# User Interface Adaptation Using Web Augmentation Techniques: Towards a Negotiated Approach

Diego Firmenich[1,2], Sergio Firmenich[1], Gustavo Rossi[1], Marco Winckler[3], Damiano Distante[4]

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

[2] DIT, Fac. de Ingeniería, Universidad Nacional de la Patagonia San Juan Bosco, Argentina
dfirmenich@tw.unp.edu.ar

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

[4] Unitelma Sapienza University, Italy
Damiano.distante@unitelma.it

**Abstract.** The use of Web augmentation techniques has an impact on tasks of owners and developers of Web sites, developers of scripts and end users. Because the Web site can be modified by external scripts, Web site's owners might lose control about how Web site contents are delivered. To prevent this, they might be tempted to modify the DOM structure of Web pages thus making harder to execute external scripts. However, communities of Web augmentation scripters are increasing since end-users still have needs not yet covered by original Web sites. In this paper we analyze the trade-offs of the introduction of Web augmentation scripts. In order to mitigate some negative effects, such as the loss of control, we propose an approach based on negotiation and coordination between actors involved in the process. We present a set of tools to facilitate the integration of scripts and to foster the dissemination of Web augmentation scripts for the benefit of all actors involved.

**Keywords:** Web augmentation, client-side adaptation, script developers.

## 1 Introduction

Web augmentation techniques have been proposed as a way for extending Web sites features without affecting the server-side code [2]. Most of the popular Web augmentation tools extend the Web browser functionalities via plugins that can run client-side scripts to manipulate the structure of Web pages. The potential of these techniques for adapting existing Web sites is huge and this can be easily illustrated by some advanced applications [10][11]. Web augmentation techniques are used to adapt sites according to users' needs that have not been originally taken into account during the design of the Web site. The flexibility provided by Web augmentation techniques motivates individual and communities of coders to develop scripts. For example,

YouTube center[1], which adds several new functionalities (e.g. download and repeat videos) for improving the user experience, has been proved very popular with more than 15K downloads. The very existence of communities might also allow end-users to provide direct feedback to developers via requests for new augmenters instead of asking the Web site's owners to change the original site [8]. Moreover, some existing tools such as WebMakeUp [7] claim to support end-user development of Web augmenters. Thus, Web augmentation might compete with other existing techniques for adapting Web sites. When compared with closed adaptation techniques broadly-used by large Web applications for supporting personalization [3], Web augmenters allow users to go beyond of the adaptation features predefined by the Web site's owner. Some Web sites provide APIs that can be used to build extension-based adaptation of contents. For developers, APIs extensibility usually implies to follow specific guidelines and constraints. Moreover, there is no guarantee that the API will provide all the sought adaptation mechanisms. Besides, without prior commitment of the participants involved in the process, the development of Web augmenters can also be frustrating since Web augmenters might stop working when owners decide to change the Web site design [5]. As for the Web site owners, neglecting users' need for adaptation and the creative potential of community of coders, might make them less competitive.

In this paper we analyze the trades-off of Web augmentation approaches and we claim that benefits can be shared among actors involved in the process. An analysis of different strategies for Web site adaptations and the actors involved in the process is presented in section 2. In section 3 we propose a negotiated approach for Web augmentation adaptation. The aim is to delegate to the crowd of users the specification of the changes they are looking for, delegate to coders (users with programming skills) the implementation of the augmenters and finally let the Web site's owner integrate augmenters into their Web sites. Such an approach is duly supported by a platform which is described in section 4. In section 5 we propose an assessment based on cost estimation. In section 6 we discuss the contribution at the light of existing work and lastly in section 7 we present conclusions and future work.

## 2 Actors, strategies and trade-offs for adapting Web interfaces

The adaptation of user interfaces (UI) requires the definition of what are the goals of the adaption, what is adapted, what events trigger the adaptation, and which programming techniques are used to perform the adaptation [3]. In the context of this work, the goal of adaptation is to modify any aspect of the UI at client-side (either rendering and/or behavior). For that purpose adaptations might change the way users perform tasks (e.g. replacing scroll navigation between the top/bottom parts of a Web page by adding navigation buttons), the contents in display (e.g. enriching the page with information obtained from other sources), and/or the page's structure and layout. These adaptations should be triggered by DOM events in the client-side. As for the techniques, the main focus of this study is on Web augmentation.

[1] YouTubeCenter. Available at: *https://greasyfork.org/es/scripts/943-youtube-center*, last access: 12/2/2015

The development, deployment and use of Web adaptation techniques affect many actors, including the *Web site owner*, the *coders of scripts* and the *end-users*. The term *Web site owner* designates here the development team that has full control of the original Web site. In opposition, *coders of scripts* refer to developers who implement augmenters to support adaptations on the client-side. *End-users* refer to the user populations that consume Web site contents (being adapted or not to their profile). We identified four strategies for adapting Web applications involving these actors:

- *Closed adaptation* refers to adaptation techniques that are embedded as part of the original Web site and totally under the control of *Web site owner*. This kind of adaptation might encompass adaptations processed on the server-side and/or on the client-side. In any case, developers don't have any constraint for accessing to the code source of the Web site. Adaptations can be built upon users' characteristics that have been obtained by explicitly collecting users' profile (via Web forms) or implicitly (by tracking the behavior of ordinary visitors) [13]. In closed adaptation, the Web site is modified as the result of a direct relationship between the *Web site owners* and the *end-users*. End-users contribute with information that can be used to personalize the adaptation and they only have access to adaptations that have been predefined by the Web site owners. Typically, recommendation systems, collaborative filtering systems, or hybrid systems [15] belong to this category.
- *Extension-based customization* is obtained with the help of dedicated APIs which allow external developers to adapt the Web site. This kind of adaptation strategy is well known in applications such as Google Drive[2] or Facebook[3]. By using APIs, *external coders* can create new forms of adaptations that have not been considered yet by *Web site owners*. Overall, adaptations require a triangulation between what *Web site owners* make adaptable via an API, what *coders* can do with such API and the *end-users'* expectations. Advanced programming skills, deep knowledge about the original Web site and the functions provided by the API are required to create adaptations. Thus, *coders* are somewhat dependent of the availability of API delivered by the original Web site. Besides that, end-users need to be supported with some tool for browsing and installing the available extensions.
- *Web augmentation* is a term used to address techniques that allow the adaptation of existing third-party Web applications in the client-site in such a way that no prior authorization from *Web site owners* is required [3]. Web augmentation techniques can be fine-tuned to work on a specific Web site but they can also be generic enough to work in any kind of Web site. The most common solutions for implementing *Web augmentation* techniques is Web browser extensions (i.e. plugins) that once installed modify the Web sites visited by the users. End-users may take advantage of a large set of scripts uploaded by their creators to public repositories[4]. Most of the popular augmenters are implemented using JavaScript, which means that coders need to have advance skills in imperative programming. The relation-

[2] The Drive Platform. Available at: *https://developers.google.com/drive/* Last access: last access: 12/2/2015
[3] The Facebook API. Available at: *https://developers.facebook.com/* Last access: last access: 12/2/2015
[4] Repositories list: *http://wiki.greasespot.net/User_Script_Hosting*. Last accessed: 2/4/2015

ship between actors is simplified by excluding *Web site owners* from the adaptation process.

- *End-User Web augmentation* is a concept built upon *Web augmentation techniques* for empowering end-users with tools allowing them to program their own scripts [4][7]. The underlying idea is that users with little programming skill can be guided through a set of visual tools that hide the complexity of the code used to perform the adaptations on a Web site. This kind of techniques assume that end-users can work by their own, so that they do not longer depend on *coders* and/or *Web site owners* for having their Web sites personalized. For example, in [8] authors propose an environment where users can modify on the fly Web pages as a means to express requirements and/or to personalize the interaction with the site.

Adaptation strategies define different types of relationship between actors. Table 1 shows how *Web site owners'* and *end-users* are involved in a *closed adaptation* strategy. This is the most traditional approach for Web adaptation. Users often have to provide personal information (explicitly via a user account, or implicitly by user footprints whilst navigating the Web site). Closed adaptation, by definition, excludes the possibility of collaboration with external coders so all the costs of adaptation are supported by Web site owners. But site owners have at least full control on the adaptations provided to end-users.

**Table 1.** Trade-offs on Closed Adaptation

| Web site Owner | Coder | End-user |
|---|---|---|
| - Implementation of adaptations models is expensive<br>+ User can provide personal information for portraying a user profile to tune adaptations<br>+ Full control of adaptation mechanisms proposed to users<br>- Adaptation mechanism are dependent of a Web site | - | + Users do not need to install anything on the browser<br>- Costs limit the daptation to large and specialized Web sites<br>+ Can support personalization via recommendation and collaborative filtering systems<br>- Users must have create an account and a profile to get the benefits of adaptations<br>- Requiring user profile/feedback raise privacy issues<br>+ Adaptations can be tune to the usual tasks with the site<br>- Might not be enough to support users requirements<br>- Users have to learn how adaptation works in every site |

**Table 2.** Trade-offs on Extension-based Adaptation

| Web site Owner | External Coder | End-user |
|---|---|---|
| - Implementation of APIs has direct costs, including for training and advertisement<br>+ Control of functions that are made available through the API<br>+ Code provided by external coders might contribute to the popularity of the owners' Web site<br>- Still requires programming on the top of APIs to implement the required adaptations<br>- The owner don't have control of API-based adaptations | + Often free of costs for coders<br>+ Support and documentation might be free of charges<br>+ Coders can build they applications on the top of existing Web sites, which is often cheaper than starting from scratch<br>- Coders creativity can be limited to functions available in the API<br>- Might not be enough to support adaptations envisaged by coders<br>+ Coders can contribute to create new adaptations | + Users do not need to install anything on the browser<br>+/- There is no guarantee that applications using APIs are free of charges for the end-users<br>- Might not be enough to support the adaptations required by users<br>+ Let users to customize the application by installing extensions |

By exposing an API to the community *Web site owners* allows the collaboration with an external community of *coders*, see Table 2. Deploying an API implies significant costs for the Web site owners but might also contribute to the popularity of the Web site, for the benefits of *external codes* and the *Web site owners* alike. To ensure protection on the backend, adaptability through APIs is often limited to a few functions which do not necessary grant access to information about the user profile. If one hand such strategy allows some level of control, it might also limit creativity of *external coders*. For the users, *extended adaptation* will only work in a few Web sites and do not necessary cover all users' needs for personalization.

In the case of *Web augmentation* strategies, it is the *Web site owner* who is excluded from the process, such as Table 3 and 4 show. The relationship occurs only between *end-users* who become the clients of *coders* who develop augmenters that operate on client-side without the supervision and control of the *owners'* of the Web. In such cases, *Web site owners* are excluded from the adaptation process and have no control in how the contents are delivered to end-users. *Web site owners* might be tempted to regularly modify the DOM structure of Web pages thus making harder to execute external scripts, which is a hard blow for *external coders* and *end-users*. Adaptations are often volatile and should be reapplied every time users return to the Web site.

**Table 3.** Trade-offs on Web Augmentation Adaptation

| Web site Owner | External Coder | End-user |
|---|---|---|
| - | + Coders creativity is not limited by APIs<br>- Adaptations might stop working if the DOM of Web page is updated<br>+ By tracking downloads of extensions proposed to the users coders can assess popularity and infer user needs<br>+ The independence of APIs allows the development of augmenters focused on specific tasks that might be generic and thus be applied in many Web sites<br>+ There is no limitation about what aspects of the client-side application the coder may adapt on the Web site<br>- Different augmenters developed by different coders might spoil the alterations between them | + Users have diverse alternatives for adapting Web sites<br>- Might not be enough to support all the adaptations required by users<br>- Require users to install extensions<br>+ Users can reuse adaptation tools in a seamlessly way whilst navigating the Web<br>- It is hard to take into account the user profile in generic augmenters, so that users might lose the benefits of recommendation systems or collaborative filtering systems<br>- Adaptation mechanism provided natively may be spoiled<br>- Since augmenters are not verified, some of them may be malicious |

**Table 4.** Trade-offs on User-Driven Web Augmentation

| Web site Owner | External Coder | End-user |
|---|---|---|
| - | - | + Users are empowered with tools to perform the adaptations they want |
| | | - End-users must develop programming skills for using EUP tools |
| | | - Adaptations can be limited by users knowledge and skills, as well as the functions delivered in EUP environments |

## 3 Towards a negotiated Web adaptation approach

All existing adaptation strategies remarkably fail to provide seamless collaboration between actors involved. For that, we present a negotiated approach to Web adaptation which relies on three basic principles: first, all actors must find advantages in the collaboration; secondly, actors must collaborate; last but not least, tooling is essential to incentivise actors collaboration. We next present a view at glance of the approach and a detailed description of the distribution of tasks among the actors and how the execution of individual tasks can contribute to advantages for all.

### 3.1 The approach in a nutshell

Fig 1. illustrates some advantages actors can find in this kind of relationship. For example, *Web site* owners can stablish a trustful relationship with *coders* that guarantee that augmenters are not malicious. The negotiation between *Web site owners* and *coders* also benefit *end-users* who, by extension, can trust that third-party augmenters have been checked by owners of the Web site they trust. By keeping *end-users* in the loop benefits both the *Web site owners* who can continue to collect information about users and even share such as information with *coders* for improving their augmenters according with user needs. The negotiated approach also implies that a kind of commitment can be reached and for that actors must collaborate. Close collaboration and commitment often demand the implementation of coordination and communication tasks [14], which require additional resources (in terms of time and cognitive effort to maintain relationships running). To prevent that additional coordination and communication tasks come to plunge the advantages of such collaboration, the negotiated approach proposes that actors can work independently (as much as possible) and only perform the tasks for which they might foresee a direct advantage. To support such as a light-tight collaboration, we rely on a distribution of tasks among the participants and the existence of appropriate tool support as presented in Fig 2.

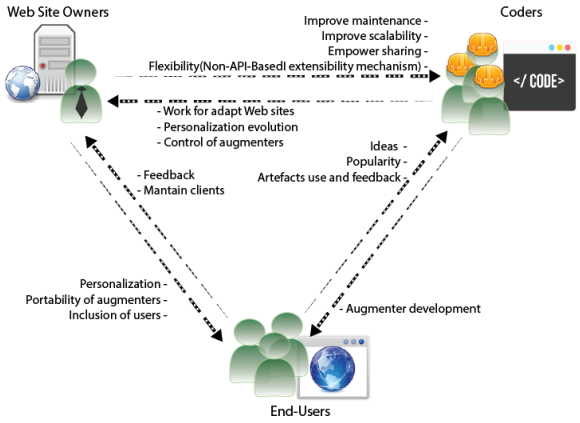



**Fig. 1.** Relationships between actors in a negotiated adaptation approach.

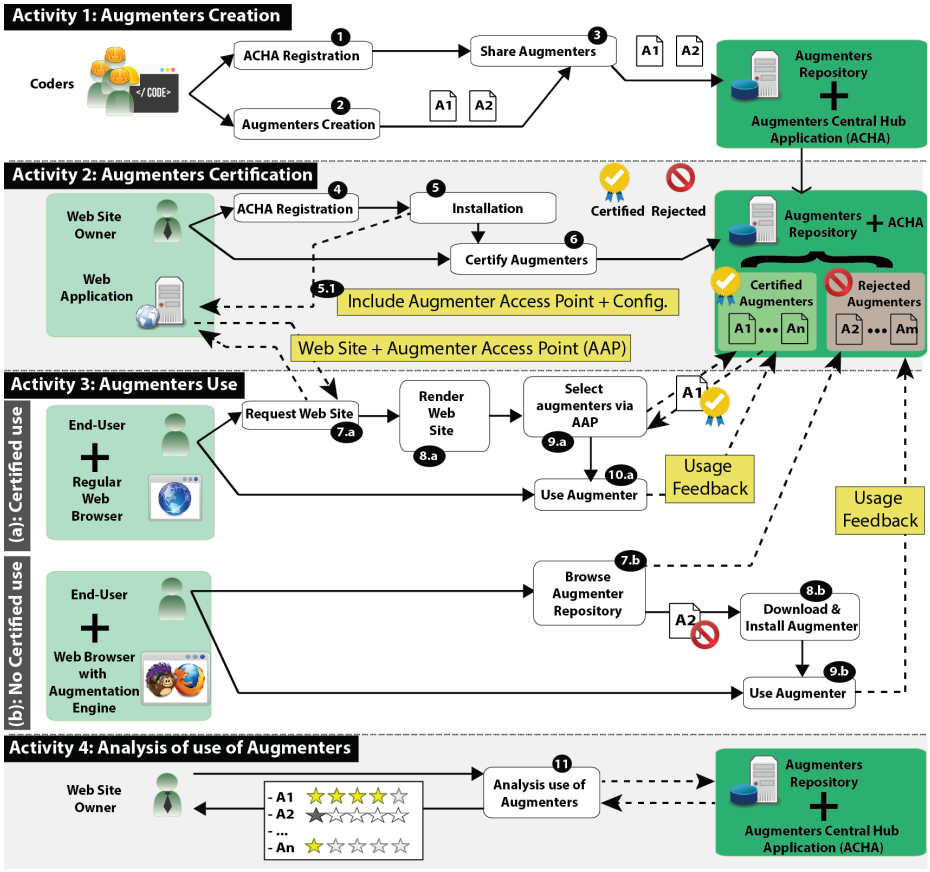



**Fig. 2.** Tasks allocation in a negotiated adaptation based on a Web augmentation approach.

### 3.2 Analysis of actors' tasks

Hereafter we analyze implications for individuals of tasks imposed by the approaches shown in Fig 2. Notice that these tasks cannot be performed without appropriate tool support. For that a set of tool have been developed, including:

- *Augmenter repository:* is a Web site that contains the augmenters;
- *Augmenters Central Hub Application (ACHA):* is the front-end application that allows the management (search, inclusion, etc.) of augmenters into the repository.
- *Augmenters Access Point (AAP):* is a client-side component embedded on the Web sites registered in ACHA thus providing direct access to certified augmenters.

A full description of these tools is provided in section 4. Nonetheless, we make explicit reference to these tools whilst describing the tasks the different actors have to perform in a negotiated approach.

**Augmenters Creation: Coders.**
The main task of *coders* is to create and share augmenters. Once duly registered at ACHA (task 1 in Fig 2), coders can create (2) and share (3) augmenters at the Augmenter Repository. Scripts proposed by codes might include generic adaptations that work for diverse Web sites (ex. replace phone numbers in Web pages with a shortcut to Skype), adaptations that exploit user profile (ex. by using user history of navigation, the augmenter might propose links to recent searches) or advanced adaptations that allows user customize user interface (ex. letting the user to rearrange the layout).

**Augmenters Certification: Web Site Owners.**
As for *Web Site owners*, they have three main tasks, as follows:

- *Registration*: in first place, owners need to register the ownership of a particular web domain (task 4 in Fig 2), for instance dblp.org. They also must provide a security file that is use to authenticate its Web site.
- *Installation*: to make augmenters available through their Web sites, owners have to include in their HTML responses the AAP component (task 5). This component is the responsible of allowing users to select augmenters without installing Web browser plug-ins. Tools included in AAP component have a look & feel by default but it is possible for a *Web site owner* to change it to make it to fit in Web site's design (see tasks *Config AAP* and *adapting look & feel* (5.1) in Fig 2).
- *Augmenters Certification*: once the registration process is finished, *Web site owner* can inspect the augmenters in the repository and certify those who he thinks useful for adapting their Web site (task 6).

**Augmenters Use: End-Users.**
The negotiated approach gives to end-users a major role as they ultimately have full control of adaptations that are going to be performed on the Web site. As we shall see in Fig 2, users have many duties with respect to the selection of certified (activity A1) and/or non-certified augmenters (activity A2):

- *Use of certified augmenters (a)*: when users visit a Web site (task 7.a) for which there are certified augmenters, the Web is rendered in the client (task 8.a) embedding the AAP tool. By using the AAP tool (task 9.a) *end-users* can select the desired augmenters which are then downloaded and executed transparently in the client-side (task 10.a). At the same pace, the ACHA record in the repository the information that a user has downloaded and used a given augmenter.
- *Use of non-certified augmenters (b)*: *end-users* use augmenters that do not have been certified by *Web site owners*. Non-certified augmenters do not automatically appear through the AAP when visiting a Web site. However, by using ACHA, users may browse non-certified augmenters (task 7.b). If the augmenter is relevant, the user may download it (task 8.b), and use with some external Web Augmentation engine such as GreaseMonkey (task 9.b). Note that this activity is not necessarily carried out by all the Web application users, but by those that are aware of the existence of the mechanisms for adapting third-party existing applications.

**Analysis of use of Augmenters: Web Site Owners.**
As show by task 11 in Fig 2, Web Site Owners can obtain feedback of use of the existing augmenters. Information about users using non-certified augmenter (task 9.b) of a Web site becomes part of the knowledge base of ACHA. This information is available for Web site owners who, thereupon, can decide to investigate (or not) why users are using such augmenters and what are they need for adapting the Web site.

# 4 An platform for Web augmentation dissemination

In this section we present further details about the tool support that we have developed to demonstrate the feasibility of our negotiated approach. Section 4.1 presents a few underlying requirements that we have identified as essential for automating (as much as possible) user tasks. This follows with the presentation of the set of tools that have been developed to support the approach and concrete example of tool usage.

## 4.1 Underlying requirements

These components were defined to address several aspects we believe to be really important for a negotiated adaptation approach:

- *Easy to install*: tools installation should be as simple as possible for the *Web application owners*. With this in mind the only actions required to the owner are the registration of the corresponding Web application in ACHA and also to add a JavaScript library on the Web pages (which contains the AAP among others). In the development of current applications supported by Web frameworks, it usually would mean to add a line of code in the main template of the application.
- *Customizability* of the look & feel for augmenters: besides to be easy to install, we allow Web application developers to define specific styles and behavior to the end-user tool (Augmenter Access Point tool) in order to make it compatible with the look & feel of the application.
- *Plug & Play*: it is essential for *end-users* to be able to select, activate and deactivate augmenters. Users must feel in control of the usage of adaptation but should guide them in the process.
- *Compatible and extensible*: the negotiated approach and the corresponding tool set should be compatible with existing augmenters in the community. In this way, our current implementation is compatible with existing user scripts, which are probably the most popular kind of artefacts. For that, our tool set must also provide an *Augmenter Engine Emulators* to make possible to execute any kind of scripts featuring augmenters.
- *Independence of Augmenter Repository*: the external repository shown in Fig 2 is proposed as a public standalone Web application. However, if *Web site owners* don't want to consume augmenters from the public repository, they instantiate a version of both the Augmenter Repository and ACHA in a private Web server accessible to a small community of *coders*.

### 4.2 Set of tools supporting the approach

We have developed a bipartite system composed by a client-side library and a server-side Web application. On a dedicate Web server-side, the tools include:

- *Augmenters Repository and Augmenter Central Hub Application*: Augmenter Repository centralizes all the augmenters created by coders. The Augmenter Central Hub Application (ACHA) allows to manage the repository according to each role, i.e., that ACHA exposes different views and functionality for repository accordingly with the responsibilities of coders, owners and end-users.

At the client-side, there is the Augmenter Access Point component (AAP), a JavaScript library that encompasses three subcomponents cooperating with each other:

- *End-User Augmenter Selection Tool*: this tool aims at helping the selection of augmenters by end-users. This tool takes into account the current context, i.e. which Web page of the application is loaded; this is because an augmenter not necessary works for the whole application but just one or a set of nodes.
- *Augmenter Injector*: this tool is used for downloading and executing certified augmenters. The tool also record the user selection, so that the next time a user visits the same Web page, the corresponding augmenter is automatically executed.
- *Augmenter Engine Emulator*: this component allows the emulation of diverse APIs such as GreaseMonkey. It was implemented to make our approach compatible with any possible Web augmentation artefacts.

### 4.3 Illustration of tools in a case study

The case study presented in this section is based on the Web site dblp.org. For the sake of illustration, in these examples we adopt the perspective of actors whilst presenting tools and the corresponding tasks. Moreover, all the examples below make reference to adaptions of the original page shown in Fig 3.

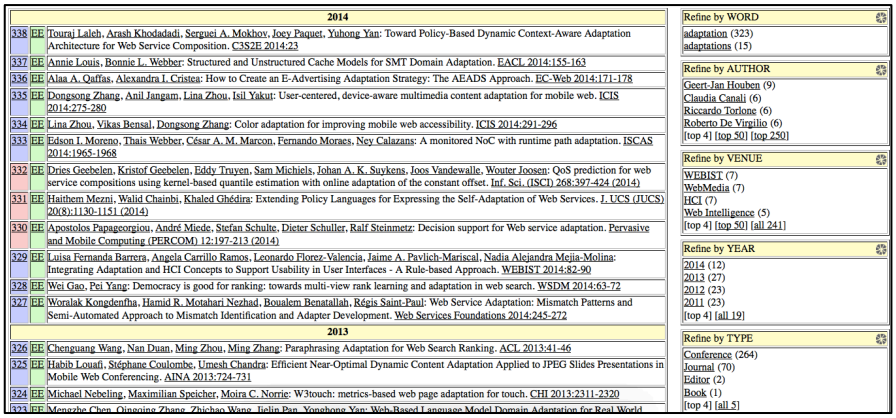
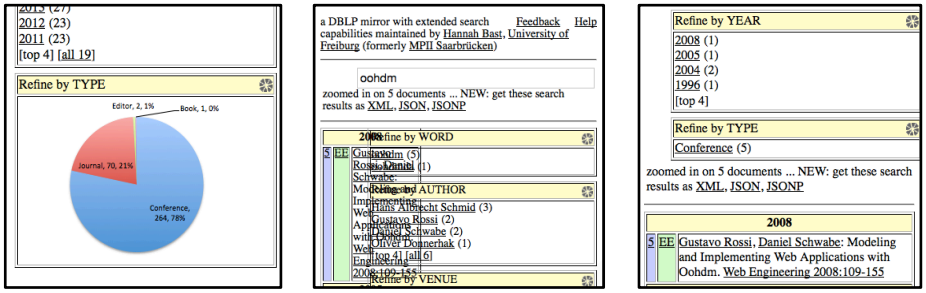



**Fig. 3.** Original search page of the DBLP web site.

**Coders.**
We assume that a coder has developed two augmenters that are aimed at improving the user experience of DBLP's users by implementing adapting features that are not yet available there such shown by Fig 4.




*a) Pie chart augmenter featuring publications per type at DBLP*

*b) original DBLP page*

*c) adapted page using responsiveDBLP augmenter*

**Fig. 4.** Example of augmenters created by coders and waiting to be shared with the community.

The first is an augmenter that is able to parse a DBLP Web page, extract information about publications and create a pie chart graph that can be injected into the Web site for depicting publication as shown in Fig 4.a. The second augmenter is aimed at modify the layout of Web page responsive to screen size. Fig 4.b shows the original Web page when visualized in a small screen and how visualization problems are fixed by the augmenter called *responsiveDBLP* Fig 4.c.

The augmenters illustrated by Fig 4 are ready to be used but the owners of the DBLP Web site did not have certified them yet. In order to get a certification and improve the visibility of these augmenters, we assume that the coder decides to share them via a public instantiation of both the Augmenter Repository and the ACHA, hosted at *UserRequirements*.org. For that, *coders* must create a user account on ACHA, define a user story describing the adaptations provided by each augmenter and finally upload it using the Web form.

**Web site owners.**
Let's assume that the owners are particularly interested by the augmenter *responsibleDBLP* shown Fig 4 mainly because such augmenters might save lots of work for making the DBLP responsive. For including the *responsiveDBLP* into the DBLP Web site, the owners have at first to register at *UserRequirements.org* which can be accomplished by following these steps:

- Create a user account in ACHA, hosted in *UserRequirements.org*
- Register themselves as owners of the domain *dblp.org*
  - Certificate ownership: in order to demonstrate that they are actually the owners of dblp.org, they must download from ACHA a file containing a security token for *dblp.org* and upload the security token file to the web application root
  - Log in in *UserRequirement* and validate domain. ACHA will check that the security token file is already in the owners' Web server.

With these steps, ACHA accepts the association between that user account and the specified domain *dblp.org*. Once affiliated, owners need to add the Augmenter Access Point component into their main HTML. This only implies to add one line of code for adding a JavaScript file, line 5 in the code shown in Fig 5.

```
1  <!DOCTYPE html>
2  <link rel=stylesheet type="text/css" href="http://www.dblp.org/autocomplete-php/autocomplete/logging.css">
3    <script type="text/javascript" src="http://www.dblp.org/autocomplete-php/autocomplete/autocomplete.js"></script>
4    ......
5    <script src="http://www.dblp.org/../AugmenterAccessPoint.js" onload="URM_init('augs.userrequirements.org');"></script>
6    ......
7  <html>
8  <head>
9    ......
10   <title>CompleteSearch DBLP</title>
11   ......
12 </head>
13 ......
```

**Fig. 5.** DBLP Web page (HTML) featuring the links binding it to the augmenters repository.

In order to certify augmenters, the *Web site owners* must look for augmenters at the Augmenter Repository suitable to work with the DBLP Web site. As shown by Fig 6, once the augmenter *responsiveDBLP* is found, the certification is done by selecting actions enable/disable options. It is also possible to download the augmenter for inspecting the code source and run it to see how it works. These tests are addressed mainly to check if augmenters are compatible with the current Web site DOM, however, owners may add further tests about the augmenter execution in order to prove if the adaptation is not spoiling relevant original content or functionality. Also from the ACHA, Web site owners can monitor user's feedback on this augmenter.


**Fig. 6.** Management of augmenters at the Web site repository: *UserRequirements.org*.

**End-users.**

When DBLP end-users visit the Web site they will be notified that certified augmenters exist by a green binding point at the up corner of the screen, as shown at the left side of Fig 7. The interaction between the end-user, the Web browser and our components is shown in Fig 8. When the user clicks on it, a menu is deployed showing augmenters available (this is the Augmenter Selection Tool). To active/deactivate an augmenter, end-users only need to click on the corresponding name in the list. Via this menu, end-users can also see the description of the augmenter left by the coders and further details about its popularity. Note that, as Fig 7 shows, the only augmenters available are those certified by the Web site owners in this case Responsive DBLP.

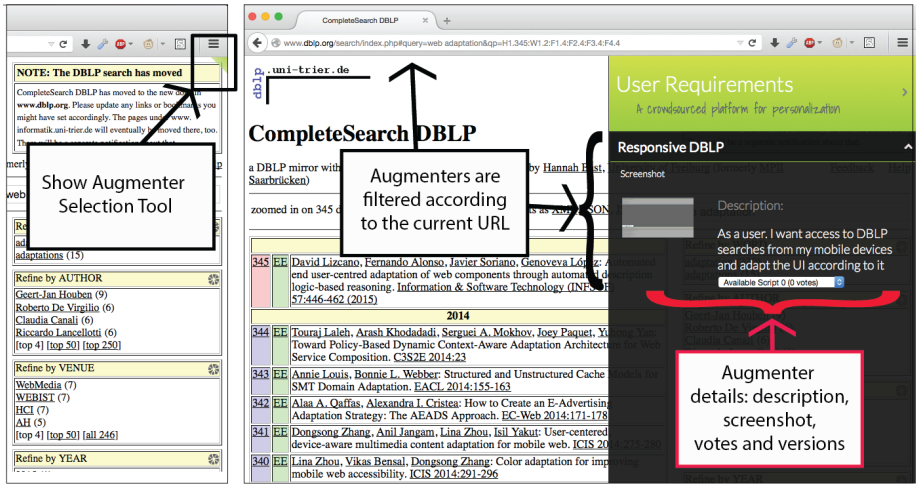



**Fig. 7.** Using augmenters at the DBLP web site.

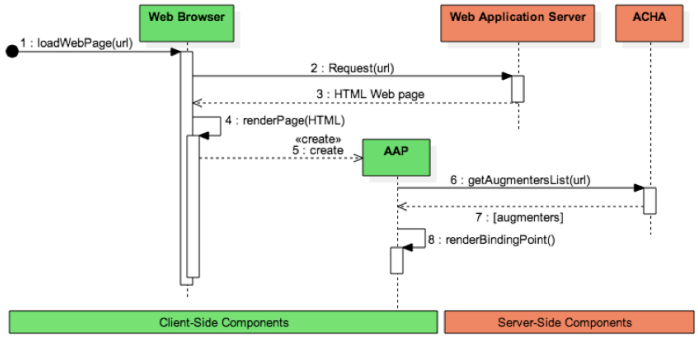



**Fig. 8.** Interaction when a Web site embedding AAP is rendered.

Fig 9 shows the selection of an augmenter from the AAP and how it connects with the Augmenter Repository to delegates the execution to the Augmenter Injector.

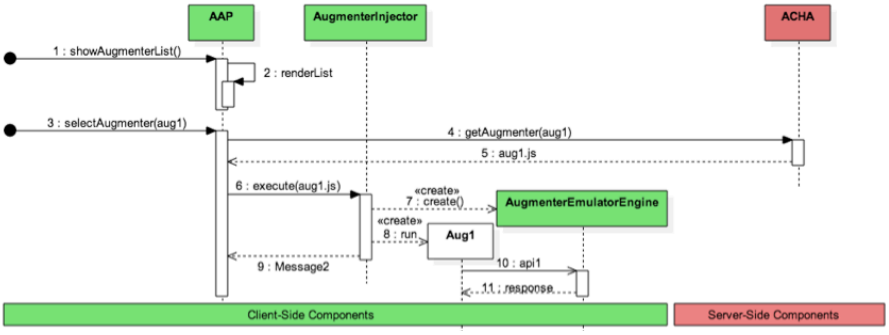



**Fig. 9.** Interaction when the end-user wants to enable an augmenter.

Although the non-certified augmenters do not appear in the Augmenter Selection Tool, if a user navigate the Augmenter Repository from ACHA, he may find also those rejected by the *Web site owner*, for instance "*Pie chart: publications per type*" among others. In these cases, the user may download and install it, in this case, with GreaseMonkey engine. Besides that, if available augmenters do not satisfy a particular user's need, he may ask to *coders* for new scripts by the addition of new user stories. For the sake of conciseness these functions are not described here but the interested reader can find further information at [8].

# 5 Preliminary assessment of tools

This section presents some preliminary assessment of the proposed platform.

## 5.1 Existent augmenters compatibility

To determine whether (or not) *existing augmenters in public repositories are fully functional and compatible with our platform* we have assessed the compatibility of 15 augmenters from public repositories, listed in Table 5. The augmenter selection was addressed to test different features:

- Generality: report if the augmenter works for any Web site or it is Web site-specific.
- Popularity: in terms of number of users (thousands of users *versus* a few known users).
- Programming effort: in terms of lines of code.
- API use: altogether, the augmenters selected use most of the API provided by the corresponding engines (User Script engines), in this way, we could show that Augmenter Engine Emulator is feasible to be built.

Since several of the augmenters are executed in very well-known Web sites that we are not able to manipulate at server-side, we have attached our platform to these applications via a bookmarklet, which is just a bookmark that executes JavaScript code when the user clicks on it. This JavaScript code is executed with the same privileges that native JavaScript code, then it is a sufficient prove of that augmenters may work from inside the application if the platform is also loaded. This also explains the case study presented in Section 3.1. The result was that the 100% of augmenters listed in Table 5 ran successfully with our platform. We have compared the result with the execution via Web Augmentation engines, and the augmentation effects were the same. Every feature of the augmenters (for instance, some personalization options that some of them support) also worked.

**Table 5.** List of augmenters assessed.

| Site | Augmenter | Description | Users | Lines of Code |
|---|---|---|---|---|
| Youtube.com | *Aug1*: Download YouTube Videos as MP4 | Adds a button to let users download YouTube videos. | 3.711 | 765 |
| Google.com | *Aug2*: Google Search Extra Buttons | Add buttons (last day, last week, PDF search etc.) to results of search page of Google | 150 | 104 |
| Imdb.com | *Aug3*: IMDB+ | Add external links to IMDb. Every feature can be enabled/disabled in settings. | 353 | 156 |
| * (any Web site) | *Aug4*: Mouseover Popup Image Viewer | Shows larger version of thumbnails. Also supports HTML5 video. | 8.580 | 1.207 |
| * (any Web site) | *Aug5*: Google Translator Tooltip Expanded | Translates the selected text into a tooltip automatically. | 493 | 1.227 |
| Imdb.com | *Aug6*: Search IMDb Item on Netflix | Places a "Search for this on Netflix" button on the main page of any TV show/movie page on IMDb | 141 | 34 |
| Trello.com | *Aug7*: Trello-minimize lists | Minimize width of lists with toggle button | 10 | 159 |
| * (any Web site) | *Aug8*: Fixed Scroller Anywhere | Scroll by fixed pages | 38 | 630 |
| Wikipedia.org | *Aug9*: Wikipedia Inline Article Viewer | Adds a hover event to internal article links on wikipedia pages, which open the article inline in a dhtml frame. | 3 | 512 |
| Geocaching.com | *Aug10*: Geocaching Map Enhancements | Adds Ordenance Survey maps and grid reference search to Geocaching.com, plus other enhancements. | 47.648 | 2.726 |
| Twitter.com | *Aug11*: Twitter Instagram Cards - Photo Viewer | Now that Instagram have pulled their twitter support, this script adds back inline instagram photos. | 361 | 46 |
| * (all Web site) | *Aug12*: Universal Syntax Highlighter | It highlights plain text source code URLs in several languages. Based on the SpiralX auto highlighter | 42 | 114 |
| Google.com | *Aug13*: Endless google | Load more results automatically and endlessly. | 4.732 | 142 |
| Google.com | *Aug14*: Google Cache comeback | Brings back links to cached pages in the Google search results | 15.208 | 248 |
| Youtube.com | *Aug15*: Youtube to mp3 | Convert youtube video to MP3 | 1.724 | 77 |

# 6 Related work

The present work has interconnections with many relevant research areas such as personalization and adaptation techniques of Web applications, development of frameworks for supporting client-side adaptation and transcoding, end-user programming and communities of developers. Since 1996, most of the papers related to adaptive hypermedia systems were focused on Web applications [3][16]. Most of the well-known methods for the design of Web applications have incorporated the design of

adaptation mechanisms. User profile modeling [13] has become also an important concern in adaptive Web applications, as well as the design of recommendation systems. However, the use of the Web not only is still increasing, which was the main factor mentioned by Brusilovsky [3], but also the way in which the Web is used has been mutating. There are several Web Augmentation communities around of existing repositories. Most important communities (in terms of size) are related to two kinds of artifacts, userscripts and userstyles. The formers are JavaScript-based augmenters such as those described in this paper. Currently, there are several userscripts repositories, altogether hosting more than 180 thousands of augmentation artifacts and several of these artifacts have been installed more than a million times. With all these existing repositories, it is clear that Web Augmentation is a current trend among the crowd of users. However, all these communities actually work without the intervention of Web site owners.

From the academy, in more recent years many works have investigated the potential of using End-User Programming techniques for allowing users to customize their applications [4][9]. Participation of the crowd of end-users is often presented as a suitable alternative for personalization, which often requires appropriate tool support and methodological approach for personalization [6]. Indeed, many works such as [7] focus on tool support for allowing end-users to tune Web sites. The results are promising but the impact in terms of number of users that can be reached by such as an approach is limited, given the skill level required to build such applications [12].

Some studies [17] have highlighted the importance of the involvement of communities of developers involved in the creation of scripts for adapting Web applications. Moreover, some authors [12] try to explain the role played by developers in the process. Other approaches tackle frequent design and implementation issues that appear when developing Web augmentation artefacts. For instance, some authors have studied how augmenter may be more resilient to DOM changes or even to improve the augmenters' reusability (i.e., usable in several Web sites) [5]. The same authors have proposed a security model in order to control what augmenters could do in a Web site [1], which is clearly utilizable in our approach. Nonetheless, very few works have investigated the relationship and possible interactions between other actors involved in the process, namely the owners of Web sites being adapted by external scripts.

## 7 Conclusions and future works

In this paper we have presented a negotiated approach that involves end-users, owners of Web sites and external communities of coders specialized in the development of Web augmentation scripts to perform client-side adaptation of Web sites. The underlying idea is to share the tasks required for Web-side adaptation among the actors involved in the process and that, for the benefits of all. By exposing the advantages that all actors might found in process, our negotiated approach presents a new perspective for the research in the areas of Web scripts development and Web site adaptation. Indeed, we claim in this paper that a deep analysis of tradeoffs for all actors is essential for deciding design options for implementing Web applications. In this re-

spect, the comparative analysis of advantages and drawbacks of adaptations approaches for each role is a contribution of this paper at its own right.

The negotiated approach is not a panacea and probably don't solve all adaptation problems. Indeed, it is not aimed at replacing adaptation mechanisms that work pretty well and already fulfill a purpose. Nonetheless, we do claim that a negotiated approach opens up a new perspective for the research in the area in particular with respect to the way we involve actors in the adaptation process, in terms of tools required to support a distributed architecture for client-side adaptation and about mechanisms for observing the evolution of end-user needs for adaptation of Web site contents. Indeed, this work allows starting to investigate many interesting research questions that for Web engineering, for example: In which extension end-users are able to comprise and collaborate with Web site owners and communities of coders? How user's needs that require adaptation of Web sites evolve overtime? How communities of coders can make a bigger impact on existing Web sites? How to prevent those communities of coders can damage the presentation of Web site contents? How to improve trustful relationship between users that have different interests in the adaptation of Web applications? How to ensure long term compatibility between Web sites and external scripts?

It is evident that for supporting the approach, and ultimately the underlying research questions, appropriate tool support is necessary. For that we have developed a set of full-fledge tools that are publicly available and we invite the interested readers to take a look at the Web site http://UserRequirements.org for further information. The tools are fully functional and can be used either by end-users, community of coders and Web site owners as it was dully illustrated in the present paper. Since the availability of such tools is very recent, we still don't have collected enough material to make any assertion about the usability and/or user experience of people using these tools. Due to the inner nature of the negotiated approach, studies in a long run are required to make the necessary observation of all users and confirm if our hypothesis (dressed here merely as an estimation effort) hold on.

As part of our future work, we have already started to advertise the platform around the community so that we can have a substantial number of users (in different roles) for supporting further analysis. We are also planning to pursue the study about compatibility between scripts and Web sites.

## References


1. Arellano, C., Díaz, O., Iturrioz, J. Crowdsourced Web Augmentation: A Security Model. In Proc. of WISE 2010. Springer LNCS 6488, pages 294-307.
2. Bouvin, N. O. Unifying Strategies for Web Augmentation. In: Proc. of the 10th ACM Conference on Hypertext and Hypermedia, 1999.
3. Brusilovsky, P. Adaptive Hypermedia. User Modeling and User-Adapted Interaction (UMUAI). Volume 11, Issue 1-2, pp. 87-110, 2001, Springer.
4. Díaz, O., Arellano, C., Aldalur, I., Medina, H., Firmenich, S. End-User Browser-Side Modification of Web Pages. In Proceedings of WISE (Thessalonoki, Greece), pp. 293-307, 2014.

5. Díaz, O., Arellano, C., Iturrioz, J. Interfaces for Scripting: Making Greasemonkey Scripts Resilient to Website Upgrades. ICWE 2010: 233-247.
6. Arellano, C., Díaz, O., Iturrioz, J. Opening Personalization to Partners: An Architecture of Participation for Websites. In Proceedings of the International Conference on Web Engineering (ICWE 2012), LNCS 7387, pp. 91–105, 2012.
7. Firmenich, S., Rossi, G., Winckler, M., Palanque, P. An approach for supporting distributed user interface orchestration over the Web. Int. J. Hum.-Comput. Stud. 72(1): 53-76 (2014).
8. Firmenich, D., Firmenich, S., Rivero, M., Antonelli, L. A Platform for Web Augmentation Requirements Specification. In proceedings of ICWE (Toulouse, France), pp. 1-20, 2014, Springer.
9. Firmenich, S., Rossi, G., Winckler, M. 2013. A domain specific language for orchestrating user tasks whilst navigation web sites. In Proceedings of the 13th International conference on Web Engineering (ICWE'13), Florian Daniel, Peter Dolog, and Qing Li (Eds.). Springer-Verlag, Berlin, Heidelberg, 224-232.
10. Garrido, A., Firmenich, S., Rossi, G., Grigera, J., Medina-Medina, N., Harari, I. Personalized Web Accessibility using Client-Side Refactoring. IEEE Internet Computing 17(4): 58-66 (2013).
11. Han, H., Tokuda, T. Towards flexible and lightweight integration of web applications by end-user programming. IJWIS 6(4): 359-373 (2010).
12. Jones, M. C., Churchill, E. F. 2009. Conversations in developer communities: a preliminary analysis of the yahoo! pipes community. In Proceedings of the fourth international conference on Communities and technologies (C&T '09). ACM, New York, NY, USA, 195-204.
13. Kobsa, A. Generic User Modeling Systems. In The Adaptive Web, Methods and Strategies of Web Personalization, pp. 136 – 154, 2007, Springer.
14. Malone, T. W., Crowston, K. The interdisciplinary study of coordination. ACM Comput. Surv., 26(1):87–119, 1994.
15. Adomavicius, G., Tuzhilin, A. Toward the Next Generation of Recommender Systems: Survey of the State-of-the-Art and Possible Extensions. Trans. Knowl. Data Eng. IEEE, 2005, pp. 734-749.
16. Rossi, G., Schwabe, D., Guimarães, R. Designing personalized web applications. In Proceedings of the 10th international conference on World Wide Web (WWW'01). ACM, New York, NY, USA, 275-284.
17. Stolee, K.T., Elbaum, S., Sarma, A. 2013. Discovering How End-User Programmers and Their Communities Use Public Repositories: A Study on Yahoo! Pipes. Information and Software Technology 55(7):1289–1303. Retrieved October 9, 2014 (*http://linkinghub.elsevier.com/retrieve/pii/S095058491200211X*).