A Platform for Web Augmentation Requirements Specification

Diego Firmenich[1,3], Sergio Firmenich[1,2], José Matías Rivero[1,2], and Leandro Antonelli[1]

[1] LIFIA, Facultad de Informática, Universidad Nacional de La Plata
[2] CONICET, Argentina
{sergio.firmenich,mrivero,lanto}@lifia.info.unlp.edu.ar
[3] Facultad de Ingeniería, Universidad Nacional de la Patagonia San Juan Bosco
dfirmenich@tw.unp.edu.ar

Abstract. Web augmentation has emerged as a technique for customizing Web applications beyond the personalization mechanisms natively included in them. This technique usually manipulates existing Web sites on the client-side via scripts (commonly referred as *userscripts*) that can change its presentation and behavior. Large communities have surfaced around this technique and two main roles have been established. On the one hand there are *userscripters*, users with programming skills who create new scripts and share them with the community. On the other hand, there are *users* who download and install in their own Web Browsers some of those scripts that satisfy their customization requirements, adding features that the applications do not support out-of-the-box. It means that Web augmentation requirements are not formally specified and they are decided according to particular *userscripters* needs. In this paper we propose CrowdMock, a platform for managing requirements and scripts. The platform allows *users* to perform two activities: (i) specify their own scripts requirements by augmenting Web sites with high-fidelity mockups and (ii) upload these *requirements* into an online repository. Then, the platform allows the whole community (*users* and *userscripters)* to collaborate improving the definition of the augmentation requirements and building a concrete script that implements them. Two main tools have been developed and evaluated in this context. A client-side plugin called MockPlug used for augmenting Web sites with UI prototype widgets and UserRequirements, a repository enabling sharing and managing the requirements.

1 Introduction

Nowadays, the Web is a really complex platform used for a great variety of goals. This advanced use of the Web is possible because of the evolution of its supporting technology and the continuous and massive demands from users that enforces and accelerates its rapid evolution. The Web has not evolved only in terms of technical possibilities but also in terms of supporting an increasing number and types of users too. The same crowd of users has evolved in parallel with the Web itself and the user expectations in terms of what the applications provide to them are very demanding nowadays. Even more, users have found ways to satisfy their own requirements when

S. Casteleyn, G. Rossi, and M. Winckler (Eds.): ICWE 2014, LNCS 8541, pp. 1–20, 2014.

they are not taken into account by Web applications developers. It is not casual that several Web augmentation [5] tools have gone emerging in this context. These tools pursue the goal of modifying existing Web sites with specific content or functionality that were not originally supported. The crowd of users continuously builds a lot of such as tools (distributed within the community as browsers plugins, scripts, etc.). It is clear that users are interested in using the Web in the way they prefer and they have evolved to know how to achieve it. Even in the academy there is a dizzying trend in the area of end-user programming [11]. The reason is that (1) part of the crowd of users may deal with different kinds of development tools, and (2) part of the crowd of users want to utilize these tools in order to satisfy a particular need/requirement.

Although several tools such as Platypus[1] allow users to perform the adaptation of Web pages directly by following the idea of "what you see is what you get", these kinds of tools have some limitations regarding what can be expressed and done. Thus, some Web augmentation requirements could not be addressed with them, and these have to be tackled with other kind of script, which require of advance programming skills to be developed. In this sense, we think that prototyping could be also a solution in the context of Web augmentation requirements, and augmenting Web pages with the prototypes is really a straightway for telling others what a user wants.

In addition, users are more familiarized in dealing with software products, which is partially proven by new agile development methodologies like Scrum [19] that center the development in tackling user needs by prioritizing requirements considering its business value. The main purpose of such processes is providing valuable functionality as early as possible giving a more active role to the stakeholders to assess that the implemented application is what they need and expect. It is usual to ask users for defining a requirement by using one of several techniques such as mockups [20][6] or User Stories [4], which are completely understandable by users. Since agile methodologies try to include stakeholders (including users) as much as possible in the development process, their processes usually are compatible with User-Centered Design (UCD) approaches. Among the most common requirements artifacts used in UCD, low-fidelity prototypes (informally known as mockups) are the most used [9]. Mockups are more suitable than textual requirements artifacts because they are a shared language between development team (including developers and analysts) and users [14], avoiding domain-related jargon that may lead to faults in delivered software artifacts originated by requirements misunderstandings. In addition, mockups have been proposed as a successful tool to capture and register fluid requirements [17] – those that are usually expressed orally or informally and are an implicit (and usually lost) part of the elicitation process. However, they are rarely used in an isolated way, being usually combined with User Stories [4] or Use Cases [13][8]. Finally, UI Mockups have the advantage of being simple since stakeholders can build these by themselves using popular and emergent tools like Balsamiq[2] or Pencil[3].

[1] `https://addons.mozilla.org/es/firefox/addon/platypus/`
[2] `http://balsamiq.com`
[3] `http://pencil.evolus.vn/`

Most of the development approaches regarding using mockup are centered on applications in which the number and type of end-users are predictable. However, this is not true for massive Web applications, since the crowds that use them are not only less predictable in quantity and type, but also are changing and evolving all the time. In order to satisfy as many users as possible, the application should be customized for each user. Although the mechanism of adaptation and personalization of Web applications, the so-called *adaptive Web* [2], have also evolved (for instance, complex user models [7] have been proposed and assessed), it is really hard to give adaptation mechanisms that contemplate the requirements of all the users.

In this context, two trends with two main goals have emerged. On the one hand, well-known mash-ups [23] approaches have surfaced for integrating existing Web content and services. On the other hand, a technique called Web augmentation [5] has emerged in order to allow users to customize both content and functionalities of Web pages modifying their DOMs on the client-side. Both approaches have communities that serve as a proof of how important is for the users to improve their Web experiences. An example of the former is Yahoo Pipes [22], a mash-up tool broadly utilized for combing services over the Web that has a big active community [3]. Regarding Web augmentation, the userscripts.org [21] community represents an emblematic example. This community has created thousand of scripts that run over the client for modifying the visited Web sites. Some scripts have been installed for end users over one hundred thousand times.

In both communities we observed two different user groups: (1) the crowd of users (at least part of them) want to personalize the Web applications they use and, (2) the part of this crowd can develop software artifacts for such purpose. Our main concern is to facilitate how these communities currently work. In the design of Web applications from scratch, different stakeholders participate on the definition of the functionality. This can be done since part of these reduced and well-known set of stakeholders directly *express* their requirements. There is an explicit delegation. On the contrary, most of the open communities that intend to introduce custom enhancements to existing Web sites work usually in the opposite way: a user from the crowd with programming skills develops an artifact with some features that may be useful to other users. If the artifact is good enough and responds to a common requirement, then, it will possibly be installed for many other users of the crowd.

In some cases both user and userscripters work together by asking new requirements and implementing them (correspondently) in two ways: (1) asking for new scripts in the community forums or (2) asking scripts improvements in the script's discussion Web page. In this work we aim to create new communication mechanisms for improve both the process and how resultant scripts match user requirements. The main motivation behind the approach presented here relies in the fact that, besides informal forums, the community has no clear ways to communicate to those users with development skills in the crowd, which are the *augmentation requirements* desired.

In this paper we propose to rely in mechanisms used on agile methodologies, such as User Stories or mockups, to empower users with tools for specifying their enhancements requirements over existing Web applications. The novelty of the approach is the usage of Web augmentation techniques: with our approach, a user can augment a Web page with those widgets that are relevant for describing his requirement. Then, our approach allows sharing the augmentation requirement so other *users* and *userscripters* may reproduce the same augmentation in their own Web browsers in order to understand that user's needs in a more detailed and formal way.

The paper is organized as follows. Section 2 introduces our approach. Section 3 presents the tools, and section 4 shows the results of an evaluation made with end-users. The state of the art is tackled in section 5. Finally, section 6 gives some concluding remarks and further works.

2 The Approach

We present a novel approach for UI prototyping based on Web augmentation. In this work, Web augmentation is used as the technique for defining requirements while Web augmentation communities are taken as the source of both requirements and scripts for these requirements.

Our goal is to improve the communication between *users* and *userscripters*. The essence of the approach can be described through the following example: let's consider that a user is navigating a common video streaming Web site like YouTube. When he searches for videos, he realizes that additional information in the results screen will be helpful for him in order to decide which video to watch. For instance, he would like to see the amount of likes/dislikes and also further information about video's audio quality, resolution, etc. Consider that the user knows how to install and run GreaseMonkey scripts to augment Web sites, but he does not how to develop one of such scripts. If there were no scripts satisfying his concrete expectations, he would like to have a way to communicate to GreaseMonkey scripters his requirements. This communication should be as concrete and clear as possible. Currently, userscripting communities like userscripts.org rely on textual messages interchanges as the solely artifacts to let users and userscripters collaborate. The presented approach proposes to help the end-users and userscripters collaboration through:

- Linking users requirements with *userscripters* who can implement the requirements.
- Using a concrete language for defining those requirements. The language must be understood by both developers and users and must be formal enough to describe the requirement in terms of UI components and features.
- Managing and evolving the requirements in a well-defined process.

Our approach follows the general schema shown in Figure 1. The main goal is that any user can specify the visual components related to a particular requirement by augmenting the target Web page with mockups.

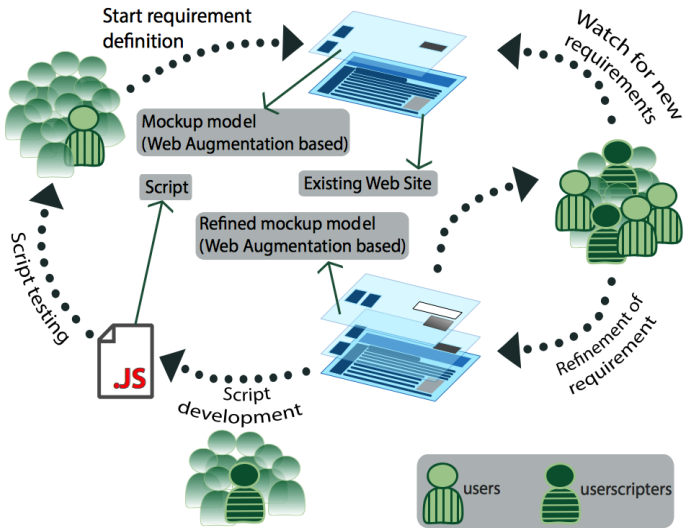

Fig. 1. Overview of the approach

Once the requirement is concretely defined (which is expressed through a model that will be introduced later); it can be uploaded into an online repository, which makes it accessible by the community to *reproduce* and enhance this. In this context, to reproduce means that any other user can augment the target Web page in the same way the user that specified the requirement did, in order to watch in detail which the required UI alterations are.

The community (i.e. users and userscripters) may colaborate in the refinement of a requirement in the repository. If a user is not capable to define the requirement with the required level of detail, another user who wants to colaborate (for example, having the same requirement) can enrich the definition. Every change is validated by the requirement's owner. When a stable version of the requirements is met, a userscripter may create the script that implements it, and then upload it into the repository. At this point, users can install and use the script, contributing with its testing.

Therefore, our approach empowers users with:

- Mechanisms for defining their requirements over existing Web pages.
- Mechanisms for managing and publishing their requirements.
- Mechanisms for proposing and bringing solutions to users, who can evaluate these solutions and eventually start the cycle again.

The main purpose of our approach, called *CrowdMock*, relies on allowing the *crowd of users* to *plug* high-fidelity *mockups* into existing Web pages. Thus, the approach involves two main contributions. On the one hand, we designed a metamodel (the *MockPlug metamodel*) for describing how high-fidelity mockups are woven into existing Web sites via Web augmentation. On the other hand, to support our approach technologically we developed a client-side tool for weaving augmentation models (called *MockPlug*) and a server-side platform for managing them collaboratively (called *UserRequirements*). MockPlug is a tool designed for allowing end-users to *plug* mockups on existing Web pages to specify their

augmentation requirements and them in a concrete, formal model. With MockPlug, users may create, update, and reproduce requirements. UserRequirements is a platform deployed as a Web application[4] and also integrated with MockPlug, for allowing end-users to share their augmentation requirements and asking for scripts that implement them. This also contemplates the evolution of the requirement and the evaluation of a particular solution allowing collaboration, traceability, and validation.

Web augmentation operates over existing Web sites, usually by manipulating their DOM[5]. If we want to augment a DOM with new widgets, we have to specify how those new widgets are woven into the existing DOM. With this in mind, we defined the MockPlug metamodel, which specifies, which kinds of augmentation (widgets and their properties) are possible within our approach and also how these are materialized for a particular Web page's DOM. Within MockPlug metamodel, which is depicted in Figure 2, both the name and the URL of the augmented Web site compose a requirement. The specification of new widgets relevant for the requirement (Widgets) may be defined in the model. Widgets can be simple (SimpleWidgets, atomic and self-represented) or composite (CompositeWidgets, acting as a container of another set of Widgets), but all of them have several characteristics in common:

- Every Widget belonging to a MockPlug model has both a type and properties.
- A Widget is prepared to respond to several events, such as *mouseover* or *click*.
- A Widget can react to an event with predefined operations.

A Widget has information related to how and where it was inserted into the DOM tree. Note that, each widget has an insertion strategy associated (*float*, *leftElement*, *rightElement*, *afterElement*, *beforeElement, etc.*).

It is worth mentioning the importance of the property *url* in a MockPlug model, since this is the property that defines which Web site (o set of Web sites when using a regular expression) the model is augmenting.

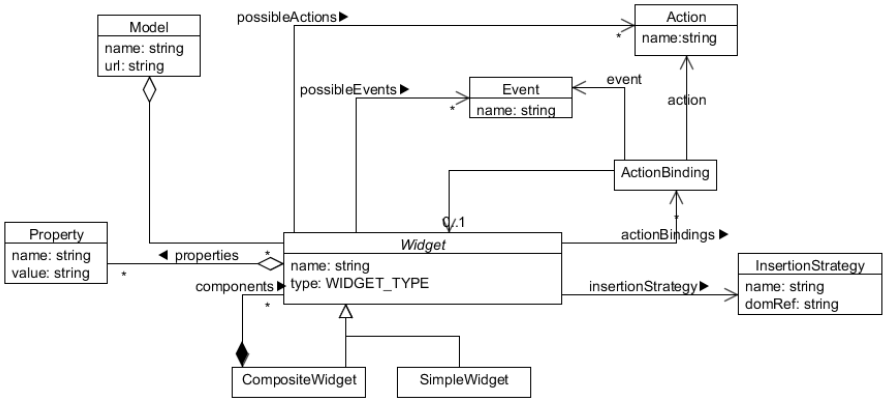

Fig. 2. MockPlug metamodel

[4] See a live demo of our approach on: `http://www.userrequirements.org`
[5] `http://www.w3.org/DOM/`

3 Tools for Web Augmentation Requirements Management

In this section we introduce the technical details behind the aforementioned MockPlug and UserRequirements implementation.

3.1 MockPlug

We considered two issues in order to empower users with mechanisms for defining their own requirements, (1) the supporting tool for such purpose has to be easy to use and (2) specifying a new requirement should not require too much effort and high technical knowledge. We developed MockPlug with this in mind. MockPlug is a Firefox extension that allows end-users to specify requirements by manipulating high-fidelity mockups over their Web sites that they want to augment. For each MockPlug requirement, a User Story is also defined.

From MockPlug's point of view, a requirement represents a set of modifications made over an existing Web site expressed in terms of the aforementioned metamodel. Thus, every requirement expressed in our approach has a MockPlug model associated, and the changes are expressed by adding widgets. From the user's point of view, these have a visual style like hand-drawn mockups similar to mockup tools like Pencil or Balsamiq. However, they are represented at runtime as ordinary DOM elements. In order to refine his requirements visually, a user may drag&drop widgets over the existing Web page, like it is shown in Figure 3. This figure depicts how the user can add a new button over the existing IMDB Web site. On the top of Figure 4, the button already added over the existing Web page is shown and the widget editor is shown at the bottom.

3.1.1 Widgets

The tool considers three kinds of widgets: *predefined* widgets, *packetized* widgets and *collected* widgets, which are described below.

Predefined Widgets

We have defined an initial set of predefined widgets (Figure 5), which are grouped in categories as Table 1 describes. Different properties may be set for each kind of predefined widget. For instance, the widget *Button* has the properties *name*, *title* and *label;* while a menu list has the *name* and a *collection of options*. Figure 5 shows how the predefined widgets are listed in the MockPlug Panel.

Pocketized Widgets

In order to allow users to easily specify integration requirements we created a functionality called *Pocket*. The Pocket (depicted in Figure 6) allows users to collect existing UI components as widgets. Its name comes from its provided functionality, which consists in allowing collecting and storing any part of an existing Web page and then place it as a *new* widget in other Web applications or pages. As an example, Figure 6 shows three already collected widgets (one collected from YouTube and two

collected from IMDB) that could be added in any other Web application in order to define some requirement of integration between these applications. The user must drag&drop the widget that had collected in the same way it did with predefined widgets, having the same tool aid and insertion strategies to apply. Pocketized widgets can be added as static *screenshots* of the original DOM or as real DOM elements. However, so far the tool does not guarantee the correct behavior defined with JavaScript routines taken from the original Web page from where the packetized widget was collected.

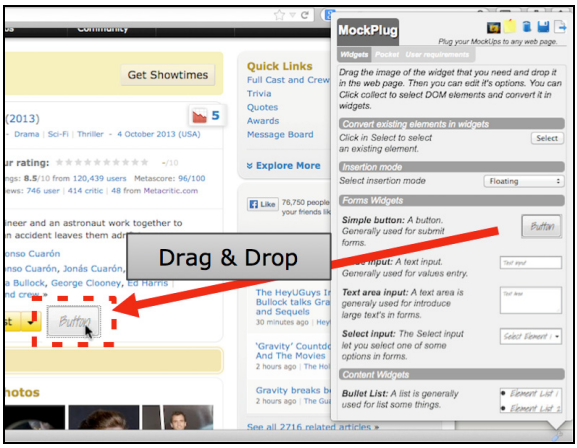

Fig. 3. MockPlug main panel

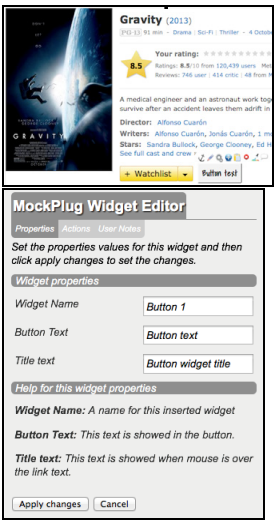

Fig. 4. Widget added and widget editor

Table 1. Predefined widgets

Category	Goal	Examples
Form Widgets	Specify requirements that involve some kind of information gathering.	Button, text input, text area, select menu
Content Widgets	Specify requirements focused on non-editable contents	Lists, images, text, anchors
Annotation Widgets	Empower users with mechanisms for describing textually, some expected behavior, presentation or content	Post-its notes, bubble comments

The user may add both predefined and pocket widgets in the DOM by using an insertion strategy. This function provides an interactive highlighting of existing elements in the DOM in order to help the user.

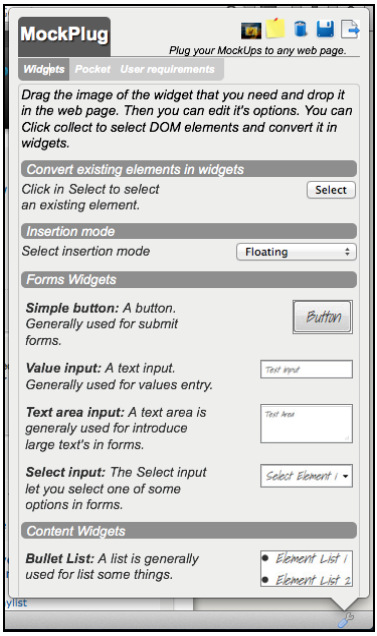

Fig. 5. Predefined Widgets panel

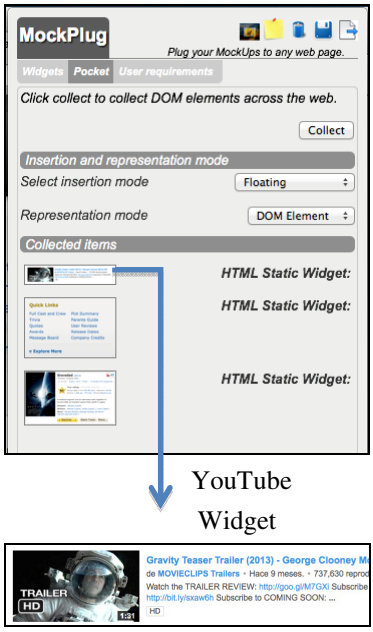

Fig. 6. Pocket panel

Collected Widgets

Although the Pocket allows users to specify some requirements of integration and also to reuse existing UI components, sometimes it could be useful to convert an existing DOM element into widgets in order to refine it. This can be accomplished using the *collected widgets* functionality provided by the tool. A *collected widget* is a widget that has been created or imported from an existing DOM element in order to manipulate it. For example, let's assume that the user wishes to indicate that a new column should be displayed in an existing table. Then, the user can convert the existing table into a new widget with the final goal of editing its contents and columns, as it is shown in Figure 7. In this case, the user collected the table containing the 250 top movies from IMDB.com.

The ability of converting existing DOM elements into widgets, also gives the possibility of removing useless items from the target web site. Since the DOM element is converted into a widget, it could be also moved to another part of the Web page. Moreover, converting previously existing DOM elements into widgets makes possible to use them as a target or reference in actions define for other widgets. For example, the user might want to add a button to show/hide a particular element of the website on which the requirement is made.

It is worth noting that collecting a DOM element with the goal of creating a collected widget is not the same that putting a widget into the Pocket. While the first one associates the underlying DOM element as a widget in the MockPlug requirements model, the second ones can be used as a sort of *clipboard* for moving UI pieces among Web applications.

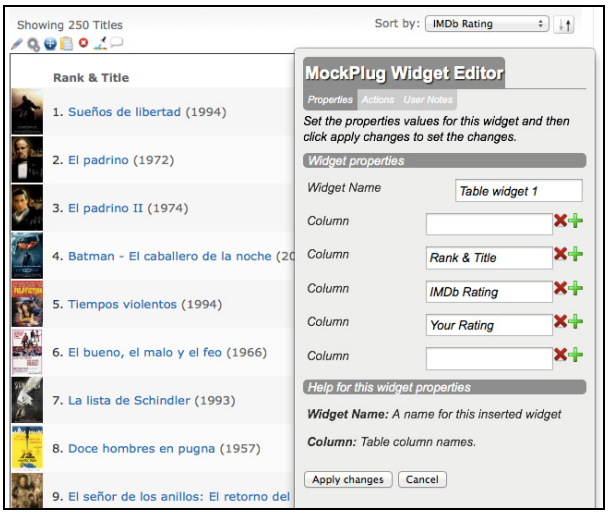

Fig. 7. Edition of collected Widget

3.1.2 Widgets Management

The widgets added with MockPlug are clearly distinguishable from the rest of the elements found in the website. However, MockPlug makes it possible to copy styles from existing DOM elements to a Widget in order to emulate an existing page style. In the example from Figure 8, the user has added a button (predefined widget) called "Button text", which is actually a DOM element (as any widget) and which with the user can interact. Different operations commonly found in mockup tools can be applied through the widget contextual toolbar (shown when the widget is selected), such as cloning, removing, moving, annotating, property editing, among others.

3.1.3 Code Generation

User requirements are abstracted with the MockPlug metamodel and their instances may be processed in order to generate code through MockPlug code generators as in well-known Model-Driven approaches [10]. The code generation capabilities in MockPlug make it possible to generate a first code stub implementing basic and repetitive features of the augmentation requirement, thus reducing the workload for the *userscripter* who wants to write a script to satisfy it. The code generator has been included by default with MockPlug and currently it is focused on obtaining GreaseMonkey scripts.

3.2 Requirements Repository

In addition to MockPlug, we have implemented UserRequirements, a repository with social features with two main purposes: (1) allowing users to collaborate in the definition and the evolution of augmentation requirements and (2) enabling *userscripters* to describe and reproduce the requirements in order to develop the corresponding script. A requirement in our repository is defined by two components in our repository:

- **A User Story (US)**, which highlights the core concerns of the requirement from a particular user role pursuing some goal in the business [4].
- **A MockPlug model**, which is the UI prototype for answering to the US. A MockPlug model is associated with only one US, but one US can be referenced in multiple MockPlug models.

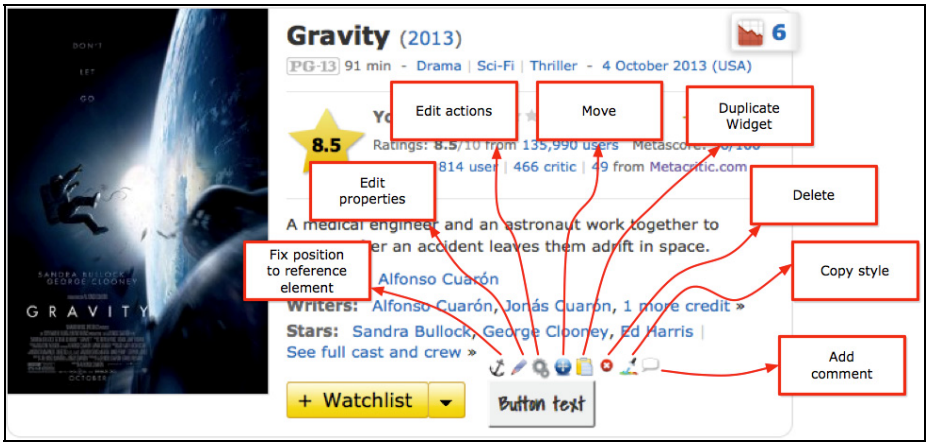

Fig. 8. Widget contextual menu

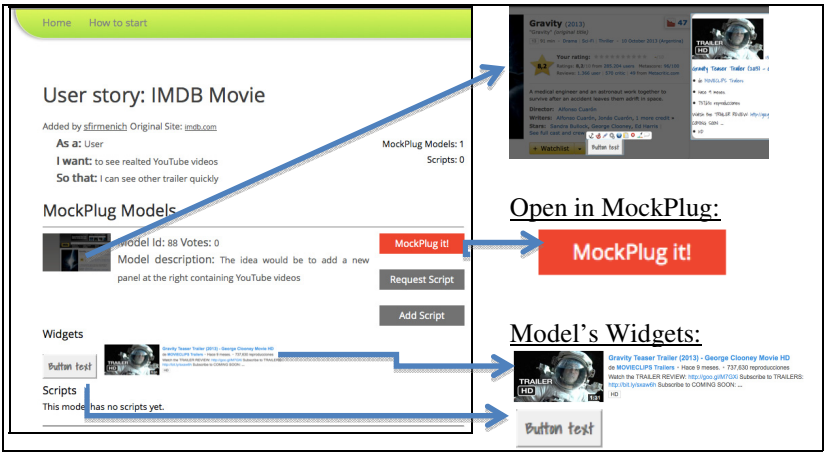

Fig. 9. Requirement view in UR

Fig. 10. UI details

Figure 9 and 10 shows a screenshot of how a requirement is described in the repository. It includes information about who created the requirement, over which Web site it was defined, its US, and a screenshot of how the target Web site was augmented with new widgets by using MockPlug.

It is really important to allow developers to reproduce and watch the resulting *augmented mockup* in action. This functionality is provided by the button labeled *MockPlug it* (see Figure 10). When the user clicks this button, a new tab is opened in order to load the target Web site and augment it with the MockPlug model built interactively using the MockPlug tool. Others users can collaborate by evolving the definition of that requirement using the tool.

The integration between the social repository and MockPlug tool is essential for our approach. We show how the user can add his current requirement in the repository on the left of Figure 11. To add a new requirement, the user has to define the User Story plus a general description of it. Finally, the requirement (formed by the US and the MockPlug model) is uploaded to the repository by clicking the *Save* button.

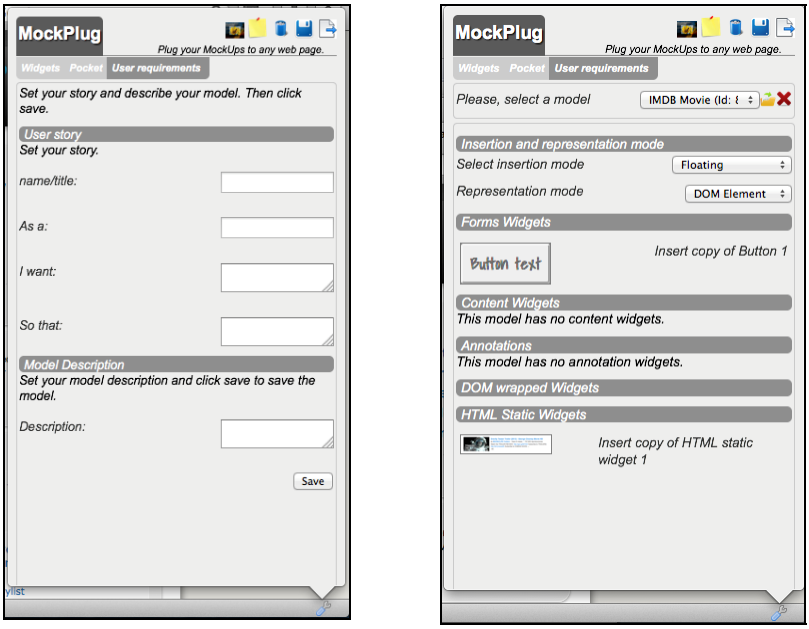

Fig. 11. Integration between MockPlug and UserRequirements

We depict how a user may choose from the menu between his already existing models in the repository on the right in the same figure. When a model is selected, all the widgets involved in that model are listed at the bottom from the same panel. The user also may choose to *open* the MockPlug model, whose action will render it in a new tab. The difference between *pre-open* (just for listing its widgets) and open a model (which actually renders it) was meant on purpose, because by pre-opening a model the user can reuse widgets defined on the model that is being defined.

For conciseness reasons we can not explain it deeply, but it is important to know that UserRequirements make it possible to "branch" and "vote" the requirements models. In this way the community may reach a personalization level of the requirement that fulfills the expectations of the majority while someone other person of the community may create a new branch of one model in order to specify further requirements.

4 Evaluation

This section describes a preliminary evaluation of the CrowdMock approach and its supporting tools. We have performed a controlled experiment focused on assessing the effectiveness achieved by experienced users when specifying requirements on existing Web applications. The experiment compared the process of specifying requirements using the proposed approach against using existing approaches. Our hypothesis was:

> *Defining augmentation requirements with CrowdMock produces better specifications than using traditional user-oriented methods (such as user stories, mockups, etc.), and consequently, CrowdMock improves the understandability of the requirements*.

4.1 Protocol and Participants

The experiment was organized into two phases: (1) defining a requirement using well-known techniques (e.g. user stories, mockups, etc...) and (2) defining requirements using MockPlug. The requirements had to be defined over well-known existing Web applications: IMDB[6] and YouTube[7]; and they had to be based on one of the following features. The features for IMDB were:

- R1.1: Filter the list of 250 best movies
- R1.2: Add some information to the 250 best movies list.
- R1.3: Change the layout and/or content of a movie's page based on the following issues:
  - Move elements of the page.
  - Remove elements from the page
  - Add new widgets into the page (button, input box, menu, etc.)
  - Add contents related to the movie from another IMDB page.

The features for YouTube:

- R2.1: Add information in the search results about the videos.
- R2.2: Idem to R1.3 focusing on a YouTube video's Web page.

Participants had to choose 2 features in each phase. Thus, each participant specified 4 requirements. Since 8 subjects participated in the experiment, a total of 32 requirements were specified. Half of them (16 requirements) had to be defined with the participants' preferred requirements artifacts and elicitation techniques, and the other 16 requirements had to be defined using MockPlug. All the 32 requirements were implemented by means of GreaseMonkey scripts, which were developed by a specific team at LIFIA.

[6] Internet Movie Data Base - `http://imdb.com`; last accessed 4-Feb-2014.
[7] YouTube – `http://youtube.com`; last accessed 4-Feb-2014.

The whole experiment was organized as follows:

1. First phase:
   a. A first face-to-face meeting with participants was organized for presenting the experiment and explaining the tasks. Before this meeting, participants had been introduced in the use of US as well as in user interface mockup building techniques and tools. In this meeting the definition of two requirements choosing one specific technique and one or more requirements artifacts was exemplified.
   b. Participants had 3 days to write two requirements. Then, they had to send their specifications (including the requirements artifacts) and an activity diary where they registered all their work and the time spent on it.
   c. When all the requirements were collected, developers had 2 days to create the 16 GreaseMonkey scripts. Both 2 scripts implementing the requirements were sent to every participant.
   d. Participants had 2 days for installing, using and evaluating both scripts and filling in two questionnaires:
      i. Questionnaire A, oriented to measure and describe the difficulty perceived during the specification of the requirements.
      ii. Questionnaire B: oriented to measure and describe how well the script satisfied his expectations.
2. Second phase:
   a. Another face-to-face meeting was organized for presenting both the new tasks to be accomplished and the tools involved in this phase:
      i. The UserRequirements (UR) application (i.e., the requirements repository) was introduced. Participants were asked to create a user account on the platform in order to start specifying requirements through it.
      ii. MockPlug was also explained and participants learned about how to upload the requirement from MockPlug to UR.
      iii. The second evaluation task was presented. It consisted in defining two requirements using MockPlug and UR. Participants had to choose only requirements not chosen in the first phase.
   b. Participants had 3 days to define the requirements and then upload them to the repository. Then, they had to send separately a document including the activity diary.
   c. With every requirement uploaded to UR by participants, the developers received a notification. Again, developers had 2 days to create the 16 GreaseMonkey scripts. When all the requirements were uploaded, instead of sending the scripts personally, developers uploaded the script for each requirement into UR and participants received the corresponding notification.
   d. Participants had 2 days for downloading and using both scripts. After that, they were asked to filling in the same two questionnaires (A and B). Also, they were asked to fill in another SUS [1] questionnaire for evaluating the usability of our tools.

Following a within subjects design, a total of 8 participants were involved in this evaluation. All participants were professionals on computer science from a post-graduate course on Web Engineering; 4 female and 4 male and aged between 24 and 60. It is important to mention that the 8 participants were instructed (before the experiment) in the installation and use of GreaseMonkey scripts.

4.2 Results

This section describes the results of the experiment. First, we describe the requirements specified by every participant in Table 2. Afterwards, we present the analysis.

4.2.1 First Phase

Table 2 shows depicted with "X" the requirements selected in the first phase for each participant. The average time for defining each requirement was 92.5 minutes (SD = 62.12 minutes). The techniques used were mockups (traditional ones), User Stories and some more textual description.

The difficulty perceived when specifying each requirement ranged from "normal" to "very easy". Most of the requirement specifications (11) were considered as a "normal" task in terms of difficulty (68.75%) in a scale from 1 to 5 (where 1="very easy", 2="easy", 3="normal", 4="difficult", 5="very difficult"). This task was considered "easy" for 4 requirements (25%), and "very easy" only for 1 (6.25%).

Table 2. Requirements selected by each participant (X = First Phase, O = Second phase)

	R1.1	R1.2	R1.3	R2.1	R2.2
Participant 1		X	X	O	O
Participant 2		O	O	X	X
Participant 3		O	O	X	X
Participant 4		X	X	O	O
Participant 5		X	O	O	X
Participant 6		X	O	O	X
Participant 7	O	X		X	O
Participant 8		X	O	X	O

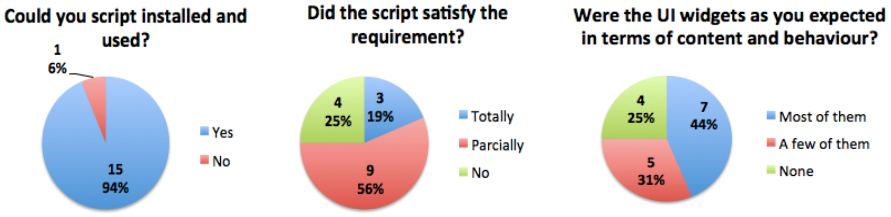

Fig. 12. Phase 1 results

After receiving all the augmentation requirements specifications, we implemented them and sent the scripts to every participant. They had to download, install and use these scripts in order to evaluate them. The results are shown in Figure 12.

4.2.2 Second Phase Results

Requirements chosen for the second phase of the experiment are depicted with "O" in Table 2. As commented previously, participants had to use MockPlug and UserRequirements in this phase. Defining a new requirement took in average 86.56 minutes (SD = 66.35). However, the difficulty perceived in the specification of each requirement had a wide range. The difficulty of the specification task was considerer "normal" (25%) in 4 requirements. Another 25% were considered "difficult" while "very difficult" was used in one requirement (6.25%). Other 37.5% (6 requirements) were qualified as "easy". Finally, 1 requirement specification was considered "very easy".

We believe the reasons were: (1) people were not experienced in the use of Web augmentation tools and (2) some incompatibilities between participants' Firefox extension version and our plug-in. This was finally confirmed by the SUS questionnaire. MockPlug scored 74.9, which is an improvable result regarding the usability of the tool.

Despite of these contingencies, we observed that participants were more satisfied regarding the implementation of their requirements when using MockPlug. It would indicate that the scripts in this phase were closer to user expectations than those developed for the first one. This information is depicted in Figure 13.

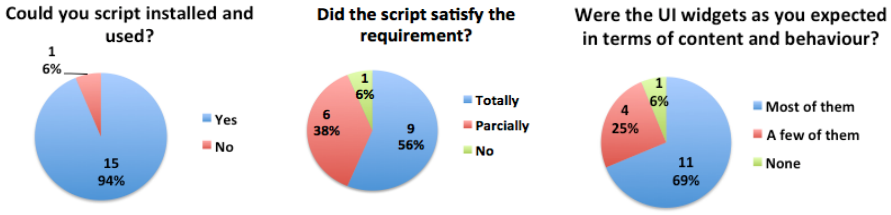

Fig. 13. Phase 2 (CrowdMock) results

4.2.3 Discussion

The results of the controlled experiment showed that when users freely *choose* the specification technique only 18.75% of the requirements were totally satisfied and 56.25% were partially specified. The remaining 25% of the scripts did not satisfy absolutely user requirements.

In the second phase, by contrast, 56.25% scripts fulfilled user's expectations, while 37.5% satisfied them partially. Because one of the scripts was not able to be executed, just one requirement (6.25%) was not satisfied.

Our tools were not well known by the participants, who reported several usability issues that are reflected in the difficulty they perceived during the task of specifying a

requirement with MockPlug. In spite of these problems, our approach allowed to participants to be more satisfied. We think that this difference is due to: (1) aid provided by mockup for focusing on specific elements (UI widgets) which resulted in more concrete and clearer specifications for scripters, (2) the possibility of manipulating existing UI components easily, and (3) the possibility of defining behavior for each widget without programming knowledge.

Another additional benefit of using our approach was related to the development process of the scripts. By using MockPlug and MockPlug models, some code could be generated automatically, at least what is related to the fact of creating new widgets and get the existing DOM elements from the target Web page that were manipulated.

5 Related Works

In this section we compare our approach with two kinds of works. First, we analyse other mechanisms and tools to specify UI prototypes. On the other hand, we review some approaches to elicit requirements in the context large crowd of users.

Using UI mockups as a requirement elicitation technique is a growing trend that can be appreciated just observing the amount of different Web and desktop-based prototyping tools like Balsamiq and Mockingbird[8] that appeared during the last years. Statistical studies have proven that mockups use can improve the whole development process in comparison to use other requirements artefacts [15]. Mockups use has been introduced in different and heterogeneous approaches and methodologies. On the one hand, they have been included in traditional, code-based Agile settings as an essential requirement artifact [20] [6]. On the other hand, they have been used as formal specifications in different Model-Driven Development approaches like MockupDD [16], ranging from interaction requirements to data intensive Web Applications modelling. In this work we propose to specify augmentation changes using (among other techniques) mockup-style widgets. Also, MockPlug combines mockup-style augmentation techniques with well known annotations capabilities of common mockup tooling, that also can be used as formal specifications in the future as in [14].

There are new works on collaboration in requirements elicitation. Some authors have used social networks to support requirements elicitation in large-scale systems with many stakeholders [18]. They have used collaborative filtering techniques in order to obtain the most important and significant requirements from a huge number of stakeholders. A more formal and process-centered approach is proposed in [12]. However, we consider that none of the approaches provide a mechanism to collaboratively elicit, describe and validate requirements by using prototype definition for large-scale systems with a large number of stakeholders. The aforementioned works are limited to eliciting narrative requirements. We propose to define requirements by specifying high-fidelity prototypes that are later managed collaboratively from a common repository.

[8] `http://gomockingbird.com`

The ability of creating, validating and evolving prototypes collaboratively just by dragging and dropping widgets over existing Web sites, may allow stakeholders to express their needs in more accurate way, beyond textual descriptions [1]. In the particular context of Web augmentation communities, by improving communication between users and userscripters, we ensure that user requirements are correctly understood before future development of the concrete scripts. In this way, our integrated platform aims to reduce users' efforts when defining their requirements and upload them to the repository where, at the same time, it enables whole the community to improve them iteratively.

In the context of Web augmentation for annotations, there are also many tools to share user's annotations – for instance, Appotate[9] or Diigo[10]. These kinds of tools allow users to share annotations across the web, although these are very useful to share and highlight ideas, annotations are not enough to define complex augmentation requirements.

By using traditional mockups tools users can create their own prototypes, but usually have to start from scratch and widgets are not linked to the Web site where the requirement surfaces. This simple fact would demand much effort from scratch (in order to give some context to the augmentation requirement). Besides that, Web augmentation requirements could be extremely related to existing UI components, and desirably, the prototype should show how these existing components interact with the new widgets. Our approach enables users to quickly define high-fidelity prototypes related to the augmentation requirement, manipulating the same Web page that is going to be transformed by a script in real time. Our MockPlug metamodel makes it possible to share and reproduce these prototypes, but additionally, it also makes possible to process a prototype in order to generate a first script template based on those manipulated widgets.

6 Conclusions and Future Work

It is very complex to gather requirements from a crowd of users for widely used Web applications. In this work we have presented a methodology and its implementation for gathering requirements in the context of Web augmentation communities. Our approach is inspired in agile methodologies were users have an active role. We found that using high-fidelity mockups is very useful for specifying Web augmentation requirements using the running Web page as a foundation for specifying them graphically. With *CrowdMock*, users may define their own requirements and share them with the community, who can reproduce, edit and evolve them collaboratively. CrowdMock and its tools have been evaluated obtaining some benefits in comparison with traditional approaches. By using the proposed high-fidelity mockups that run over existing Web pages, we agilize the definition process since it is not necessary to construct an abstract conceptualization from existing UI. Users know what they want from Web applications and we try to give them the tools and mechanisms for naturally expressing their requirements.

[9] `http://appotate.com`
[10] `https://www.diigo.com`

We described the details of an evaluation of CrowdMock approach, which showed some positive results in Web augmentation requirements gathering and also some usability issues in the MockPlug tool that we are currently addressing. Integration between MockPlug and UserRequirements is also being improved. Additionally, we have an ongoing work on the server-side application (UserRequirements), which is being enhanced to support better collaboration mechanisms.

Another interesting further work path includes testing support and better code generation. We are planning to automate test for userscripts. Code generation is currently made without considering good *userscripters* practices and patterns; thus, the generated code may be not be easy to understand or manually adapt by userscripters. Thus, we are working on improvements our MockPlug code generators API in order to allow *userscripters* to develop their own code generators, which are a particular type of MockPlug plug-ins.

Although we are focused on Web augmentation requirements, our approach could also be useful for projects intended to build software products from scratch instead of augment existing ones. Thus, we are planning to improve the expressiveness of our metamodel, extending the tool with new kind of widgets and finally improve the server-side repository with features related to general developer support.

References

1. Brooke, J.: SUS: A 'quick and dirty' usability scale. In: Usability Evaluation in Industry, pp. 189–194. Taylor and Francis, London (1996)
2. Brusilovsky, P., Kobsa, A., Nejdl, W.: The Adaptive Web (2008)
3. Cameron Jones, M., Churchill, E.: Conversations in Developer Communities: a Preliminary Analysis of the Yahoo! Pipes Community. In: Proceedings of the Fourth International Conference on Communities and Technologies, pp. 195–204. ACM (2009)
4. Cohn, M.: User stories applied: for agile software development, p. 268. Addison-Wesley (2004)
5. Díaz, O.: Understanding Web augmentation. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE Workshops 2012. LNCS, vol. 7703, pp. 79–80. Springer, Heidelberg (2012)
6. Ferreira, F., Noble, J., Biddle, R.: Agile Development Iterations and UI Design. In: Proceedings of AGILE 2007 Conference, pp. 50–58 (2007)
7. Gauch, S., Speretta, M., Chandramouli, A., Micarelli, A.: User Profiles for Personalized Information Access. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) Adaptive Web 2007. LNCS, vol. 4321, pp. 54–89. Springer, Heidelberg (2007)
8. Homrighausen, A., Six, H., Winter, M.: Round-Trip Prototyping Based on Integrated Functional and User Interface Requirements Specifications. Requir. Eng. 7(1), 34–45 (2002)
9. Hussain, Z., Slany, W., Holzinger, A.: Current state of agile user-centered design: A survey. In: Holzinger, A., Miesenberger, K. (eds.) USAB 2009. LNCS, vol. 5889, pp. 416–427. Springer, Heidelberg (2009)
10. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society (2008)

11. Ko, A., Abraham, R., Beckwith, L., Blcakwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M., Rothermel, G., Shaw, M., Wiedenbeck, S.: The State of the Art in End-User Software Engineering. ACM Computing Surveys, 1–44 (2011)
12. Konaté, J., El Kader Sahraoui, A., Kolfschoten, G.: Collaborative Requirements Elicitation: A Process-Centred Approach. Group Decision and Negotiation Journal (2013)
13. Kulak, D., Guiney, E.: Use cases: requirements in context. Addison-Wesley (2004)
14. Mukasa, K., Kaindl, H.: An Integration of Requirements and User Interface Specifications. In: Proceedings of 6th IEEE International Requirements Engineering Conference, pp. 327–328 (2008)
15. Ricca, F., Scanniello, G., Torchiano, M., Reggio, G., Astesiano, E.: On the effectiveness of screen mockups in requirements engineering. In: Proceedings of ACM-IEEE Int. Symp. Empir. Softw. Eng. Meas. ACM Press, New York (2010)
16. Rivero, J.M., Rossi, G.: MockupDD: Facilitating Agile Support for Model-Driven Web Engineering. In: Sheng, Q.Z., Kjeldskov, J. (eds.) ICWE Workshops 2013. LNCS, vol. 8295, pp. 325–329. Springer, Heidelberg (2013)
17. Schneider, K.: Generating fast feedback in requirements elicitation. In: Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality, pp. 160–174 (2007)
18. Lim, S.L., Finkelstein, A.: StakeRare: Using Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation. IEEE Transactions on Software Engineering 38(3), 707–735
19. Sutherland, J., Schwaber, K.: The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process, `http://assets.scrumfoundation.com/downloads/2/scrumpapers.pdf?1285932052` (accessed: February 16, 2014)
20. Ton, H.: A Strategy for Balancing Business Value and Story Size. In: Proceedings of AGILE 2007 Conference, pp. 279–284 (2007)
21. UserScripts, `http://userscripts.org` (accessed: February 16, 2014)
22. Yahoo Pipes, `http://pipes.yahoo.com/pipes/` (accessed: February 16, 2014)
23. Yu, J., Benatallah, B., Casati, F., Florian, D.: Understanding mashup development. IEEE Internet Computing 12(5), 44–52 (2008)