Script Programmers as Value Co-creators

Cristóbal Arellano, Oscar Díaz, and Jon Iturrioz

ONEKIN Research Group, University of the Basque Country,
San Sebastián, Spain
{cristobal.arellano,oscar.diaz,jon.iturrioz}@ehu.es
http://www.onekin.org/

Abstract. Website owners are gradually realising the benefits of viewing customers as co-creators of value. Unfortunately, current development models offer little help in understanding and managing this new form of value co-creation. The Metropolis Model has recently identified three realms of roles for crowdsourcing: the kernel (providing the core functionality), the periphery (the partners) and the masses (the end users). Technically wise, the periphery requires mechanisms for the commons to suggest, develop and maintain additional services on top of the kernel. This work concretizes the Metropolis Model for crowdsourced website development based on user scripts. We outline some technical challenges to foster the relationship between end users (the masses), scripters (the periphery) and the web site (the kernel) on the way to promote script-based crowdsourcing.

Key words: Greasemonkey, JavaScript, Crowdsourcing, Web2.0

1 Introduction

Crowdsourcing entails a change in how organizations perceive their relationships with their customers—"the crowds"—and how they leverage them and their resources. Kazman and Chen introduce the Metropolis models where the collaborative development of a city is used as a metaphor of the new crowdsourcing paradigm [2]. They distinguish three realms of roles (and associated infrastructure) within a Metropolis Model: kernel, periphery and masses. Example roles for people involved in the kernel include architect, business owner, and policy maker; roles at the periphery include *developer* and *prosumer*; and roles for the masses include customer and end user. This work focuses on the periphery.

The periphery is populated by developers and prosumers. Unlike developers that do not need to be consumers of the final application, prosumer activities entail the creation of products and services by the same people who will ultimately use them, on the grounds that a large rate of interesting innovations come not from the suppliers but from the users. However, while mechanisms to sustain the *developer* role have been proposed based on the open-software movement, prosumers have been generally overlooked. An exception is Web2.0

applications where wikis and blogs move *prosumers* at the forefront as the value creators. Web2.0 sites offer consumers the means to become also the purveyors of the website content. So far, this contribution is basically limited to basic interactions such as typing some text, providing tags, or clicking some buttons for ranking.

This work departs from this scenario, and focus on consumers that have some programming skills: **the scripters**. Scripters are first, consumers of websites, but they use their programming skills to improve not just the content but the functionality of the website. Script repositories such as *userscript.org* accounts for up to 36,057 registered users, while downloads are counted by millions. This large number of downloads evidences that there is a lot of value to end-users to repurpose the valuable functionality of websites for their own needs. But, what is the gain for webmasters?

Scripting can imply a threat to the monetization model of the website (e.g. banner removal), hence, it is currently perceived as a threat rather than an opportunity. As a result, scripting has to face no-collaborative, if not obfuscatory practices from the side of the website. We claim however that there is a lot to be gained by moving scripters from the masses to the periphery, i.e. as co-creators of value. Our contention is that the larger the number of scripts available, the more valuable and useful becomes the website. In an increasingly crowded panorama with distinct websites offering similar functionality, the website's periphery could mean the difference between success and failure.

This document outlines some technical challenges in developing the periphery, namely: (1) insufficient decoupling between user scripts and the underlying website (i.e. the platform); (2) no means for website customers to know about user scripts; and (3), lack of mechanisms for websites to certify scripts.

2 Script Development

So far, user scripting is a "lonely practice". No collaboration among scripters, end users and website owners. Specifically, all effort involved in developing those scripts can become useless when the underlying website is upgraded. Since scripts act directly upon *DOM* trees, minor changes in the *DOM* tree can make *XPath* functions retrieve the wrong node. This "sword of Damocles" certainly discourages scripters for writing elaborate scripts, and distances your website from becoming a platform.

Crowdsourcing implies encouraging contributor participation. In our setting, this implies sheltering scripts from upgrades in the underlying website. To this end, we propose the notion of *Modding Interface* [1]. So far, scripts rest on low-level User Interface (UI) events. Those events are dependent on the current HTML realization. If the page is upgraded, the scripts can stop working. The *Modding Interface* aims at sheltering the scripts from UI events by abstracting UI events into so-called *Conceptual Events*. A *Conceptual Event* describes the happening of an action (e.g. load, mouseOver, etc) on a *Concept*. A *Concept*

stands for a *data compound whose rendering is liable to be modded as a unit.* In this way, websites are described in terms of the concepts their pages convey, hiding the circumstantial representation of these concepts in HTML. For instance, *"Paper"* could be a concept for *icwe2009.webengineering.org*, *"Bookmark"* for *del.icio.us*, *"Book"* for *amazon.com*, etc. Now scripts subscribe to these events rather than UI events. Hence, upgrades on how concepts are supported do not impact the scripts. The *Modding Interface* for the ICWE'09 site can be obtained at *modding.icwe2009.webengineering.org/mi.xml*.

3 Script Advertising

So far, script finding is achieved through script repositories. *Userscripts.org* is a case in point. These repositories act as yellow pages which offer general information about scripts. In most cases, the user is forced to install the script to see what the script looks like. This difficulty in both finding and understanding the script are detrimental for the end user but also for both the script programmer and the website.

The website should take a more active role in publicizing community-provided scripts. So far, websites provide no indication about the scripts available to augment the base experience. However, it is for the benefit of the website to expose those augmented experiences to its customers.

The first step is to make the user aware that the website offers some augmented experiences (i.e. user scripts). This situation is similar to advertise the existence of *RSS* channels. Current browsers are able to detect a special "meta" in the HTML heading that causes the *RSS* icon to be displayed in the menu bar. However, this is not the case for "script channels". Therefore, the website itself should provide some rendering means to make users aware of this

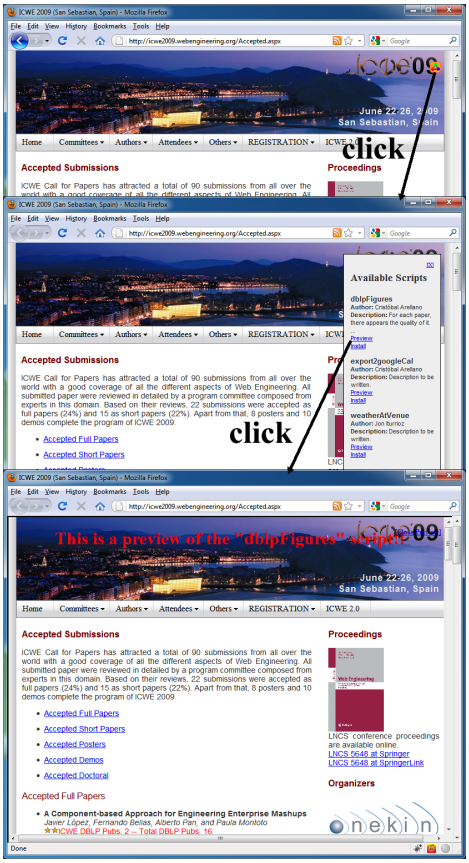

Fig. 1. Advertising user scripts through the web site.

channel. After all, *RSS* icons are still visible in most pages offering this service. Figure 1 shows the sample page but now a script icon is displayed on the right side. By clicking on this icon, a menu bar is worked out that lists the distinct user scripts available for this page. Besides the name, each user script includes a brief description, one link to preview, and another link to install. Clicking on the preview causes the page to be refreshed. Now the base experience is augmented with the user script functionality.

4 Script Sandboxing

By publicizing community-provided scripts, the website's reputation might be at risk. From a user perspective, the responsibility of user script malfunction tends to be handed over from the scripter to the website. Mechanisms are needed to ensure sound user scripts before being released to the general public.

Traditionally, the developer is responsible for the design, implementation and execution of the test cases. Now, the platform (i.e. the website) is at least as interested as the script developer to ensure the safety of the script. This first implies the existence of a clear interface between the platform and the user scripts. And second, an active involvement of the platform in defining the test cases to be passed for the script to be certified (i.e. for the script to be available through the website).

To this end, we introduce the *Modding Contract*. This contract states script functionality in terms of the causal relationship between event subscriptions and publications. These contracts are located at the script's metadata block. For instance, the contract: *[loadPaper -> appendChildPaper]* states that each *loadPaper* event will cause an *appendChildPaper* to be signalled. It can be understood as follows: if the pre-condition *"loadPaper is risen"* is satisfied then, the postcondition *"the script will raise an appendChildPaper event"* will be ensured.

Unit testing can be used to check that a script meets its contract assuming its subcontractors meet theirs (i.e. the website generated appropriate *Publishing Events*). However, unit-testing design is time consuming and requires knowledge about how to obtain full coverage of test cases. This puts an extra burden on the script developer. On the other side, it is in the own interest of the website to thoroughly validate the user scripts. On these grounds, our approach leaves to the platform (i.e. the website) the duty of defining the test units. Scripts need first to be verified against these test units. Only if test passes, the user script is publicized, and liable to be installed.

References

1. O. Díaz, C. Arellano, and J. Iturrioz. Layman Tuning of Websites: Facing Change Resilience. In *the 17th International Conference on World Wide Web (WWW '08)*.
2. R Kazman and H. Chen. The Metropolis Model: A New Logic for Development of Crowsourced Systems. *Communications of the ACM*, 52:76–84, 2009.