A Flexible Architecture for Client-Side Adaptation

Sergio Firmenich[1,2], Gustavo Rossi[1,2] , Silvia Gordillo[1,3], and Marco Winckler[4]

[1] LIFIA, Facultad de Informática,
[2] Universidad Nacional de La Plata and Conicet Argentina
[3] CiCPBA
{sergio.firmenich,gordillo,gustavo}@lifia.info.unlp.edu.ar
[4] IRIT, Université Paul Sabatier, France
winckler@irit.fr

Abstract. Currently the Web allows users to perform complex tasks which involve different Web applications. Anyway they still have to face these tasks in a handcrafted way. Although it is possible to build service-based software, such as mashups, to combine data and information from different providers, many times this approach has limitations. In this paper we present an approach for Client-Side Adaptation aimed to support complex concern-sensitive and task-based adaptations with user-collected data. Our approach improves user experience by supporting user tasks among several Web applications.

1 Research Context

The evolution of the web has given more and more capabilities to users. From only allowing them browsing contents, at present users not only modify and add new ones but that they choose the way in which the content is presented. Nowadays, users perform several tasks on the Web in general using more than one application. As a consequence they have to perform some extra work when they change the Web application in use and want to continue their task in it. With each switch the work context is generally lost. It means that both what the user was doing and the relevant information which was used is lost when he navigates. As part of a solution to this problem, users can combine contents and services from different resources in order to generate new applications like mash-ups. Another current trend is to adapt Web sites on the client-side, which implies modifying the original contents of Web applications by adding, removing or modifying elements into the Web sites DOMs. This phenomenon of Client-Side Adaptation (CSA) has given new capabilities to users, who now can adapt Web sites correspondingly with their preferences or requirements even those which are not contemplated by Web applications developers.

CSA is a promising trend since at the Client-Side all information about the user activity is available even outside the boundaries of a single Web application. However, while CSA provides a new opportunity to integrate information from several Web sites this power does not seem to be fully exploited. Our first step was to apply concern-sensitive navigation (CSN) at the Client-Side [2]. CSN is a conceptual tool to improve the user experience by taking into account his concern while he navigates the Web. Suppose the user is navigating from A to B; then when he arrives to application B, the application is adapted with information or functionalities

A. Harth and N. Koch (Eds.): ICWE 2011 Workshops, LNCS 7059, pp. 327–331, 2011.

according with the user concern in A, but if the user would be navigated from C the adaptation would be different. The main idea of CSN is to adapt the current node considering the contents of previous ones in the navigation. CSN can occur in two ways. When both pages A and B belong to the same Web application we say that it is *intra-application* CSN. On other hand, CSN is *inter-application* when target and source nodes belong to different applications.

In this work we show a flexible architecture to support CSN and other kinds of adaptations. We have developed a set of tools for supporting users in developing adaptations, which can be shared such as in a crowdsourced development structure.

This paper is organized as follows: in section 2 we introduce the objectives of our research; in section 3 we present a software structure to support all our approach. Finally, section 4 presents conclusions and further works.

2 Research Objectives

The research proposal of this position paper tries to answer the following research question: *How can adaptation of existing Web applications be directed in order to improve the user's experience by empowering him with mechanism to fulfil volatile adaptation requirements while he navigates different Web applications to accomplish his tasks?*

Different objectives were identified in order to success deal with the main research question. The first one is to clearly *identify which information is available at the Client-Side.* In comparison with typical approaches for adaptation, CSA gives us new possibilities. For our approach two kinds of information are mainly relevant:

- Navigational history: on Client-Side all navigational history is available instead of being restricted to a single Web application. This information is really important to realize inter-application CSN when the user switches from an application to another (either by following a link or typing a new URL).
- Information Used: knowing which other Web applications the user has used or is using is really powerful to make adaptations, but it would be even more if we could also use information from these applications.

A second objective is to *analyze and design mechanisms to allow users make adaptations under demand in order to fulfil volatile requirements to adapt Web sites in a concern-sensitive or task-based way.*

Differently to other, more static, Web augmentation [1] approaches (e.g. those using GreaseMonkey [3]), we want to empower adaptation by allowing users to develop complex components that will be used in the adaptations (see section 3). Basically the main idea is to accomplish a deeper CSA approach by fully exploiting all Client-Side capabilities instead of limiting ourselves to attach restricted JavaScript code when the Web pages are loaded as when using GreaseMonkey. Then, we have a third objective, which is *to clearly determine what kind of software artefacts can be developed by users with programming skills.* As a consequence of the previously described objectives we have the followings new ones. First we want *to ease the development process of the artefacts found by giving users the guidelines to develop a particular kind of artefact.* On the other hand we have to *design a robust but flexible architecture to orchestrate the – crowdsourced – developed artefacts.*

Figure 1 shows an adaptation to depict the potential of our approach. Here a user is using IMDB. The image 1 shows the page of the movie "Black Swan" and from it the user navigates (with the link showed in image 2) to Amazon.com in order to buy the "Black Swan" DVD. When Amazon.com is loaded a menu is added (image 3) to offer searches about other movies which the user had visited before in IMDB (image 4).

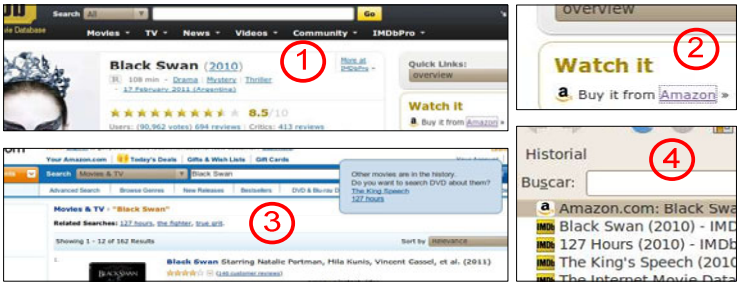

Fig. 1. Example of Client-Side adaptation based on navigational history

In the following section we propose a structure where the whole approach is modularized into components with specific responsibilities.

3 Solution Approach

Current solutions for CSA provide mechanisms to modify Web pages DOM in order change content or functionalities. Our goal is to provide a better support to the current user task or concern, but at the end we always need to change the pages' DOM to materialize the adaptation. We have to perform two different activities: 1) detect the current user concern/task, 2) change the Web pages DOM accordingly with the needs of the concern or task detected. Note that two different concerns could need similar DOM changes, for instance to remove some DOM elements, only changing the target DOM elements. As a consequence it is reasonable to reuse those software pieces which manipulate DOMs. We call these components: *augmenters* since they respond to classical definition of Web augmentation. Augmenters perform generic adaptations such as automatic form filling or text highlighting. Meanwhile, those artefacts which analyze concerns and apply *augmenters* are called *scenarios* since they realize scenario of Web applications usage. By combining augmenters, *scenarios* support customized adaptations for specific domains such as trip planning, house rental, etc. For example a scenario can use the form filling augmenter when the user is navigating among several Web sites for booking flights and hotels. The same augmenter can be used to fill forms related to a product search in e-commerce Web sites, for example by taking the department (e.g. *electronics*) and the keyword (e.g. *iphone4*) used in *amazon.com* to complete the form automatically in *fnac.fr*. Scenarios can use data collected by users to execute augmenters with different arguments.

A third kind of artefacts in our approach are *components*. These are auxiliary tools used to perform adaptations. From the point of view of *scenarios*, *components* perform critical functionality such as accessing the user navigational history, getting relevant information, etc. On the other hand, *components* are useful to empower

augmenters by providing them more privileges that would not be available in simple JavaScript code, for example to get geo-location information. These three kinds of elements are developed and shared by users, then a crowdsourced development of three layers is achieved where: 1) *augmenters* are responsible of manipulating the DOMs, 2) *scenarios* are aware of the user activity and opportunely trigger some *augmenter* in order to adapt the Web pages, 3) *components* act as libraries used to get information or make operations allowing to reuse them by *augmenters* or *scenarios.*

Augmenters, Scenarios and Components are software artefacts that need to be coordinated. In the following section we introduce our framework for CSA which includes a set of tools which help to coordinate these user-developed artefacts.

3.1 A Framework for Client-Side Adaptation

Figure 2 shows the framework architecture based on the pyramid approach [4].

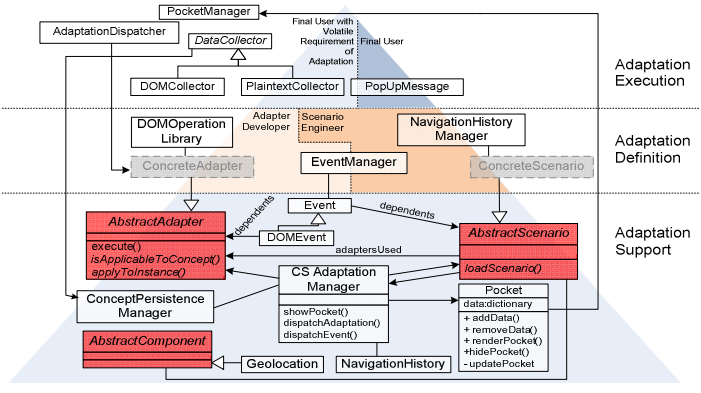

Fig. 2. Framework structure

Top levels are more abstract while lower ones are more detailed. At the top layer, final users can collect relevant information for their current task or concern by using *DataCollector* tools. When they navigate to other sites they are able to execute augmenters using this information; in this way they can satisfy volatile adaptation requirements (not foreseen by developers). At the middle layer, end users with programming skills can extend the framework by developing augmenters or scenarios as classes inheriting of *AbstractAdapter* and *AbstractScenario*, two outstanding framework hot-spots. The bottom layer shows the framework design in a more detailed view; a third hot-spot, *AbstractComponent* abstracts concrete components; for example we developed a component which offers geo-location information.

By conciseness reasons we outline only the main components:

- **Adaptation Support Layer**
  - ClientSideAdaptationManager: is the Framework's core, whose functions are to coordinate others elements and to serve as communicator with the browser.
  - ConceptPersistenceManager: is responsible for saving and restoring user data into the local files system.

- AbstractAdapter and AbstractScenario: are abstract classes from which concrete augmenters and scenarios, correspondingly, developed by users must inherit.
- AbstractCommponent: is an abstract class used for extending the framework by developing components to support new capabilities (e.g. geolocation).

- **Adaptation Definition Layer**
  - EventManager: is the responsible of adding and removing listeners (Adaptation Definition Layer) of events from the lower layer.
  - ConcreteAdapter and ConcreteScenarios: are scripts developed by users with programming skills. These classes are shown in Figure 3 in order to highlight their place in the hierarchy. Some concrete augmenters as HighlightAdapter, WikiLinkConverter, CopyIntoInputAdapter are included in our framework.
- **Adaptation Execution Layer**
  - DataCollector: is the tool to allow users collecting information while navigating. So far, two concrete DataCollectors have been implemented: one for selecting plaintext information, and another to handle DOM elements.
  - PocketManager: is our tool to allow users to move information among sites.
  - AdaptationDispatcher: is the responsible of executing an adaptation under user demand. It is useful to accomplish volatile requirements of adaptation.

4 Conclusions and Future Work

In this work we have presented a novel approach for CSA which proposes a flexible structure to support concern-sensitive and task-based adaptations. The framework allows three kinds of extensions to support and execute adaptations on Client-Side. The framework can be extended and/or used both by end-users or developers (e.g. by developing JavaScript code). In comparison with usual the CSA tools, we provide a flexible mechanism to integrate information while users navigate the web, instead of "just" providing tools to statically adapt Web sites.

We are working in two main directions to improve the approach. The first one is to improve the development process using the framework. The second one is to raise the abstraction level for developers by creating a domain specific language that will simplify the specification of both augmenters and scenarios; this will let users without JavaScript knowledge to develop adaptations easily.

References

1. Bouvin, N.O.: Unifying Strategies for Web Augmentation. In: Proc. of the 10th ACM Conference on Hypertext and Hypermedia (1999)
2. Firmenich, S., Rossi, G., Urbieta, M., Gordillo, S., Challiol, C., Nanard, J., Nanard, M., Araujo, J.: Engineering Concern-Sensitive Navigation Structures. Concepts, tools and examples, JWE 2010, pp. 157–185 (2010)
3. Greasemonkey, `http://www.greasespot.net/` (last visit on April 11, 2011)
4. Meusel, M., Czarnecki, K., Köpf, W.: A Model for Structuring User Documentation of Object-Oriented Frameworks Using Patterns and Hypertext. In: Aksit, M., Auletta, V. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 496–510. Springer, Heidelberg (1997)