The Modding Web: Layman Tuning of Websites

Cristóbal Arellano (Student),
Oscar Díaz, and Jon Iturrioz (Supervisors)

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

Abstract. The Web is still much regarded as a user space rather than an author space. Hence, Web engineering cares for both current *user* requirements (e.g. usability) and future *user* requirements (e.g. maintainability), but overlooks *author* needs. This tendency is already observed in the increasing availability of open APIs and mashup applications. This work addresses another way of end user authorship, client scripting, whose vigour is evidenced by initiatives such as *Greasemonkey*. Client scripting permits end users to locally customize content, layout or style of their favourite websites. But current scripting suffers from a tight coupling with the website. As a result, website upgrades can make the script to fall apart. This can refrain users from participating, and slow down open innovation for website owners. To avoid this situation, this work proposes to characterise websites with a "*tuning interface*" in an attempt to decouple layman's script from website upgrades. Scripts do not longer access the website code (i.e. the implementation) but a stable description of the website (i.e. the interface). This interface limits tuning but increases change resilience, and offer a balance between openness (scripter free inspection) and modularity (scripter isolation from website design decisions).

Key words: Personalization, Tuning, Web 2.0

1 Introduction

The evolution of Web applications can be staged based on the degree of layman's involvement. In Web 1.0 applications, layman activities are mainly restricted to reading and form filling. Next, Web 2.0 puts content authoring in the user's hand: blogging and tagging are nowadays common practices among "webies". This work is about the last frontier of layman participation, tuning.

Traditional adaptive techniques [2] permit to adjust websites to the user profile with none (a.k.a. adaptive) or minimum (a.k.a. adaptable) user intervention. Traditional adaptive techniques seem to follow the motto of the Enlightenment movement (i.e. "*Everything for the people, nothing by the people*") where data is collected to improve the user experience but avoiding user involvement as much as possible. Yet, it is not always easy for designers to foresee the distinct customization settings and user profiles.

No design can provide information for every situation, and no designer can include personalized information for every user [9].

Hence, traditional customization approaches do not preclude the need for a do-it-yourself (DIY) approach where users themselves can locally tune websites for their own purposes [3,4,7]. To denote this scenario, the term "*tuning*" is borrowed from hardware and computer-game practices to denote *the practice of locally changing an existing website by the layman for the layman's purposes*.

So far, a popular DIY technology to website tuning is *JavaScript* using special weavers such as *Greasemonkey* [1]. A script can *react* to events when interacting or loading a page. The script *accesses* any DOM[1] node of the page. And finally, the script can also *change* the DOM at wish. But this freedom has a trade off. Making use of the knowledge about how a page is implemented, can make the script bound to the actual page structure, data and style. If the page changes, all the scripting can fall apart. The problem is that websites are reckoned to evolve frequently, and this can jeopardize all your efforts in tuning the website. This is a main stumbling block for robust, scalable and widely-adopted scripting.

2 Problem Statement

Tuning implies at least three actors, namely,

* the *base website*, i.e. the website to be tuned. It is perceived as an agent that delivers DOM documents,

* the *scripter*, an end user that modifies the rendering of the website to perform a function not originally conceived or intended by the webmaster. This is achieved through client-side scripting, specifically, *JavaScript*. For the purpose of this paper, it suffices to say that a script operates on the DOM tree that realises the website's page. The script is triggered by low-level, User Interface (UI) events on this DOM tree (e.g. *load, click, ...*). Finally, the script's action can cause the tree to be updated: deleting/adding nodes on the actual DOM tree,

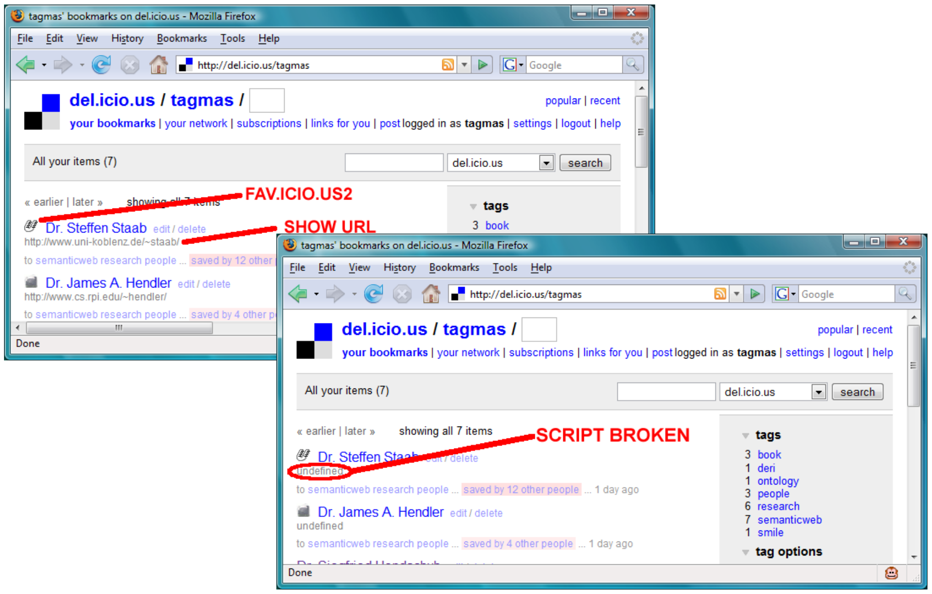
* the *weaver*, a mechanism for inlaying the script's outputs into the *base website*. Since websites are delivered to the user through general-purpose browsers, *weavers* are realized as extensions to browsers. This is the case for Firefox (e.g. *Greasemonkey*), Internet Explorer (e.g. *Turnabout* or *Trixie*), Opera (e.g. *User javascript*), Safari (e.g. *SIMBL+GreaseKit*) and Chrome (e.g. *Greasemetal*). This work uses *Greasemonkey* and *Firefox*, though the insights can be easily extrapolated to other browsers.

As a real case, consider that *fav.icio.us2*[2] script is simultaneously deployed with *delicious_show_URL*[3] script at *delicious* base website. The latter script adds the URL link before the bookmark tags. Both scripts work fine when installed separately. However, if both are run concurrently, the final output is not as expected. Based on the raw page structure, both scripts place new content and in so doing, modify this

[1] The Document Object Model (DOM) is a platform- and language-independent standard object model for representing HTML or XML documents as well as an Application Programming Interface (API) for querying, traversing and manipulating such documents.

[2] Script http://userscripts.org/scripts/source/3406.user.js thanks to Vasco Flores.

[3] Script http://userscripts.org/scripts/source/7043.user.js thanks to Noah Sussman.


FAV.ICIO.US2
SHOW URL
SCRIPT BROKEN
tagmas' bookmarks on del.icio.us - Mozilla Firefox
File Edit View History Bookmarks Tools Help
del.icio.us / tagmas /
popular | recent
your bookmarks | your network | subscriptions | links for you | post
logged in as tagmas | settings | logout | help
All your items (7)
search
« earlier | later »
showing all 7 items
tags
tag options
Done

Dr. Steffen Staab edit / delete
http://www.uni-koblenz.de/~staab/
to semanticweb research people ... saved by 12 other people
Dr. James A. Hendler edit / delete
http://www.cs.rpi.edu/~hendler/
to semanticweb research people ... saved by 4 other people

Dr. Steffen Staab edit / delete
undefined
to semanticweb research people ... saved by 12 other people ... 1 day ago
Dr. James A. Hendler edit / delete
undefined
to semanticweb research people ... saved by 4 other people ... 1 day ago

3	book
1	deri
1	ontology
3	people
6	research
7	semanticweb
1	smile

Fig. 1. Script co-existence. Installation order matters: (up) *delicious_show_URL* before *fav.icio.us2* (down) *fav.icio.us2* before *delicious_show_URL*.

page structure. This makes the final result dependent on the order in which these scripts are enacted. Figure 1 shows the results: (up) *delicious_show_URL* before *fav.icio.us2*, and (down) *fav.icio.us2* before *delicious_show_URL*. Notice that in Figure 1(down) the *delicious_show_URL* script does no longer work, the cause being that the structure of the page has been changed by the first script. This highlights the weakness of the current approach that does not scale up to even two simple scripts, not to mention deeper changes in the base website.

This certainly hinders our vision of the modding Web by refraining *Greasemonkey* practitioners from becoming a mature community, not so in size but on the complexity of the scripts available. So far, most scripts are just few lines long, and the lack of "stable platforms" (i.e. websites) in which scripts can be anchored, is certainly a main stumbling block. As learnt from previous experiences in Software Engineering, the approach is to abstract the way scripts are developed by moving away from "the implementation platform" (basically, the DOM document, and the UI events). This is the aim of the tuning interface.

3 Tuning Requirements

Therefore, the challenge is not on feasibility. The previous mods can be supported with current *JavaScript* technology. The difficulty stems from making previous scripting robust and easy enough for laymen to do themselves, not just for experience practitioners. More to the point, the larger the number of scrapped websites, the more exposed is your script to website upgrades (e.g. upgrades in the *delicious* site can also break the script apart) [6,8]. Additionally, scrapping is tedious and error-prone which prevents you from focusing on the real value of your script: the integration.

Based on these observations, we strive for modding to be:

- non-disruptive. Rather than providing a bright-new paradigm, we stick with the *JavaScript* programming model. *JavaScript* is mainstream among Web practitioners, even more with the booming of *AJAX*. Therefore, this project is targeted to the large *JavaScript* community, not just the professionals but the long-tail of amateur programmers whose contributions can serve small communities (e.g. users of *delicious*) but their total value is very significant as demonstrated by *Greasemonkey* supporters.
- agile. This goal has been traditional achieved through three strategies: encapsulation, modularity, and loose coupling. So far, scripting can be a daunting endeavour since these strategies are not common place. This calls for bringing componentware to the scripting realm.
- low footprint. The solution should account for easy adoption and minimal system requirements. To this end, tuning should be built on top of the existing Web standards and browser implementations.

Therefore, *JavaScript*, componentware and standard compliance define our strategy for Web modding. So far, modding is mainly a programming activity. However, meeting the above requirements advice to perceive websites as components, and the script as the glue that keeps these components together. Componentware facilitates to meet the agile requirement through [5]:

- components should be preexisting reusable software units which developers can reuse to compose software. Specifically, "black box reuse", which allows using an existing component without caring about its internals, as long as the component complies with some predefined set of interfaces,
- components should be produced and used by independent parties. That is, component developers need not be the same people as component customers, such as system developers. This is important for ensuring that components are truly reusable by third parties,
- components should be composable into composite components which, in turn, can be composed with (composite) components into even larger composites (or subsystems), and so on. Composition means not only reuse but also a systematic approach to system construction.

4 Research Objectives

This work raises two research questions,

1. how can *websites* be developed that facilitate layman tuning while still permitting the website to evolve?
2. how can *scripts* be developed so that they do not interfere with the tuned website, hence, ensuring resilience to website upgrades?

By addressing these questions, this work introduces the notion of "*tuning interface*". This interface is provided *by the website for scripters* to safely build on top of it. It aims at shielding the script from design decisions that are likely to change in the website. These decisions are restricted to those of how content is structured, rendered or browsed. Accordingly, a *tuning interface* declaratively specifies *what*, *when* and *how website's pages* can be *safely* changed, i.e. changes being resilient to upgrades on the underlying website. In this way, we attempt to find a compromise between the freedom that layman's creativity requires, and the stability that is needed for this effort to pay off.

The main aim of this research is to create a "stable platform" for the scripters. To obtain this goal is mandatory to define and to support the tuning interface. The definition should be expressed using standards as much as possible and be easy to understand. The support should be and extension of the actual weavers, a *component container*. This extension should take into account the requirements of simplicity and non-disruptive with *JavaScript*. As a secondary aim, it is recommendable to provide help to scripters and webmasters, promoting the adoption of this initiative. Tools as tuning interface *creation/update* and *visualization* are necessary to motivate them.

5 Contribution

Fostering a win-win relationship between website owners and website users, substantiates the efforts from moving away from "fragile scripting" to scalable, robust scripting. To this end, this work introduces *the tuning interface* as an attempt to isolate layman's script from upgrades in the website while abstracting the way scripts are developed. From the owner's viewpoint, this interface realizes a controlled setting for tuning, and fosters the user community that adds value to the site. From the scripter's perspective, the tuning interface reduces the freedom but increases change resilience, and eases coding.

Acknowledgements

This work is co-supported by the Spanish Ministry of Education, and the European Social Fund under contract TIN2008-06507-C02-01/TIN (MODELINE), and the Avanza I+D initiative of the Ministry of Industry, Tourism and Commerce under contract TSI-020100-2008-415. Arellano has a doctoral grant from the Spanish Ministry of Science & Education.

References

1. Greasemonkey Homepage. http://www.greasespot.net/.
2. P. Brusilovsky and M. T. Maybury. From Adaptive Hypermedia to the Adaptive Web. *Communications of the ACM*, 45:30–33, 2002.
3. O. Díaz, S. Pérez, and I. Paz. Providing Personalized Mashups Within the Context of Existing Web Applications. In *International Conference on Web Information Systems Engineering (WISE)*, 2007.
4. R. Ennals and M. Garofalakis. Mashmaker: Mashups for the Masses. In *ACM SIGMOD International Conference on Management of Data*, 2007.
5. K. Lau and Z. Wang. Software Component Models. *IEEE Transactions on Software Engineering*, 33:709–724, 2007.
6. J. Y. Lee, S. H. Lee, and Y. Kim. An experiment on visible changes of web pages. In *Semantic Information Integration on Knowledge Discovery SIIK Workshop*, 2006.
7. S. Lingam and S. Elbaum. Supporting end-users in the creation of dependable web clips. In *International World Wide Web Conference (WWW)*, 2007.
8. J. Raposo, A. Pan, M. Álvarez, and J. Hidalgo. Automatically maintaining wrappers for semi-structured web sources. *Data & Knowledge Engineering*, 61(2):331–358, 2007.
9. B. J. Rhodes. Margin Notes: Building a Contextually Aware Associative Memory. In *International Conference on Intelligent User Interfaces*, 2000.