this way we enrich the tags shown in Figure 18 with the *Search at Wikipedia* text. As we have only selected the *Tag* concept, only occurrences of this concept will be stored to be used in the target node. In Figure 20.2 we show a code section corresponding to the script generated for Flickr. We show how the anchor required by the plug-in is added. Note that, after the script is generated, the developer may eventually modify how the attribute *href* is made within the function *generateAnchorForTag*.

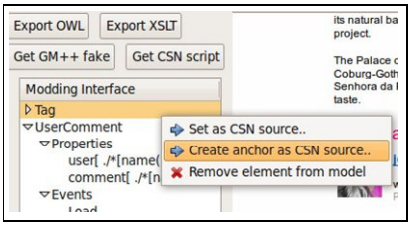

Figure 20.1: Create new plug-in

```
function generateAnchorForTag(value){
  var anchor = document.createElement("a");
  anchor.setAttribute("href","http://en.wikipedia.org/wiki/" + value.split());
  anchor.innerHTML = "search at wikipedia!";
  return anchor;
};
function getPropertyValue(conceptOcurrence,property){
  return conceptOcurrence.getElementsByTagName(property).item(0).nodeValue;
};
function topTagListener(eventOcurrence){
  var tag = eventOcurrence.currentTarget;
  var nameTag = getPropertyValue(tag, "name");
  var anchor = generateAnchorForTag(nameTag);
  addAnchorAsCSNSource(tag, anchor);
};
document.wrappedJSObject.addEventListener("Tag", topTagListener, true);
```

Figure 20.2: Script section

The function *addAnchorAsCSNSource* is the place where the behaviour to save the information in memory, is added to the link (in low-level terms, by setting a value for *onclick* attribute), and it is finally the function that appends the new anchor into the DOM.

Next we create the script for the target application (Wikipedia). By using the tool we navigate to Wikipedia, the tool indicates that we have information to be used in a CSN adaptation (the stored occurrences of the *Tag* concept in Flickr). By accepting the suggestion (Figure 21 top) a configuration dialog is open. First we choose the kind of CSN enrichment we want to add to Wikipedia; the enrichment options are based on those explained in section 2.3. In the second step we can choose how to use the information available (information which has been saved by the source script). Finally, in the last step we decide where this information will be used. Note that, perhaps, we will want to use the information when the menu is loaded (Load event); in other cases we might make those links available when another event, such as *MouseOver* occurs.

As a result of this process we export the corresponding GreaseMonkey scripts which will include the listeners for the concept *Menu*, the functions to access the occurrences of *Tag* in Flickr and an API to create the new links corresponding to the concern.

For simple CSN adaptations, as those shown in Figure 18, the developer can be released of all the low-level burden of script programming. For more complex adaptations our tool generates a script template to be completed by the developer.

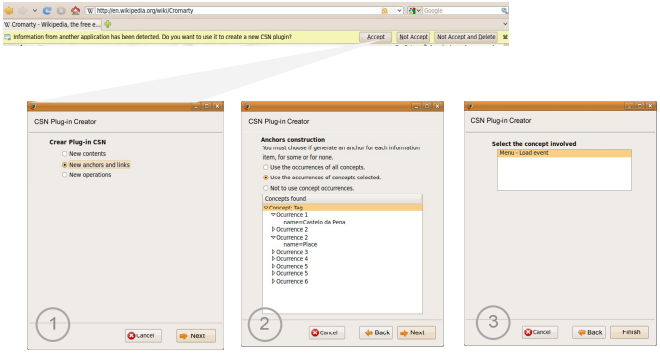

Figure 21: Selecting Concepts to enrich target application

5.3 Improving development for application families

As explained in Section 4.4, applications in the same "family" usually deal with similar concepts and therefore involve the same navigational concerns. For example, applications such as LinkedIn, FaceBook, Hi5[23], etc. have similar domain models, and provide a set of common functionality. In this context, once we have built support for navigational concerns involving one of the members of the family, we could use the experience to reduce the effort in subsequent developments in the same family.

In some way, the abstraction process to generalize a concrete CSN development (e.g. for LinkedIn) to a broader set of applications (other social networks) is a simplified example of framework development; for the sake of conciseness we explain only its more important steps.

First, we need to specify a common ontology with the shared concepts in the family. These concepts will be used by the "family" scripts.

Next we specify the *data extraction layer* for each application in the family; this is mandatory because even sharing the same concepts, the mapping into user interface objects is usually different. Now we are able to create more generic concrete scripts, i.e. scripts which can work with any application of the family. These scripts will work on the shared concepts defined by the common ontology. For instance, we can develop a generic script for social communities, covering contact information management.

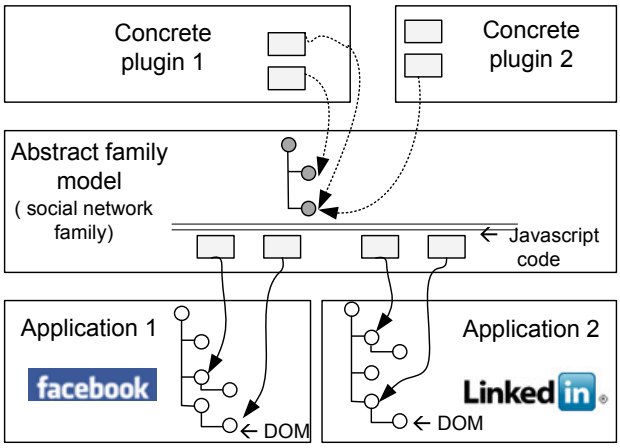

Figure 22: Family Abstract plug-in architecture

In Figure 22 we show how an *abstract family* model provides a model for giving support to concrete scripts which enhance social network applications. Each concrete plug-in uses the abstract model, applying specific changes, and these are translated to DOM changes at the underlying application such as Facebook or LinkedIn. A slightly more complex model is necessary when we also wish to exploit the differences existing in members of the family. For example, the concept of *Company* is outstanding in LinkedIn but irrelevant in Hi5. The two main differences in these cases (not shown in the Figure 16) are the following: first, the scripts will be "proprietary" for the specific application, and second it will use concepts not defined in the Family model. This sit-

uation can be solved by either including the "different" concepts in the family model (but somewhat corrupting its structure or making it more complex), or by using additional layers which contains each application's model and using these layers when required.

With these ideas in mind, we have implemented the Delicious CSN plug-in, presented in section 4.1, using a MI specification. In this way, the Delicious script accesses to elements defined in the MI showed in Figure 23 (for the sake of conciseness we only show the properties defined for the concept *Post*, however, it must be noted that the concept *Post* and the conceptual event *PostLoad* are included too). Suppose that we want to use the Delicious script with another application, like Google Marks. As we can find the same concepts in Google Marks, and considering that the Delicious script only accesses to concepts occurrences, we can use the same MI for Google Marks and the script defined originally for Delicious, will work unchanged with Google Marks too. Note that, we only should implement a new *data extraction layer* to found the *Post* occurrences into Google Marks DOM.

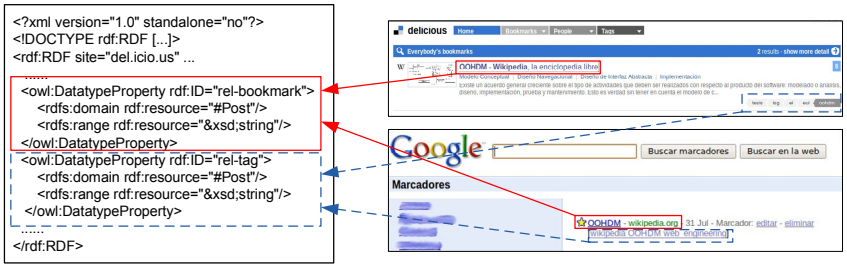

Figure 23: Tag family's abstract model

5.4 Towards good design practices for CSN

We have conducted several experiments with users to assess the usability of applications enhanced with CSN; this experience has contributed our research with a valuable feedback and allowed us to improve the user interface aspects of our plug-ins.

The first and one of the most interesting feedbacks is the need to provide contextual information about the current navigational concern and how it was triggered. In our first developments, we just concentrated in the enhancement of the target page with components that support CSN (such as new information, links and operations), but users complained about the lack of informative description of which topic or task originated the enrichment. For instance in the *Tag navigational concern* example between Flickr and Wikipedia (Figure 18), after navigating a tag link to the corresponding page in Wikipedia, users asked for some information that helped to realize why they were there. In the example, they needed to know that the Portugal page was reached from Da Pena's palace (See Figure 18).

Additionally users usually found hard to distinguish a specific navigational concern in pages saturated of data. As one of the consequences of this observation, facilities for CSN should be designed to be recognizable to the user, easing its localization

and usability. In Figure 24 we show how a Youtube video can be enhanced when being accessed from a Facebook post. Note that, in the example, the actual user concern - i.e. Facebook's information and functionalities – is highlighted so that the user can find it easily. In this case it is also important that the added content uses a style similar to the "source" application.

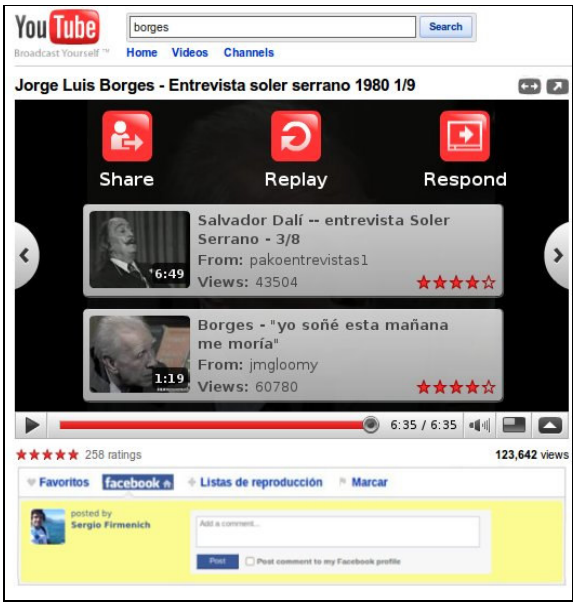

Figure 24: Facebook concern in Youtube

When inter-application navigation occurs, the current user concern must "travel" from one application to the other; however, very often target applications do not share the same domain business model as the corresponding source applications. Therefore, some kind of ontology alignments is demanded, transforming the involved concern data for being compliant with the navigation target semantics. For instance, in our examples, we aligned Flicker's tag concept with Wikipedia's article concept. This operation required the participation of a designer who was responsible for adapting both models.

We are currently defining a set of good practices for developers to use when extending Web applications with new navigational operations, being them restricted to CSN or more general.

6 Related Work

Separation of concerns has been a driving force in Software (and Web) Engineering for years since the seminal work of Parnas [34]; the driving force for the research in this area has been, as discussed before, to improve modularity and therefore simplify maintenance, evolution and reuse. Advanced design techniques such as design

patterns [17] or brand new approaches such as Aspect-Oriented software design (AOSD) [14] have as their objective to provide better ways to isolate different application concerns and provide different types of support for the late weaving of software components which realize these concerns.

In the Web Engineering field there has been also an early recognition of the need to separate the coarse grained concerns of Web software in the different stages of development. In this way, all mature approaches [38,7, 27, 25] separate business logic from navigation and presentation issues. More recently these approaches have incorporated elements of AOSD to deal with further application themes such as adaptation, security, etc. [27, 33].

In our research we have used these ideas but with a different objective. In [32] we introduced concern-sensitive navigation and demonstrated how to systematically produce better navigational structures by using separation of concerns, generalized some existing approaches to enhance navigation in specific contexts such as sets of related objects (called Navigational Contexts in OOHDM [38]) or business processes [39].

While in [32] we use our understanding of application requirements when building applications from scratch to improve usability, in this paper we go further by using the basic ideas of CSN as a starting point to work on existing applications. Our contribution is manyfold. First, we extend the scope of CSN to existing applications and show that CSN enrichments can be done on a server or client-based style. Second, we show how to use the same ideas in an inter-application basis and finally we present a set of concepts and tools to aid the developer in building CSN structures in a client-based way, by transparently improving existing applications' interfaces.

CSN has some points in common with the work on adaptive hypermedia [1]. Adaptive hypermedia approaches seek to improve user's navigation by taking into account the user's profile and needs. In an adaptive hypermedia application, nodes and links vary according to the characteristics of the user, his navigation history, etc. Adaptive hypermedia systems rely on a user model which represents the meaningful user's features and an adaptation model in which the adaptation rules and algorithms are specified. The increasing interest in adaptive hypermedia has motivated that most well known Web engineering approaches have extended their modeling repertoire to describe different kinds of adaptive behaviors. For example WebML [9] provides facilities for building context-aware Web applications, i.e. those applications which adapt to the user's context (his profile, location, interests, etc.) describing the adaptation using event-condition-action rules. UWE meanwhile supports adaptive navigation [4] using aspects focusing mainly in links adaptation.

CSN, as briefly indicated in Section 2.4, has some points in common with these works since our work also aims at improving user's navigation by adding contents or links to the visited nodes. Clearly, while the adaptation actions that can be specified in CSN and adaptive hypermedia are similar, the adaptation conditions in adaptive hypermedia or context-aware Web approaches are richer than in CSN because these techniques consider different aspects of the user, while CSN only takes into account his current navigation interest (his actual concern).

In this way, CSN provides a useful type of adaptation (as shown in all the preceding examples) while requiring less resources than adaptive hypermedia techniques. Specifically, it is not necessary to build a user model since all applications will behave similarly for different users. Besides, the adaptation actions, which are expressed explicitly as rules in adaptive hypermedia, result naturally from a "conventional" de-

sign process in which we only require a clear separation of concerns from requirements.

Additionally, and as shown in Section 5 the approach also works in already built applications by just using the same kind of analysis and a set of simple and non-intrusive tools. In this way, we can provide a useful kind of adaptation (considering the tasks and topics in which the user is now working) which is focused to the specific application, without collecting much information about the user, as we only need his actual navigation path.

Our approach to improve navigation in existing applications by using scripting techniques makes an extensive use of the ideas in [11]. While the Modding Interface does not give a solution to CSN issues by itself, it is an outstanding idea to simplify and modularize the scripting process, making it feasible and helping to produce a more solid and modular client-based solution.

Inter-application communication has been a subject of research by different companies for some years now; Passport .Net [35] and Google Applications Suite [21] provide authentication ways across different applications and combine service functionalities. Unfortunately these technologies have not become standard yet and therefore navigation support is usually restricted to the boundaries of the corresponding owner (Microsoft and Google in this case). Our experiments with CSN in an inter-application way show that it is possible to have customized improvements without much scripting effort, when a base Modding Interface, or a support for a family is already provided.

7 Concluding Remarks and Further work

In this paper we have presented a novel approach for improving navigability and access to information in Web applications. By clearly understanding the different concerns which are "packed" in a Web page, we can improve the user experience enriching those pages with information, operations and links pertaining to the actual user concern.

We have shown how this approach can be used when building new applications profiting from the identification of application concerns during the development process. In that sense we have shown that a good software engineering practice (separation of concerns) can be used not only to improve modularity and maintainability, but also to improve usability when these concerns are wisely used during navigation design. We have also shown how we used the same basic ideas to improve existing applications and particularly to enrich those applications using a client-side scripting strategy. Considering that a brand new generation of Web applications (the so called Web 2.0 applications) has emerged, there are many opportunities for improving navigation between these applications. In this sense, we have illustrated our approach and showed that it is possible to enrich inter-application navigation in a way which is oblivious to the existing software. Given that this kind of "intervention" may be troublesome when the application evolves, we have also explained how to use existing techniques (The Modding Interface concept) to modularize the interaction between scripts and the intervened DOM; particularly we have shown how to use an abstraction mechanism to make scripts more stable and even generic. We also discussed how to go

further in this strategy when dealing with families of applications (e.g. social software like Facebook or LinkedIn).

We have tested our extensions with users and obtained valuable feedback to improve the characteristics of the concern-sensitive extensions.

This area is relatively new and naturally there is a good deal of work to be done. We are actively working in different areas. From the modeling point of view, we are extending our approach to the interface realm; this implies analyzing how the navigational concern concept reflects on the abstract interface and how interface modeling languages can be seamlessly improved for this by following the same principles of composition transparency we presented in [19].

First we are improving our comprehension on good ways to use concern information to improve navigability. In order to do that we are studying popular applications to find recurrent navigability problems to later derive enrichment patterns specializing those in Section 2.3. From a more concrete point of view we are developing concrete plug-ins to experiment with different kinds of Web 2.0 applications and have a more extensive user experiences base. Considering that the development of these plug-ins is generally complex, we are improving our tools to ease the enrichment process. These tools support the developer task by providing a visual interface and thus avoiding the burden of having to deal directly with low level DOM structures. We are also studying the problem of ontology alignment to deal with inter-application enrichments in a more systematic way, particularly for applications families.

References

1. Adaptive Hypermedia Reference Library. http://wwwis.win.tue.nl/ah/publications.html
2. Amazon. http://www.amazon.com/
3. Barnes and Noble. http://www.barnesandnoble.com/
4. Baumeister H., Knapp A., Koch N. and Zhang G.. Modelling Adaptivity with Aspects. In David Lowe and Martin Gaedke, editors, Proc. 5th Int. Conf. Web Engineering (ICWE'05), LNCS 3579, pages 406-416. Springer, Berlin, 2005.
5. Bäumer D., Riehle D., Siberski W., and Wulf M., The Role Object Pattern. Proceedings of Plop,1997, USA. Available at: http://hillside.net/plop/plop97/Proceedings/riehle.pdf.
6. Busch M. and Koch N. . MagicUWE - A CASE Tool Plugin for Modeling Web Applications. In Proc. 9th Int. Conf. Web Engineering (ICWE'09), LNCS, volume 5648, pages 505-508. Springer, Berlin, 2009.
7. Ceri, P., Fraternali, P., Bongio, A., Web Modeling Language (WebML), A Modeling Language for Designing Web Sites. Computer Networks and ISDN Systems, 33(1-6), 2000, 137-157.
8. Clarke, S., Baniassad, E. Aspect-Oriented Analysis and Design. The Theme Approach. Addison-Wesley, 2005.
9. Ceri S., Daniel F., Facca F. M. and Matera M.. Model-Driven Engineering of Active Context-Awareness. World Wide Web (WWW), Spring, 2007, 387-413
10. Clemens, P., Northrop, L. Software Product Lines: Practices and Patterns. Addison Wesley, 2001.
11. Diaz, O., Arellano, C., Iturrioz, J. Layman tuning of websites: facing change resilience. in Proceeding of WWW2008 Conference, (Beijing, 2008).
12. Django. http://www.djangoproject.com/
13. Facebook. http://www.facebook.com/

14. Filman, R., Elrad, T., Clarke, S., Aksit, M. Aspect Oriented Software Development. Addison Wesley, 2004.
15. Flickr. http://www.flickr.com/
16. Fons, J., Pelechano, V.,Pastor O.,Valderas P., Torres V. Applying OOWS model-driven Approach for developing Web Applications. The Internet movie database case study. in Web Engineering: Modelling and Implementing Web Applications, Springer, 2008.
17. Gamma, E., Richard H., Johnson, R., Vlissides, J. Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
18. Ginzburg J., Distante D., Rossi G., Urbieta M.. Oblivious Integration of Volatile Functionality in Web Application Interfaces. Journal of Web Engineering, Vol. 8, No 1, pp 25-47, 2009.
19. Ginzburg, J., Rossi, G., Urbieta, M., Distante, D.: Transparent interface composition in Web Applications. in Proceedings of ICWE2007 Conference, (Como, 2007). Springer, 2007, 152-166.
20. Gmail Greasemonkey API. http://code.google.com/p/gmail-greasemonkey/wiki/GmailGreasemonkey10API
21. Google applications suite. http://www.google.com/apps/
22. Güell, N., Schwabe, D., Vilain, P. Modeling Interactions and Navigation in Web Applications. in Proceedings of ER Workshops 2000 Conference, (Utah, 2000), Springer, 115-127.
23. Hi5. http://hi5.com/
24. Horchani, M., Nanard, J., Nanard, M. Les Hypermédias comme Paradigme d'Interfaces Adaptatives. in Les hypermédias Journal, I. Saleh (ed), Hermès, 2004, 119-146.
25. Houben, G.J., van der Sluijs, K., Barna P., Broekstra, J., Casteleyn, S., Fiala, Z., Frasincar, F. Hera. In Web Engineering: Modelling and Implementing Web Applications, Springer, 2008, 263-301.
26. Knol. http://knol.google.com/
27. Koch, N., Knapp, A., Zhang, G., Baumeister, H. UML-Based Web Engineering. in Web Engineering: Modelling and Implementing Web Applications. Springer, 2008, 157-191.
28. Kristensen, B.B., Osterbye, K.: Roles, Conceptual Abstraction Theory and Practical Language Issues. Journal of Theory and Practice of Object Systems, 2(3), 1996, 143-160.
29. LinkedIn. http://www.linkedin.com/
30. MonkeyGrease. http://code.google.com/p/monkeygrease/
31. MySpace. http://www.myspace.com/
32. Nanard, J., Rossi, G., Nanard, M., Gordillo, S., Perez, L. Concern-Sensitive Navigation: Improving Navigation in Web Software through Separation of Concerns. in Proceedings of CAiSE'08 Conference, (Montpellier, 2008), Springer, 420–434.
33 Niederhausen, M., van der Sluijs, K., Hidders, J., Leonardi, E., Houben, G.J., Meißner, K. Harnessing the Power of Semantics-Based, Aspect-Oriented Adaptation for AMACONT. in Proceedings of ICWE 2009 Conference (San Sebastián, 2009), Springer, 2009, 106-120.
34. Parnas. http://dret.net/biblio/reference/par72
35. Passport .NET. http://www.passport.net/
36. Riehle D. and Gross T.: Role Model Based Framework Design and Integration. OOPSLA 1998: 117-133
37. Rossi, G., Nanard, J., Nanard, M., Koch, N. Engineering Web Applications with Roles. Journal of Web Engineering, 6 (1), 2007, 19-48.
38. Rossi, G., Schwabe, D.: Modeling and Implementing Web Applications with OOHDM. in Web Engineering: Modelling and Implementing Web Applications, Springer, 2008.
39. Schmid, H., Rossi, G.: Modeling and Designing Processes in E-Commerce Applications. in IEEE Internet Computing Journal, 8 (1), 2004, 19-27.
40. Sowa, J. Conceptual Structures: Information Processing in Mind and Machine. Addison Wesley, 1984.

41. Struts. http://struts.apache.org/
42. Sutton, S. and Rouvellou, I.: Modeling of Software Concerns in Cosmos. in Proceedings of ACM Conference, (Enschede, 2002), ACM Press, 2002, 127-133.
43. The guided tour of the Roman Open Air Museum. http://www.villa-rustica.de/tour/toure.html
44. Wikipedia. http://www.wikipedia.org/
45. Youtube. http://www.youtube.com/
46. YouTube API. http://code.google.com/intl/es-zAR/apis/youtube/getting_started.html#player_apis