Flexible distribution of existing Web interfaces: an architecture involving developers and end-users

Sergio Firmenich[1], Gabriela Bosetti[1], Gustavo Rossi[1], Marco Winckler[2]

[1] LIFIA, Facultad de Informática, Universidad Nacional de La Plata and CONICET
{sergio.firmenich, gabriela.bosetti,
gustavo}@lifia.info.unlp.edu.ar

[2] ICS-IRIT, University of Toulouse 3, France
winckler@irit.fr

Abstract. This paper presents a novel approach towards the opportunistic and lightweight distribution of existent Web User Interfaces. We describe an architecture that allows end-users to collect UI objects into a distributed UI-Component-oriented PIM, accessible from different user's devices. Once in the PIM, different DUI-based behaviours (that may be triggered by the user) are added to the collected UI components as PIM's objects plug-ins. We present an overview of the architecture; some default DUI-based behaviours are introduced and illustrated through examples. Besides, we show how the architecture supports the development of new behaviours.

Keywords: Client-Side Adaptation, DUI, End-User Development

1 Introduction

The distribution of user interfaces (DUI) has been a growing trend in the last ten years. Beyond the understanding about how DUI applications can improve user experience [12], several works for engineering DUI Web applications have emerged [10, 13]. Even more, approaches for more specific cross-device interaction support has been defined, such as the use of Kinect [11]. However, it is still hard to find popular Web sites or applications supporting DUI.

When applications do not offer features that users may need, it has been shown by experience that the crowd react trying to satisfy these needs by itself. This is a very common practice in Web Browsing Augmentation, i.e. using tools (deployed such as Web Browser Extensions) to augment a Web application capabilities. To cite one example, we can say that simple solutions for cross-device interaction such as "Slides – Presentation Remote"[1] has more than sixty thousand users, just offering a remote control for presentations in well-known Web applications (Google Drive, SlideShare,

[1] https://chrome.google.com/webstore/detail/slides-presentation-remot/mhfdnafbhfglkcjgkgoopjoadaopcomi

Prezi, etc.). Examples like this clearly show that while ad-hoc developers may create this kind of experience, there are also users expecting them.

We have been researching how to apply lessons learned in the field of client-side Web Augmentation [3] to the field of DUI. Web Augmentation comprises those approaches that aim to adapt content and functionality of existing (and third-party) Web applications, once these are loaded on the client. This technique has been used successfully in different domains such as, mash-ups [17], end-user driven Web tuning [4], etc. Web Browser Augmentation is a perfect target for some DUI applications such as supporting opportunistic remote control, layout distribution, UI migration, etc.

In this paper, we propose to involve not only developers but also users in the process of user interface distribution. The main idea is to provide developers with an easy way to implement DUI layers over existing Web pages, while end users may decide how to apply such layers on their preferred Web sites. This is achieved by managing a UI-Component-oriented PIM, where users may collect the target UI components to be manipulated in the applications defined by developers. The paper presents the overall architecture and the supporting tools through some case studies.

The paper is structured as follows. First, we present the related works in Section 2. Section 3 introduces our approach and the main components. The supporting tools are illustrated via case studies in Section 4. Finally, we discuss about our contribution in Section 5, together with our future work.

2 Background and related works

Several approaches have emerged for adapting or integrating existing Web contents since the early years of Web Augmentation [2] (WA) and Mash-Ups applications [5]. Diverse communities of users, both developers and amateur end users with technical know-how, set up the basis and contributed in the creation of new repositories of augmentation artefacts, which improve the Web with extra features that original Web applications did not contemplate. Such is the case of Greasyfork, or Mozilla addons. Some End User Development [9] (EUD) works arose later in the field, to empower users to solve their particular needs by themselves. Concerning DUI in conjunction with the WA and EUD fields, and as pointed out in [14], we can appreciate that it is not easy to provide DUI for existing and third party Web content, although some approaches have tackled isolated specific dimensions, specifically involving end-users. User-Driven DUI was previously defined in [15]; however, although this approach does not consider third-party existing Web sites as potential targets, other approaches do. For instance, [7] lets users to annotate some parts of exiting Web UI in order to migrate components under user demand. The main idea is to allow users to access a Web application on a desktop systems and then, to continue interacting with it from a mobile device migrating those relevant portions of the UI. Other similar approaches using proxies are Proxywork [16] and WebSplitter [8]. Other works address flexible interface migration [1].

In this paper we present an architecture to apply DUI layers over existing Web content. But in contrast with existing work, we do not provide an end-user tool for

performing specific DUI applications, but an architecture to enable developers from Web Augmentation communities to define easily new kind of DUI applications taking advantage of the underlying distributed UI-Component-oriented PIM. However, since we share the philosophy behind empowering end-users with specific tools, our approach lets them to instantiate a DUI application in their preferred Web sites.

Our approach may support several kinds of DUI dimensions, such as light interface migration, remote control, distributed layout, etc. We have designed a client-site visual tool that lets users to specify how to distribute the UI of any Web site.

3 Distributing Web UI Objects

The main idea behind our approach is to provide end users with a specialized PIM with extra WA and DUI capabilities. Instead of collecting and structuring personal data such as traditional PIMs, we propose a UI-component-oriented PIM. This allows users to collect arbitrary DOM elements from existing Web sites. A DOM element represents a particular component of the user interface. These UI components are collected into the PIM and materialized as UIObjects. Our approach rests on the idea that a UIObject may be used by the end-user for triggering different behaviours for supporting DUI. We call this kind of actions DUI behaviour. For instance, the user may activate the DUI behaviour "Close on other devices…" for an UIObject. This action would hide that UIObject in any other device registered in the platform by the user, and it would be functional only in the current device. Furthermore, although users may use some predefined "DUI behaviours", developers may create new ones. We call them behaviours because these are performed individually for a UIObjects; however, if several UIObjects are collected for a specific Web site, and different behaviours are executed with each of them, then a more complex DUI experience (involved combinations of objects) could be defined. In this way, at the end, the addition of several DUI behaviours may abstract a specific kind of distributed use of a Web site. For instance, a simple distributed layout could be supported if, after collecting several UIObjects, the user executes the "Close on other views…" behaviour for different UIObjects running in different devices/monitors. Meanwhile, other combined use of these behaviours may be oriented to control the UI displayed on a desktop computer from a mobile device. Since it is not our purpose to foreseen every possible DUI behaviour we propose instead a flexible architecture based on this UI-object PIM. The architecture was designed with two premises in mind: 1) users should be able to collect UIObjects and use a specific DUI behaviour with the objects collected into the PIM, and 2) developers should be able to create new kinds of DUI behaviours that may be added to the collected UIObjects. If the user does not select a behaviour, the object behaviour is set by default according to the type of DOM element (we detail this later).

The overall idea is that by triggering some behaviour, the user is enriching the existing Web site with DUI features. In order to do that, users are allowed to annotate some existing DOM elements as UIObjects that will be collected into the PIM, and then presented in another view. In this section, we introduce the components and con-

cepts of the approach and some aspects about the architecture. Then we show how end-users may create and use UIObjects.

Underlying Architecture

As it is shown in Figure 1, our approach mostly works at client-side. The UIObject-PIM is a Web Browser extension that, once installed in different devices, allow users to share a unique space of information among them. In order to maintain the UI-Objects synchronized, we provide a synchronization server (accessible from our Web Browser extension) that allows an instance (UIObject) of a particular UI component to be synchronized, by acting as a mediator among all the contexts where such instance is running. The approach is no constrained to a centralized server, the browser extension may be configured to work with any specific deployment of that server.

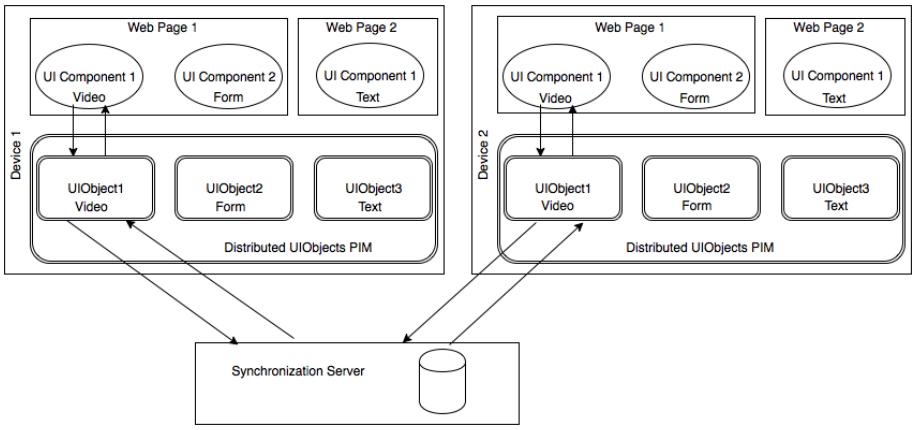

Fig. 1. Architecture schema of the UIObject-PIM platform

In this paper, we focus on the client-side features of this architecture, in which we can find three main components:

- UI Component: in this way, we refer to the native UI components or widgets that compose a particular Web site. For instance, a UI Component could be a form, a video, a part of the DOM tree, etc. The main idea is that end-users may select under demand those UI Components of their interest. Through this selection, from an UI Component is generated a UIObject.
- UI Objects: these are representations of UI Components, enriched with some behaviour that they do not have associated by default (in the original Web page). These UI Objects live in the UIObject-PIM.
- UI-Component-oriented PIM: this a PIM oriented to maintain references to those UIObjects created by users. This PIM requires authentication and an authenticated user may retrieve his preferred UIObjects from all his devices. The UIObjects collected into the PIM are not just a façade of UI Components, but managers of UIComponents with specific DUI-based behaviour added to its corresponding UI-Object. This behaviour is materialized as operations that are executed for a particu-

lar UIObject, and it is executable directly by the user. For instance, if the user wants to render a UI Component only in one of his devices, then he must run the "Show only in…" operations, which will ask the user which is the target device and then it will carry on the desired UI effect.

In this way, the PIM maintain the current state of the DUI model based on the operations (DUI-based behaviour) that were executed for the collected objects. In this way, the UI Component is like a view (in the same way as in the MVC pattern).

UIObjets and DUI-based behaviours in detail

UIObjects are JavaScript objects abstracting UIComponents. During the abstraction process, the user may choose a target UIComponent. Figure 2 shows in (1) a user enabling the DOM selection and, in (2), a DOM element being highlighted for selection with a proper context menu enabling the harvesting. Once selected, he can configure some of its aspects. For instance, give it a name and associate it some properties, as shown in (3). One of the most relevant properties is the UIComponentStereotype. This property allows users to choose among different kinds of UI widgets, such as images, text, forms, videos, etc. Once defined, all the UIObjects are available in the PIM, as the one in (4), so the user can interact with the offered behaviour.

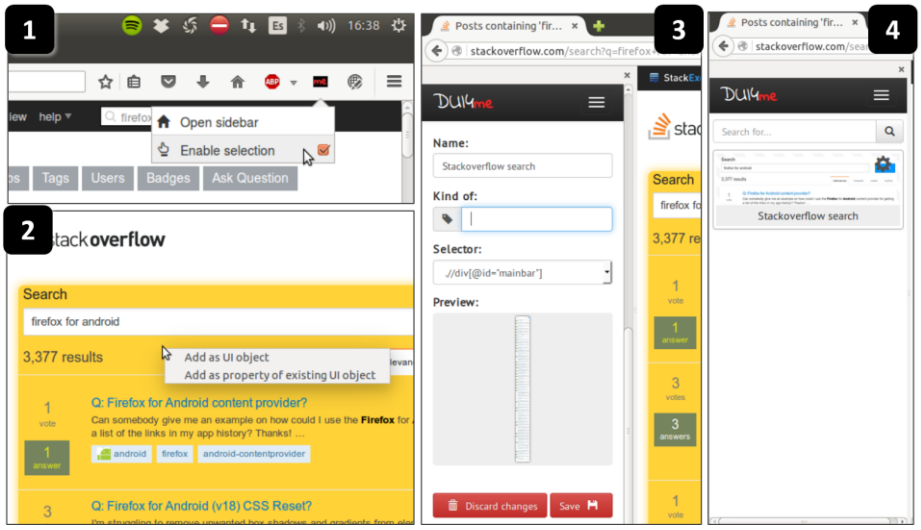

Fig. 2. Defining a UIObject

When a UIObject is collected into the PIM, the matching DUI-based behaviours (those preferred by the user) are attached to it under the basis of the decorator pattern [6]. There are two kinds of DUI-based behaviours. First, we can have stereotype-agnostic behaviours that may be attached to every UIObject since their goals are compatible with all kind of UIComponents. For instance, showing a particular UIObject only in a particular device. On the other hand, stereotype-specific behaviours

are attached only to those UIObject that were collected as a specific UI stereotype, such as a YouTube Video. In this case, more specific operations such as *"Play video on…"* can be defined.

It is worth mentioning that the set of decorators applied to the UIObject can be configured by the end-user, by clicking the gear icon shown in Figure 1; after doing this a context menu opens and the user has to select «manage» and finally «applied behaviour». The list of decorators is presented, and a series of parameters can be set.

Although we provide some basic behaviours, new ones may be defined by developers. To develop a new behaviour requires programming with client-side Web technologies (HTML, JavaScript, CSS), specially using a JavaScript API that allow developers to access, manage and add behaviour to UIObjects collected into PIM, considering that the communication mechanism among devices is solved transparently.

New behaviours may address new kind of DUI applications (for instance a particular kind of distributed layout), but may also perform specific operations wrapped in messages that the objects into the PIM may respond to. For instance, a developer may define a new behaviour for controlling a YouTube player by defining messages such as «*stop*» and «*play*». Then, if a user collects this object into the PIM and apply the decorator, then these messages could be sent from any device transparently. In these cases, developers are benefited with the underlying synchronization mechanism among PIMs state from different devices.

4 Case Study and Prototype

Although the approach lets developers to create new DUI behaviours, we have defined some predefined ones. So far we have identified: 1) opportunistic and volatile interface migration, 2) multi-monitor layout, 3) distributed layout, 4) messaged-based remote control, 5) navigation control and 6) remote UI Effect. For the sake of space, we are covering just one of them, in order to depict the feasibility of the approach.

At the time of this publication, we have partially implemented two prototype client-side tools supporting the approach. The first one is a Firefox 44 extension for desktop edition, the second one is a Firefox for Android extension, for version 42.0a1. At server-side, we implemented a minimal functionality to synchronize changes of the UIObjects both tools manage.

Consider a scenario where Máximo is living abroad for some months and he wants to write about every detail of his experiences in his blog. There are moments when he can write an entire entry from his computer, but he must migrate the task to his phone often, because his job requires constant mobility. His mother tongue is Spanish, but he is writing the blog in English. Sometimes, he have some doubts about language expressions, so he checks it out in linguee.com. It is desirable for him to have a mean for distributing some elements of the Linguee and Blogger Web sites.

Figure 3 shows our tool sidebar, with some UIObjects created from both sites; a search box and a results box from Linguee, and then a Blogger's entry main options, and a text area. So, when Máximo has to leave his house, he may open such elements in his mobile, the «XT1021» listed device, as shown in the left-bottom area of Figure

3. Then, from his device, the browser extension will be notified to open the same URL and just open the defined UIObjects the user has defined for such URL.

While the user changes the document, such modifications are notified to our server module, who asks every registered listener to update the view of the proper element. As we can see, the site functionality is not altered by our adaptation: the blog is still functional (you can see the automatic saving being executed in the right screen of Figure 3). The advantages of accessing the blog through our platform is that: 1) Blogger does not provide a mobile version, so the same interface elements are presented in small devices; 2) While Blogger offers the ability to save changes in the entry, it does not offer the ability to keep the content up-to-date from two or more devices, and without our platform, it is required for every device to refresh the Web page every time they want to see the changes from another device.

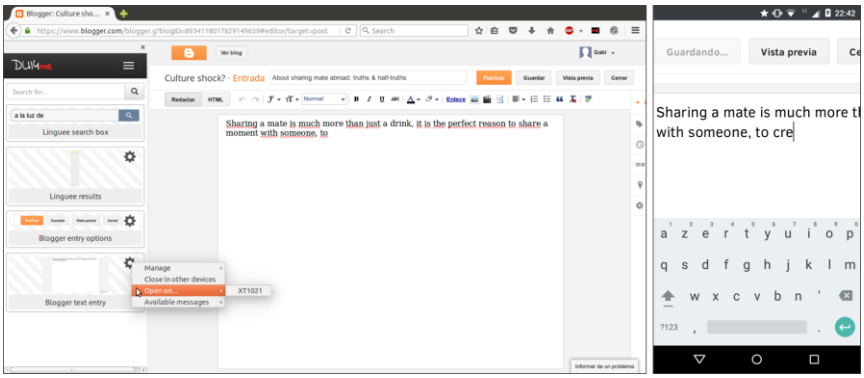

Fig. 3. Distributing UI components from a Web application to mobile

5 Conclusions and future works

In this paper, we have presented an architecture for supporting the development of Web Browser Augmentation artefacts for the field of distributed user-interfaces. Browser augmentation oriented to DUI was also propose in previous works [7] [16]. We share the philosophy behind them, but we think that providing developers with an architecture that abstracts the more complicated technical issues underlying to DUI applications may help to create several kinds of new experiences by developing new DUI-based behaviours. Besides that, the same architecture contemplates a mechanism to allow end-users to instantiate those behaviours opportunistically in any Web site.

Currently, we are completing the implementation of the support system, which is partially operative. Once it is finished, we plan to carry on an evaluation with end-users in order to measure how useful and easy is for them to use our tools. We also plan to investigate new possible behaviours and study how this approach may raise new possibilities in the context of Web Augmentation, mash-ups and collaboration.

References

1. Bandelloni, R., & Paternò, F. (2004, January). Flexible interface migration. In Proceedings of the 9th international conference on Intelligent user interfaces (pp. 148-155). ACM.
2. Bouvin, N. O. (1999, February). Unifying strategies for Web augmentation. In Proceedings of the tenth ACM Conference on Hypertext and hypermedia: returning to our diverse roots: returning to our diverse roots (pp. 91-100). ACM.
3. Díaz, O., & Arellano, C. (2015). The augmented Web: Rationales, opportunities, and challenges on browser-side transcoding. ACM Transactions on the Web (TWEB), 9(2), 8.
4. Díaz, O., Arellano, C., Aldalur, I., Medina, H., & Firmenich, S. (2014). End-User Browser-Side Modification of Web Pages. In Web Information Systems Engineering–WISE 2014 (pp. 293-307). Springer International Publishing.
5. Ennals, R. J., & Garofalakis, M. N. (2007, June). MashMaker: mashups for the masses. In Proceedings of the 2007 ACM SIGMOD international conference on Management of data (pp. 1116-1118). ACM.
6. Gamma, E., Helm, R., Johnson, R. & Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Pearson Education, 1994.
7. Ghiani, G., Paternò, F., & Santoro, C. (2010, September). On-demand cross-device interface components migration. In Proceedings of the 12th international conference on Human computer interaction with mobile devices and services (pp. 299-308). ACM.
8. Han, R., Perret, V., & Naghshineh, M. (2000, December). WebSplitter: a unified XML framework for multi-device collaborative Web browsing. In Proceedings of the 2000 ACM conference on Computer supported cooperative work (pp. 221-230). ACM.
9. Lieberman, H., Paternò, F., Klann, M., & Wulf, V. (2006). End-user development: An emerging paradigm (pp. 1-8). Springer Netherlands.
10. Melchior, J., Vanderdonckt, J., & Van Roy, P. (2011, June). A model-based approach for distributed user interfaces. In Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems (pp. 11-20). ACM.
11. Nebeling, M., Teunissen, E., Husmann, M., & Norrie, M. C. (2014, June). XDKinect: development framework for cross-device interaction using kinect. In Proc. of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems (pp. 65-74). ACM.
12. Santosa, S., & Wigdor, D. (2013, September). A field study of multi-device workflows in distributed workspaces. In Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing (pp. 63-72). ACM.
13. Schreiner, M., Rädle, R., Jetter, H. C., & Reiterer, H. (2015, April). Connichiwa: A Framework for Cross-Device Web Applications. In Proc. of the 33rd ACM Conference Extended Abstracts on Human Factors in Computing Systems (pp. 2163-2168). ACM.
14. Vanderdonckt, J. (2010). Distributed user interfaces: how to distribute user interface elements across users, platforms, and environments. Proc. of XI Interacción, 20.
15. Vandervelpen, C., Vanderhulst, G., Luyten, K., & Coninx, K. (2005). Light-Weight distributed web interfaces: preparing the web for heterogeneous environments. In Web Engineering (pp. 197-202). Springer Berlin Heidelberg.
16. Villanueva, P. G., Tesoriero, R., & Gallud, J. A. (2013, June). Proxywork: Distributing User Interface Components of Web Applications. In DUI@ EICS (pp. 58-61).
17. Wong, J., & Hong, J. I. (2007, April). Making mashups with marmite: towards end-user programming for the web. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 1435-1444). ACM.