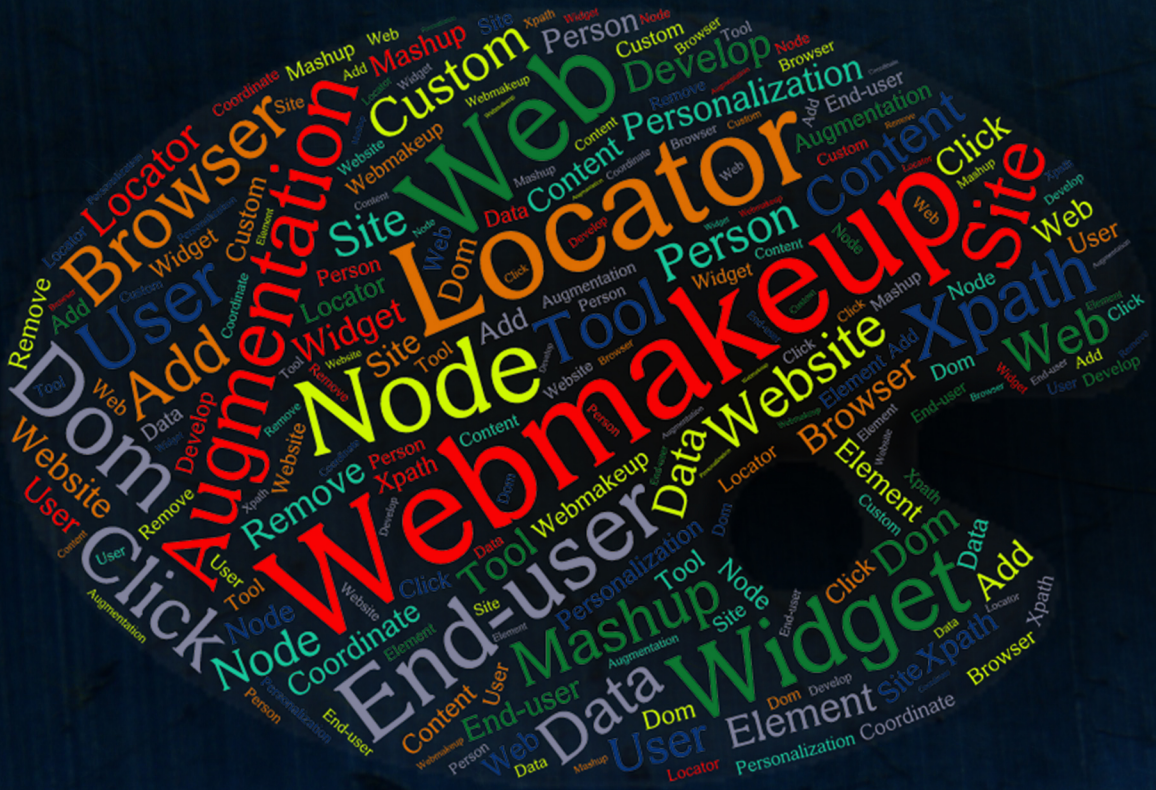

Ph.D. Thesis

Personalizing the Web: A Tool for Empowering End-Users to Customize the Web through Browser-Side Modification

Iñigo Aldalur Ceberio

2017

eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Web Engineering Research Group
Supervisor: Oscar Díaz

Personalizing the Web: A Tool for Empowering End-Users to Customize the Web through Browser-Side Modification

Dissertation

presented to

the Department of Computer Languages and Systems of

the University of the Basque Country in

Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

("*international*" mention)

Iñigo Aldalur Ceberio

Supervisors:

Prof. Dr. Oscar Díaz García

Donostia-San Sebastián, Spain, 2017

This work was hosted by the *University of the Basque Country (Faculty of Computer Sciences)*. The author enjoyed a doctoral grant under de **FPI** (***F**ormación de **P**ersonal **I**nvestigador*) from the *Spanish Ministry of Science & Education* during the years 2012 to 2016. The work was co-supported by the *Spanish Ministry of Education*, and the *European Social Fund* under contract *Scriptongue* (TIN2011-23839).

Summary

Web applications delegate to the browser the final rendering of their pages. This permits browser-based transcoding (a.k.a. *Web Augmentation*) that can be ultimately singularized for each browser installation. This creates an opportunity for Web consumers to customize their Web experiences. This vision requires provisioning adequate tooling that makes *Web Augmentation* affordable to laymen. We consider this a special class of *End-User Development*, integrating *Web Augmentation* paradigms. The dominant paradigm in *End-User Development* is scripting languages through visual languages.

This thesis advocates for a Google Chrome browser extension for *Web Augmentation*. This is carried out through *WebMakeup*, a visual DSL programming tool for end-users to customize their own websites. *WebMakeup* removes, moves and adds web nodes from different web pages in order to avoid tab switching, scrolling, the number of clicks and cutting and pasting. Moreover, *Web Augmentation* extensions has difficulties in finding web elements after a website updating. As a consequence, browser extensions give up working and users might stop using these extensions. This is why two different locators have been implemented with the aim of improving web locator robustness.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

<hr />

1.1 Context

Web Personalization refers to making a web site more responsive to the unique and individual needs of each user [CDA00a]. To achieve this goal, the web application is adapted to the user needs; the web master designs a website where its content/layout/navigation changes depending on the user. The increasing volume of content and actions available on the Web, combined with the growing number of mature digital natives, *anticipate a growing desire of controlling the Web experience*. Often, in order to perform activities conducted through the Web, various websites are needed [KLN05] which all *"live in their own world"*. This leaves *the users themselves in charge of integrating the resources and services required in carrying out their inter-site activities*. Here, the approach is to empower end-users for them to develop such functionality by themselves.

End-User Development can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact [LPW06]. *End-users* are able to start with

simple adaptation mechanisms and gradually advance to more powerful adaptation mechanisms without facing insuperable barriers [SDW08]. *This work looks into Web Augmentation as a mechanisms to adapt Web pages by end-users*.

Web Augmentation is to the web what *Augmented Reality* is to the physical world: layering relevant content/layout/navigation over the existing web to customize the user experience *[DAA13]*. *Web augmentation* techniques have been proposed as a way for extending Web sites features without affecting the server-side code [Bou99] what is more, augmentations are not made by the creator but by the user of the web application. Unfortunately, *Web Augmentation* incurs in important drawbacks. Augmentations are vulnerable to page changes. According to [Fil06], "augmentations must often rely on pattern expressions, but pattern matching can be an easily disrupted technique" and "web page formats evolve", therefore if a web page changes, the augmentation may stop working.

1.2 Problem Statement

Problem statement

- Personalization techniques can not always cater for/foresee individual practices

This problem is provoked by some causes and it has some consequences, which are listed below:

Causes

- **No cost effective:** software companies are not interested in developing personal software due to the more clients you have, the more versions you will implement. The cost of producing these personal products would be unacceptable for companies owing to the expensive cost.

- **No ease to foresee:** No design can provide information for every situation, and no designer can include personalized information for every user [Rho00]. That is why users yearn to customize websites. Despite the existence of applications to automate and personalize web interactions, these are insufficient to guarantee the accomplishment of user goals when changes in relevant context cannot be fully anticipated at design time [CVM14].

- **Significant user effort:** End-users do not have programming knowledge and the effort they need to carry out web modification with programming languages is colossal. Reducing the user's efforts and, therefore, the gulf of execution and the evaluation gulf is the aim of end-user augmentation tools [WCB+15].

Consequences

- **Scrolling:** Some studies claim that users scroll a lot [AGJ+16]. The outcome is that people used the scrollbar on 76% of the pages, with 22% being scrolled all the way to the bottom regardless of the length of the page. Despite these facts, designers "are inundated with requests to cram as much information above the fold as possible, which complicates the information design" [Scra]. As for the former, different studies confirm that users will scroll to find information and items below the fold. Chartbeat, a data analytics provider, analysed data from 2 billion visits and found that "66% of attention on a normal media page is spent below the fold [Scrc]. Jared Spool's usability tests tell us that, even though people say they don't like to scroll, they are willing to do so. Moreover, longer and scrollable pages even worked better for users [Scrb]. Nonetheless, scrolling up and down a page without reading the content may be a signal of frustration and lack of confidence [AKG10].

- **Tab switching:** We now use the Web to multi-task the activities we do every day, to the extent that it is not unusual to see users

with a dozen applications and browser instances open at a time; e.g., sharing pictures, listening to music, or shopping, just to name a few [DB10]. [HW10] found that users switch tabs at least 57.4% of the time, but user activity, measured in page views, is split among tabs rather than increasing overall activity. [AN15] studied that Multiple windows and tabs have significant flaws that hinder users' performance. 70% of participants achieved a 21.37% decrease in the time required to complete a comparison task with our prototype. In addition, they found further advantages in flexibility of movement, learning time, and reduced memory load. Another study showed that navigations using the back button or opening new tabs, which result from considering a user dynamic interaction with the provided contents, clearly allow users to achieve their goals in less time, so increasing productivity. Experimental results have shown that parallel tab browsing behaviour noticeably increases the user's productivity (measured in number of finished sessions) up to 200% with respect to browsing the website in a sequential way [PGSP15]. Finally, [RPR17] defined a browser session as a sequence of request-responses serviced by the browser application pivoting on a single browser window containing exactly one browser tab. When multiple sessions are involved it becomes necessary to understand and represent the information contained in each session, which is essential to differentiate them.

- **A large number of clicks:** it is suggested to be an indicator of problems [AKG10]. The more clicks the user does, the worse interaction is. Some exploratory studies notice that in three exploratory tasks involve a higher number of clicks than the lookup tasks and this is caused by users trend to do unnecessary clicks [AGJ+16]. More to the point, often times we found that users click backwards [SPWL14] to assure themselves they clicked on the right link by reading again its content. It has been documented

as a revisitation strategy [ATD08] and also as way to have a quick preview of a page [KS08].

- **Cut and paste:** [SER09] saw that end users are certainly creatures of habit and perform the majority of their copy and paste tasks within just a few applications. [SER09] also saw that repeating copy paste sequences between the same two windows, and distributing data from one source to multiple destination windows, are the most popular behaviours. One possible solution to cut and paste are hyperlinks which are essential for several reasons because from a human–computer interface perspective, they allow users to explore the available information in a natural way by effortlessly following pointers to references [WPL15].

Scrolling, tab switching or a large number of clicks are the symptoms of a "sickness": websites do not fully match user profiles. To face this issue, this thesis explores the combined used of *Web Augmentation* and *End-User Development*. Specifically, the solution has been to develop a browser extension that allows end-users to customize web pages. A browser extension is a plug-in that extends the functionality of a web browser [Bro]. Websites are not static, sooner or later the owner of the website will change the content, the structure or the style. The more updates a website has, the more possibilities a browser extension will fail. Web extensions for *Web Augmentation* have a significant problem when a web page is updated due to the fact that they might not find the relevant node to carry out the augmentation. Should the extension fail, the user might stop using this extension. It is of the utmost importance to realise that *End-User Development* tools for *Web Augmentation* generates their own locators (a mechanism for uniquely identifying an element on the web page [RLS+13]) automatically because end-users do not know how to create locators. That is why, with a shadow of a doubt, it is highly important to develop the most robust locator system.

Problem statement

- Web extensions stop working

The causes and consequences of this problem are:

Causes

- **Locator fragility:** Different articles compare the robustness of each type of locator [LCRT14, BMM12, LSRT16] and their fragility to face web site upgrades [FB11b, DBS09].
- **Frequent website upgrades:** Web pages are frequently updated because they were not able to meet users' needs [LCRT13]. Structural changes are quite important, since web site re-styling, a frequently occurring activity, tends to affect the DOM structure, leaving the application logic unaffected [LSRT14].
- **Web extensions are not often developed by website owners:** this prevents developers from noticing when a website has evolve and hence, [KKA06] shortcomings of the proposed approach is that extensions had to be explicitly prepared by website owners.

Consequences

- **Failures might make users forsake the extension:** Locator fragility impacts on maintenance. If augmentations no longer recover the right DOM node, augmentations need to be redone and meanwhile, if the extension does not work, the user may think that the extension is dreadful and he will not continue using it.
- **Maintenance cost:** Frequently modifications applied on a web application lead to have one or more locators broken and repairing the Web extension is a time-consuming and expensive task [LCRS13].

1.3 This Dissertation

In this dissertation the following three issues are tackled:

Review about End-User Development features for Web Augmentation and Mashups

- Problem statement: Different studies have been carried out related with *End-User Development* tools for different aspects. Nonetheless, there are not studies based on *End-User Development* tools for *Web Augmentation*.

- Solution: A survey on *End-User Development* for *Web Augmentation* has been accomplished taking into account the most important conferences in *End-User Development.*

Empowering people to customize web content

- Problem statement: End-users desire to reorganize a website for better information extraction. End-users yearn for web customization to delete unnecessary information, add information from different websites to complete the target site, move content to avoid scrolling or better information extraction, to avoid opening new tabs, cutting and pasting, URL typing and also to decrease the number of clicks. As a consequence, the users will be able to better concentrate on their task.

- Solution: An *End-User Development* tool has been developed to facilitate web customization by end-users. The aim is to abstract technical details into a visual Domain Specific Language that facilitates end-user implication. To accomplish this objective, a Google Chrome browser extension called *WebMakeup* has been developed.

Enhancing web locator robustness

- Problem statement: After websites upgrades, previously desired web nodes will no longer be found by standard locators. How can I enhance locator robustness in order for them to be able to find the desired nodes for a longer time even after website changes?

- Solution: Two different algorithms have been proposed, *Kidney locator* and *Regenerative locator*. *Kidney locators* are based on redundancy that utilizes different locator techniques to find a web node and regenerates the information needed in those locator techniques that have not been able to find the desired node. *Regenerative locators* employ all attribute information about the target node and all its ancestors to try to regenerate a valid XPath that permits the correct node location after web upgrades.

1.4 Research approach

Design science is the scientific study and creation of artefacts as they are developed and used by people with the goal of solving practical problems of general interest [JP14]. Thus, design science is one approach to investigating artefacts. Design science takes a problem solving stance, starting from problems experienced by people in practices and then tries to solve them. It does so by creating, positioning, and repurposing artefacts that can function as solutions to the problems. Design science is viewed mainly from an IT and information systems perspective. However, the principles underlying design science are applicable to many other areas [JP14].

There are some variations depending on the author of the Design Science Research (DSR) proposal. Paul Johannesson and Erik Perjons summarised DSR [JP14] proposals and illustrated a methodology fulfilling all previous authors' requirements in figure 1.1.

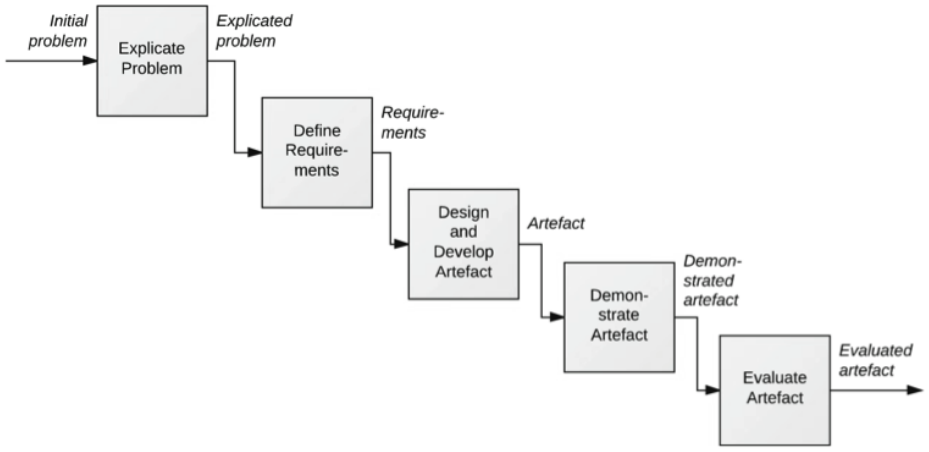

Figure 1.1: DSR methodology process model [JP14]

- Explicate Problem. The goal of this activity is to investigate and analyse a practical problem. The problem must be precisely formulated and justified by showing that it is relevant. The problem has to be of general interest and remarking causes to the problem it might be identified and analysed.

- Define Requirements. The Define Requirements activity outlines a solution to the explicated problem in the form of an artefact and elicits requirements, which can be seen as a transformation of the problem into demands on the proposed artefact.

- Design and Develop Artefact. The Design and Develop Artefact activity creates an artefact that addresses the explicated problem and fulfils the defined requirements. Designing an artefact includes determining its functionality as well as its structure.

- Demonstrate Artefact. The Demonstrate Artefact activity uses the developed artefact in an illustrative or real-life case, sometimes called a “proof of concept”, thereby proving the feasibility of the artefact. The demonstration will show that the artefact actually can solve an instance of the problem.

- Evaluate Artefact. The Evaluate Artefact activity determines how well the artefact fulfils the requirements and to what extent it can solve, or alleviate, the practical problem that motivated the research.

As indicated by [JP14], these tasks do not follow strictly in sequence. Rather, research is commonly iterative, moving back and forth between all the activities of problem explication, requirements definition, development, and evaluation. The arrows in Figure 1.1 should not be interpreted as temporal orderings but as input–output relationships. In other words, the activities should not be seen as temporally ordered but instead as logically related through input–output relationships.

This dissertation has been developed along DSR hallmarks. First, the problem was detected and an investigation and analysis was conducted to define the problem properly. Next, we decided what were the objectives and how they will be solved. Once the objectives were clear, *WebMakeup* (3) was designed and implemented. This design and implementation was updated several times after evaluations. First evaluations were carried out with degree and master students. In the last evaluation, end-users evaluate *WebMakeup* with a satisfactory result. The evaluation in chapter 4 was accomplished at the beginning with different websites and examples of *WebMakeup* and finally with other browser extensions.

1.5 Outline

The content of each chapter is summarized in this section. Figure 1.2 contains a map that illustrates the relationships between the chapters of this dissertation.

Chapter 2

This chapter presents a study of *End-User Development* tools for the Web using techniques of *Web Augmentation*. This study sets the bases for the rest of this thesis.

Chapter 3

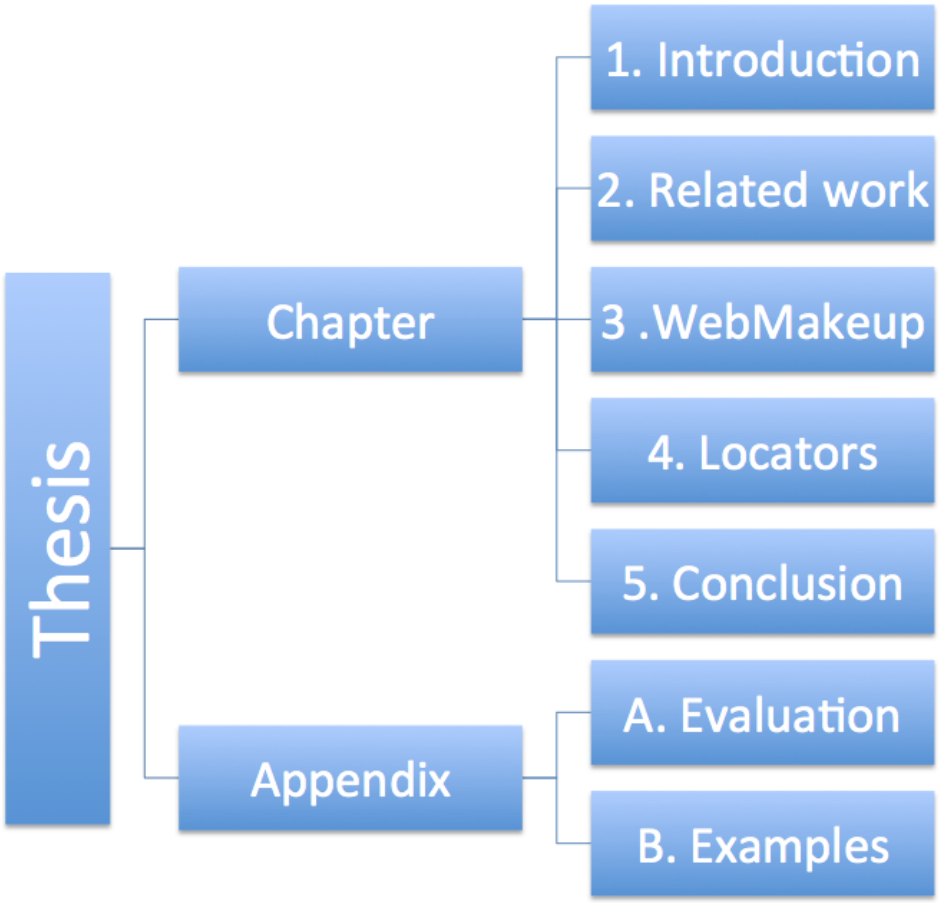

Figure 1.2: Chapter map of the dissertation.

This chapter describes an approach for end-user client-based augmentation of websites. To this end, a visual programming tool is presented, *WebMakeup*. *WebMakeup* permits end-users to generate components of their interest from different websites with the intention of customizing their favourite web pages. End-users are able to add these components, called *widgets*, remove elements from the customized websites and even move them to different locations.

Chapter 4

This chapter shows different techniques to locate DOM nodes. Moreover, it proposes two different techniques to enhance *locator*

robustness based on redundancy, *Kidney locator* and *Regenerative locator*.

Chapter 5

This chapter concludes the dissertation by remarking the main results, listing publications of the author's thesis, enumerating the limitations of the current solutions and proposing future work.

Appendix A

This appendix presents the exercise made by end-users to evaluate *WebMakeup* and it presents the questionnaire created for the experiment.

Appendix B

This appendix shows some examples that check the spectrum of possibilities that *WebMakeup* proposes.

1.6 Conclusion

In this chapter, the definition and concepts of *Web Augmentation* for web customization are introduced as the main topic of this dissertation. For such topic, three contributions are identified: First, a study of *Web Augmentation* tools for *End-User Development.* Second, a visual programming browser extension for *Web Augmentation*. Finally, an approach to enhance web node location for web extensions. The next chapter provides the necessary background to understand the rest of the chapters.

Chapter 2

Related work

<hr />

2.1 Introduction

Nowadays, many applications which, formerly, would have been designed for the desktop such as calendars, travel reservation systems, purchasing systems, library card catalogs, maps viewers or even games have made the transition to the Web, largely successfully. Many Web sites are created every day to help users to find information and/or to provide services they need. However, there are cases where rather than a new Web site, what users need is to combine information or services that are already available but scattered on the WWW. Some examples follow: (1) users who want to have additional links on a Web page to improve the navigation (for example to create a personalized menu that gathers in one location multiple personal interests), (2) users who need to integrate contents from diverse Web sites (for example to include a Google's map into a Web page that originally only shows addresses as flat text) in order to improve their performance in identifying distance from their personal location or (3) simply to remove content from Web pages (such as contact details they consider irrelevant) to improve reading and selection performance as

identified by Hick's law [Hic52]. Because these needs might be perceived as idiosyncratic, volatile (being short-lived or occasional) or dissenting with the interests of the Web site, they might well not be considered (or even not known) by Web developers [FURS16]. This is because Web sites are, by definition, designed for the masses and that at design time only few users are available.

Previous work on *End-User Development* [IAD14, LPW06] has demonstrated that, if appropriate tools are provided, end-users might be able to create what they need (or at least define more precisely part of what they need). DENIM is a pioneer example that illustrates how tools can be used for involving users into the design of the Web sites to be developed [NLHL03]. A more demanding scenario is when the target is not in-home Web sites but Web pages that have already been created by third parties. The options are here, either to redevelop what has already been done by the third party or to try to convince the third party to tune its development to fit a particular user need. This deeply collides with the principle of Web development that target the masses and not the individual. The term *Web Augmentation* is used to describe tools that can be used to improve (hence the word "augment") existing Web pages (found for instance whilst browsing the Web) to create better fit user's needs and activities. Some of the most popular *Web Augmentation* tools work by extending the functionalities of the Web browser used by the user via plug-ins that can run client-side scripts to manipulate the structure of Web pages loaded in the browser. In that case the augmentation will be applied to all the visited Web page featuring specific characteristics. The potential of *Web Augmentation* techniques can be illustrated by some advanced applications such as lightweight integration of information extracted from the Web, context-sensitive navigation across diverse Web site, context-dependent multimodal adaptation [GMPP14] or refactoring Web sites for accessibility [FWRG11b]. Another example is a spellchecking plug-in that would automatically check the text entered by the user on any Web page. The degree of expertise required for using *Web Augmentation* tools

varies dramatically [HT10]. For example, some tools only require basic knowledge of how to install plug-ins in the Web browsers while others may require integrating sophisticated scripting code created by the user.

In this chapter, we examine the potential of *Web Augmentation* technology for supporting *End-User Development* for the Web. In section 2.2, we discuss the relationship between *Web Augmentation* and *End-User Development*. Section 2.3, proposes a classification of *Web Augmentation* technologies, positions existing tools with respect to this classification and provides a study of research contributions for each main category of the classification. In section 2.4, we explain some of the users and usage difficulties specific to the adaptation of Web applications. Finally, section 2.5 concludes the chapter.

2.2 Web Augmentation and End-User Development

Web Augmentation (WA) is not *End-User Development* (EUD) for the Web but some of the features provided by WA tools can be used for that purpose. To highlight similarities and differences, we revisit their definitions.

Many authors have tried to define precisely the term end-user programming [BS11, LPW06]. In this chapter, we adhere to the definition provided by Ko et al. [KAB+11] who state that "*end-user programming is programming to achieve the result of program primarily for personal, rather than public use*". That definition has many implications. First, it is important to note the absence of any reference to an application domain and/or technology highlighting the large scope for the use of EUD tools. Next, the term "programming" refers to a general activity, which might encompass the development of software from scratch and/or making modification to an existing software. Finally, the term "end-user" does not refer to the user's skills in so for as a professional developer is engaged in end-user programming when writing code to fulfill a personal need, such

as visualize the data structure to help diagnose a bug. Moreover, even if the definition implies a particular intention behind the development of the program, it does not exclude the possibility of sharing the program with other users.

There are fewer attempts to define precisely the term *Web Augmentation*. This term was originally coined by Bouvin in 1999 [Bou99] to describe a tool that "*through integration with a Web browser, a HTTP proxy or a Web server adds content or controls not contained within the Web pages themselves to the effect of allowing structure to be added to the Web page directly or indirectly, or to navigate such structure. The purpose of such a tool is help users organize, associate, or structure information found on the Web. This activity may be done by a single user or in collaboration with others*". More recently, Díaz [DA15] said that "*WA is to the Web what Augmented Reality is to the physical world: layering relevant content/layout/navigation over the existing Web to customize the user experience*". These definitions highlight WA as a non-intrusive approach: augmentations are "layers" on top of an existing content. These augmentation layers might be needed to cater for situational and idiosyncratic needs, difficult for designers to foresee. Technically, augmentations do not need the participation of the Web sites used for the augmentation since the augmentation occurs on the Web browser. *Web augmentation* technology only acts on the user interaction and does not change the original Web page stored on the Web server. It is interesting to note that whilst Bouvin does not assign any particular intention for the use of WA tools, Díaz explicitly mentions that augmentation layers might aim at improving the user experience with the Web page.

For our purposes, WA describes tools that allow people to modify Web pages to improve user performance and satisfaction. This definition connects WA to EUD as EUD "*is programming to achieve the result of program primarily for personal, rather than public use*". Indeed, WA realizes this vision in the web sphere as far as it helps to support users' needs that have not been originally been identified or taken into account

during the design of the Web site.

2.3 End-User Development tools for the Web

The evolution of Web technology is changing the way users interact with Web sites. At first, users could only consume contents provided by Web sites. Later, users could actively contribute with content by using tools such as CMS (Content Management Systems) and wikis. More recently, WA tools empower people in different ways making these tools real EUD tools: (1) to create their own web sites, (2) to combine information from diverse Web sites into a single hub (using mashups), and even (3) to modify Web pages created by others (using WA tools e.g. MADCOW [BCL+04] and DiLAS [AAF+05]). This highlights the broad range of approaches that Web-centered EUD tools explore. Figure 2.1 introduces a set of dimensions to classify these tools while the positioning of existing tools with respect to this classification is shown in Table 2.1.

Mashup technology is an interesting alternative for final users to combine existing resources and services in a new Web application [AP11]. Mashups are often very specialized and only operate with specific types of contents (quite often structured data sources). For example, FaceMashup [MS15] is a EUD tool for mashup that allows users to manipulate social network APIs to combine data and sharing them with other users through the social networks. It is interesting to notice that some WA tools such as CSN Framework [FWRG11b] borrow from mashups the ability to integrate contents but they are even more flexible allowing to compose any kind of DOM element from a Web page.

Tool wise, Figure 2.2 highlights how mashups (66%) have received more attention throughout w.r.t. WA tools (34%). This seems to suggest that integrating different data sources is being considered more important than customizing existing Web sites. Though this might be true in a general sense, when it comes to empowering end-users, data integration might be more costly and hence, more difficult to end users to achieve. By contrast,

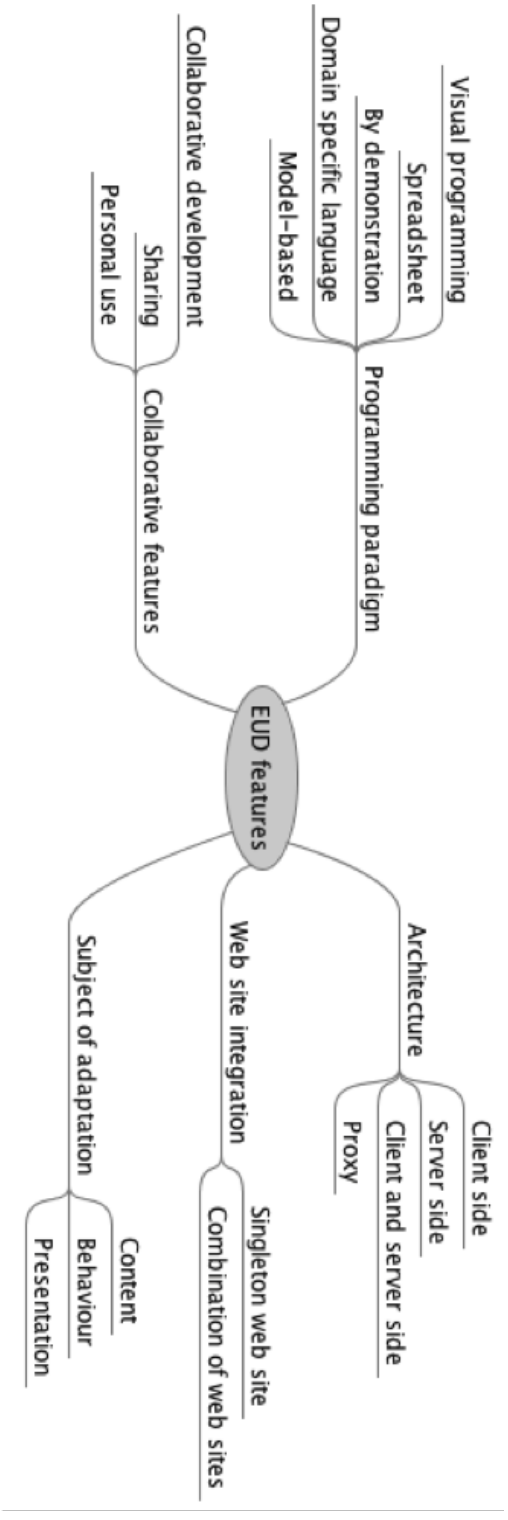

Figure 2.1: Features of *End-User Development* tools for Web applications

Tools	Year	Type	Execution mode			Subject of adaptation			Web site Integration	Collaboration features	Programing Paradigm	Ref.
			C	S	P	Co	Be	Pr				
Marmite	2007	M	C			Co			Combination	Personal use	Visual program.	[WH07]
MARGMASH	2007	**WA**	C			Co			Combination	Personal use	By demonstration	[DPP07]
CoScripter	2008	M	C			Co	Be		Singleton	Collaborative dev.	By demonstration	[LHML08]
Reform	2009	**WA**	C			Co			Combination	Personal use	By demonstration	[TDD+09]
SemanticWebPipes	2009	M		S		Co			Combination	Collaborative use	Visual program.	[PPH+09]
Mashroom	2009	M	C			Co			Combination	Personal use	Spreadsheets	[WYH09]
Deep	2010	M	C			Co		Pr	Combination	Personal use	By demonstration	[GHT10]
MashSheet	2010	M	C			Co			Combination	Collaborative dev.	Spreadsheets	[HPB10, HPD11]
Atomate	2010	M	C			Co			Combination	Collaborative dev.	Model-based	[KMK+10]
RUMU	2010	**WA**		S		Co		Pr	Singleton	Personal use	Visual program.	[Pol10]
CSN framework	2011	**WA**	C			Co	Be		Combination	Collaborative use	By demonstration	[FWRG11b]
OntoCompo	2011	M	C			Co	Be		Singleton	Personal use	Model-based	[BDRR11]
Mixer	2011	**WA**	C			Co			Combination	Collaborative use	By demonstration	[GTZ+11]
IVO	2011	M	C	S		Co	Be		Singleton	Collaborative use	By demonstration	[RDR11]
Mashupeditor	2011	M			P	Co			Combination	Collaborative use	By demonstration	[GPS11][GPSP16]
DashMash	2011	M	C/S			Co	Be		Combination	Personal use	Visual program.	[CDM+11,CMP+11]
MAIDL	2011	M	C/S			Co			Combination	Personal use	By demonstration	[CPT11]
VisPro	2011	M	C/S			Co	Be		Combination	Personal use	Visual program.	[BMM+11]
SOA4All Studio	2011	M	C/S			Co	Be		Combination	Collaborative use	Visual program.	[WNM11]
Cowpath	2012	**WA**	C				Be		Combination	Collaborative use	DSL	[DSAT12]
Webcrystal	2012	**WA**	C			Co		Pr	Combination	Personal use	By demonstration	[CM12]
Baya	2012	M	C			Co			Combination	Collaborative use	Visual program.	[CRDC12, DRC+12]
ResEval Mash	2012	M	C/S			Co			Combination	Collaborative use	Visual program.	[ISK+12]
CrowdDesign	2012	M	C/S			Co	Be		Combination	Collaborative use	Visual program.	[NLN12]
Chudnoskyy et al.	2012	M	C			Co			Combination	Collaborative use	Visual program.	[CNG+12]
MOWA	2013	**WA**	C			Co			Combination	Collaborative use	Model-based	[CFB+13]
Sticklet	2013	**WA**	C			Co			Combination	Collaborative dev.	DSL	[AD13, DAA13]
Social Overlays	2013	**WA**	C			Co		Pr	Singleton	Collaborative use	Visual program.	[DANP13]
OpenHTML	2013	**WA**		S		Co		Pr	Singleton	Collaborative dev.	By demonstration	[PSJ+13]
Ardito et al. (a)	2013	M		S		Co		Pr	Combination	Collaborative use	Visual program.	[ABC+13]
MobiMash	2013	M		S		Co	Be		Combination	Personal use	Visual program.	[CMP13]
DireWolf	2013	M		S		Co	Be		Combination	Collaborative dev.	Visual program.	[KRNK13]
Rana et al.	2013	M		S		Co			Combination	Personal use	Visual program.	[RMS13]
CapView	2013	M		S		Co	Be		Combination	Personal use	Visual program.	[RBM13]
WebMakeup	2014	**WA**	C			Co	Be	Pr	Combination	Collaborative use	Visual program.	[DAA+14]
CrowdMock	2014	**WA**	C			Co	Be		Combination	Collaborative dev.	Visual program.	[FFRA14]
Ardito et al. (b)	2014	M		S		Co			Combination	Collaborative use	Visual program.	[ACD+14, ACD+15]
MultiMasher	2014	M		S		Co			Combination	Collaborative use	Visual program.	[HNPN14]
NaturalMash	2014	M	C	S		Co			Combination	Collaborative use	By demonstration	[AP14]
SmartComposition	2014	M	C	S		Co			Combination	Collaborative use	Model-based	[KWG14]
Tayeh et al.	2014	**WA**	C			Co			Singleton	Personal use	Visual program.	[TS14, TS15]
FaceMashup	2015	M		S		Co			Singleton	Personal use	Visual program.	[MS15]
IWC	2015	M		S		Co	Be		Combination	Collaborative use	By demonstration	[NK15]
MAMSAAS	2015	M		S		Co			Combination	Collaborative use	Visual program.	[WW15]
EasyApp	2016	M	C	S		Co	Be		Combination	Personal use	Visual program.	[ZCW+16]
WOA	2016	**WA**	C			Co	Be	Pr	Combination	Collaborative dev.	By demonstration	[FBR+16]
Miján et al.	2016	**WA**	C			Co	Be		Singleton	Collaborative use	Visual program.	[MGF16]

Legend: M: Mashup, **WA**: Web augmentation | C: client side, S: server side, C/S: both client and server sides, P: proxy | Co: content, Be: behavior, Pr: presentation | DSL: domain specific language|

Table 2.1: Classification of *End-User Development* tools for the Web

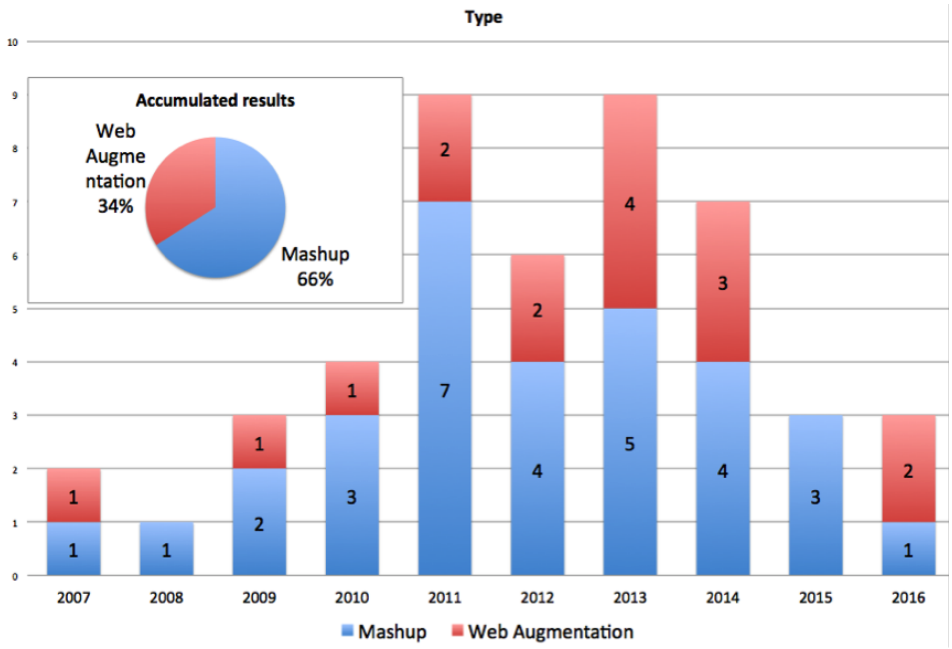

Figure 2.2: Contributions presenting tools: Mashup versus WA technology

WA is not so demanding, and hence more affordable to end-users. This makes WA tools more likely to be adopted by end users.

The rest of this section explains the classification presented in Figure 2.1 and provides examples of the corresponding Web technology.

2.3.1 Architecture

Tools might rest on the client side, the server side or both. Client-side tools are executed as Web browsers' extensions (or plug-ins) and processing happens on the user's local computer. Common programming languages used to implement client-side applications include HTML, CSS, and Javascript. Conversely, server side technology runs on a remote machine, and only the outcome of the execution returns to the user's local computer. Common programming languages include Ruby, Python, PHP, C#... Server side technologies can store persistent data. However, data can

only be accessed than through HTTP requests for a particular URL.

Miján et al. [MGF16] and WebCrystal [CM12] illustrate the client-side approach. WebCrystal is a Firefox plug-in that allows the inspection of code corresponding to visual objects. WebCrystal provides users feedback using a textual description and a customized code snippet that can be copied-and-pasted to rebuild the user-selected properties. Additionally, Miján et al. resort to a set of personalization rules to be applied in the client-side with minimum alterations defined without requiring either advanced programming skills or advanced configuration.

Whilst Web browsers can store data in the local cache, server-side technology is used by many tools such as DireWolf [KRNK13], FaceMashup [MS15], Ardito et al. [ACD+14, ACD+15] and MultiMasher [HNPN14] as a means to support data persistence. DireWolf provides several extensible components for adapting Web sites and it implements a service for data persistence such as user device profiles and shared application states.

As for client-server tools, most requests a kept in the client with sporadic calls to the server. For example, DashMash [CDM+11, CMP+11] has a client-side module for mashup creation and a server module responsible for integrating and storing data from different types of services. In the mobile world, IVO [RDR11] follows a similar architecture. For mashups, MashupEditor [GPS11, GPSP16] allows for adaptations to be created on the client (using a dedicated editor). Next, a proxy server store those adaptations that can be later reused during the creation of the mashup.

From the accumulated results in Figure 2.3, it is clear that the client-side approach is the most popular architecture (49%). The Client-server option (21%) boosted in 2011, presumably due to the popularity of the Web 2.0 and the focus on sharing and the need to have common repositories. The server-side option (28%) rose from 2013 onwards, arguably on the search for a business model for mashup platforms.

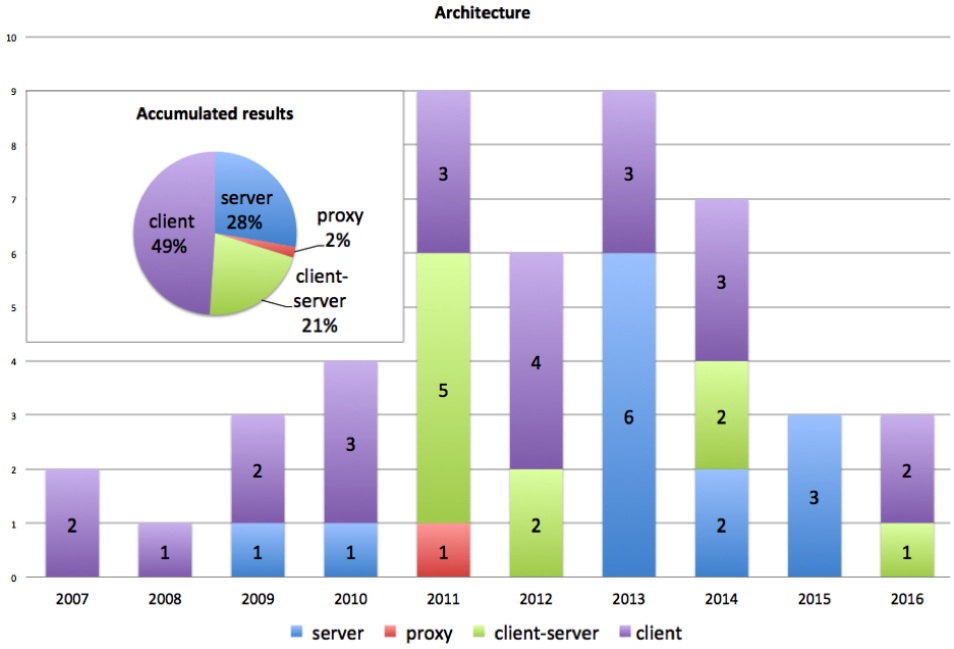

Figure 2.3: Distribution over the years of tools and what part of the Web architectures they were exploiting

2.3.2 Subject of adaptation

Web sites might be adapted in different ways: including brand-new content, changing the behavior associated to DOM elements or altering the appearance (style and layout). Most tools provide functions to add/remove/replace contents. Adding content from other sources is often used as a means for making information readily available whilst removing content is useful to improve focus, preventing users from distraction. Mixer [GTZ+11] resorts to WA to improve the organization of Web pages simply by letting users to move contents around and include/exclude contents needed. Mashups are also used to add content from different websites. SmartComposition [KWG14] is another content-based approach that is primarily used to build mashups but it also features unique functions that allow to reorganize contents to fit into different screen sizes. Chudnoskyy et al. [CNG+12] take a step forward by assisting users with

recommendations and automatic composition.

Whilst modifying CSS code (color, font, etc.) is relatively simple, few tools account for this kind of adaptation. RUMU [Pol10] is a web-based WYSIWYG editor that resorts to a semantic language to change the page style and simplifying web design. OpenHTML [PSJ+13] is also a web editor to introduce laymen into HTML and CSS.

Finally, changing the behavior of Web sites is far from trivial. It often requires adding some Javascript code to DOM elements like show or hide web nodes, click on certain button, change the content of an element, etc. Changing the behavior of web sites might be necessary, for example, for automating repetitive tasks. Inter-Widget Communication (IWC) [NK15] is a semi-automatic, end-user friendly approach to extend widgets employing the programming-by-demonstration paradigm. IWC is built by composing interactive widgets. IWC leaves users with the tedious task of manual wiring widgets to create mashups. SOA4All [WNM11] is a visual development environment that addresses adaptation of Web applications through the connection of different service components into an assembly line.

2.3.3 Web site integration

This dimension tells if users work with one (singleton) or more (combination of) Web sites in a single project. Whilst many EUD tools are designed to augment a particular type of singleton Web site (e.g. OpenHTML), some tools allow to mix content from diverse Web sites.

Mashups tools like Baya [CRDC12, DRC+12], Deep [GHT10], MamSaas [WW15] and Marmite [WH07] are typical examples of tools that allow to extract data from different Web sites and recombine them in a form that better fulfill user's needs. Nonetheless, other strategies combine Web sites that don't necessarily involve structured data sets. For example, Ardito et al. [ABC+13] is a platform for end-users to compose personal information spaces by assembling pieces of information from different

sources. Such personal information spaces can be enacted in different devices and shared with other users. MamSaas [WW15] is a layered architecture to deploy and identify mashup components as well as link and execute mashups for quick application development. MOWA [CFB+13] is another EUD tool for WA that enables end-users to create a custom guided tour of a city based on contents collected from diverse Web sites. Its aim is to augment existing Web applications with mobile features. Using MOWA end-users can pinpoint in a map content from a different Web site and then generate a custom script. This mobile Web application prompts the users add points of their interests while they move around the city.

Finally, CrowdDesign [NLN12] can also be classified as a EUD tool in so far as it supports mashup based on the integration of scripts coming from diverse sources. CrowdDesign works as a storage for scripts and user interface components shared by a community of developers. CrowdDesign also features a visual authoring environment that allows users to combine contents and scripts available at the platform to create a more personal version of Web sites.

2.3.4 Collaborative features

Whilst a WA strategy can be adopted only for personal purposes, sharing is an important aspect of *End-User Development* [LPW06, RAR+11]. We distinguish between sharing and collaborative development.

Sharing. Some tools focus on personal use, i.e. results cannot be reused and/or shared with other users. Tayeh et al. [TS14, TS15] is a case in point. These authors provide a tool for the linking and the integration of arbitrary documents and multimedia content dynamically. Rana et al. [RMS13] and EasyApp [ZCW+16] are also tools for personal use. Both tools provide a systematic way of designing, developing and deploying personalized apps. Reform [TDD+09] is a Firefox extension that contributes with architecture for web enhancement that allows end-users to integrate existing enhancements with new websites. Despite

the fact that it allows end-users to communicate with developers for requesting new features, they do not allow sharing developments. CapView [RBM13] is a mashup platform that provides instant feedback for user development actions. CapView helps non-programmers form components with recommendations provided by the system and it manipulates a mashup through visually composing component features.

Moving away for the personal realm, Social Overlays [DANP13] and the CSN framework [FWRG11b] illustrate the use of repositories for script sharing. Social Overlays focuses on repairing either the behavior or the appearance of Web sites. Updates made by individuals are visible to the community which use a voting mechanism to decide if the updates are relevant and if so, be incorporated as part of the Web site offerings. CSN features a plug-in that allows users to adapt Web pages by triggering different types of scripts. It has different features depending on the user profile: developer or end-user. Developers can write new augmentation scripts to extend the set of original sets of scripts available in the framework. Such scripts can then be obtained by other users who on their turn can execute them to adapt the Web sites. Finally, it is interesting to notice that a few tools allow to publish the code in social networks (e.g. Sticklet [DA15, AD13]) whilst others allow to export files for personal use on an individual basis (e.g. WebMakeup).

Collaborative development. CrowdMock [FFRA14] does not provide a voting mechanism but it permits to amend/complete augmentation script by people other than the author. CoScripter [LHML08] resorts to programming by demonstration to enable users to record all the information related to user interaction to edit a Website. The outcome is a script macro that can be automatically stored in a Web server from where they can be delivered to other users and they can use a collaborative scripting environment for recording, automating, and sharing web-based processes.

Figure 2.4 helps to apprehend differences and similitudes between WA and mashups as for “collaboration features” and “subject of adaptation”

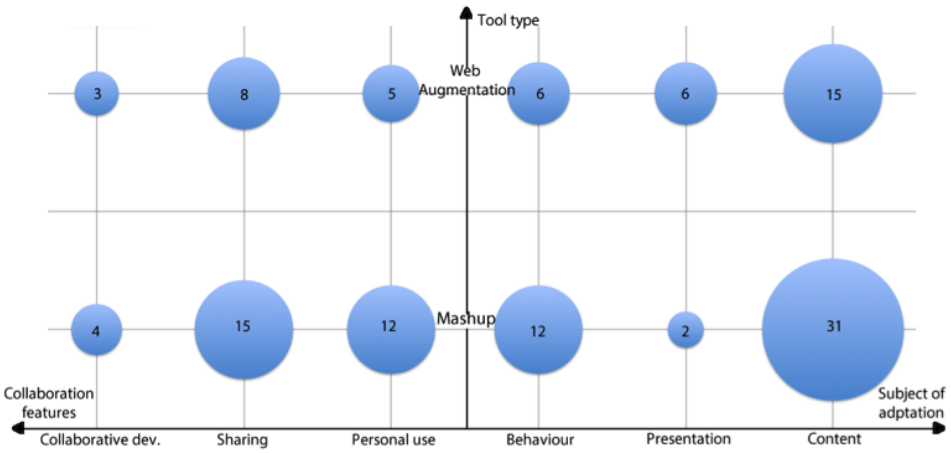

Figure 2.4: Mapping WA tools and Mashups across "Collaboration features" and "Subject of adaptation"

support. As for the former, both scenarios (i.e. WA and mashups) pay attention to the idiosyncratic scenario ("Personal use"), while the potential of reuse (i.e. sharing) is felt to be more intensive for mashups than for WA developments. Also, mashups and WA coincide in their interest in handling content (31 vs. 15) while WA underscores in addressing presentation concerns (2 vs. 6). This is according to expectations since WA adapts existing web sites whose presentation might need to be tuned to better meet users' needs. By contrast, behavior modification has received more attention in the mashup realm.

2.3.5 Programming paradigm

EUD tools resort to diverse programming paradigm: visual languages, spreadsheets, programming by demonstration, domain specific languages (DSL) and model-based automation [AP11].

Visual programming is mainly found in mashup tools that allow drag-and-drop to connect components to create a mashup. Examples include VisPro [BMM+11], ResEval Mash [ISK+12], MobiMash [CMP13], SemanticWeb Pipes [PPH+09] and WebMakeup [DAA+14]. VisPro

creates mashups by dragging and dropping widgets from a library. ResEval Mash is a domain-specific mashup tool that explores dedicated mashuping, in this case in the domain of research evaluation. MobiMash resorts to visual notations to create mobile mashups. The particularity of SemanticWeb Pipes is to blend mashups and the Semantic Web. Here, ontologies are used for better matching widgets parameters that build up the mashup. WebMakeup is an editor that delivers Chrome plugs-in for augmentation purposes. A DSL is defined that sets the expressiveness of the augmentation. WebMakeup helps construct DSL expressions on top of the page being augmented. Once constructed, WebMakeup generates and installs the corresponding Chrome extension.

Programming by demonstration is most popular for data extraction and visualization, where service composition and orchestration play an ancillary role. NaturalMash [AP14], WOA [FBR+16], Margmash [DPP07] and MAIDL [CPT11] illustrate this approach. NaturalMash is a WYSIWYG mashup tool. NaturalMash stands out for its formative support where the tool is able to collect user feedback. WOA enables users to create/extract Web contents in the form of objects that they can manipulate to create Personal Web experiences. Margmash creates augmentations out of personalized information, which are gathered from diverse Web sites. Margmash behaves as a lightweight wrapper that guides end-users on both data gathering and data recombination. MAIDL permits the rapid creation of mobile mashup out of components.

Model-based Automation is concerned with the automatic creation of mashups out of knowledge about the user and the context of use. This technique's weakness is the risk of generating irrelevant mashups w.r.t. the given requirements. Ontocompo [BDRR11] and Atomate [KMK+10] illustrate this approach. Ontocompo makes use of an ontology to generate new applications based on existing ones. Atomate is a personal information assistant engine that automatically carries out tasks for the user. Atomate combine RSS/ATOM feeds from social networking into a simple RDF model representing people, places and things.

DSLs strive to abstract from general-purpose programming language. The challenge here is to find a compromise between expressiveness and learnability. DSLs in the augmentation realm can be illustrated by Cowpath [DSAT12] and Sticklet [AD13, DAA13]. Cowpath focuses on "Web trails", i.e. recurring navigation paths across distinct Web sites. Rather than switching between tabs and typing once and again the same URLs, Cowpath augments the affected websites with additional hyperlinks that "pave the way" of these Web trails. On the other hand, Sticklet explores the use of a dedicated assistant that help users to come with Sticklet expressions to augment Web sites.

Spreadsheets-like programming are often considered ease-of-use, intuitive and with enough expressive power to represent and manage complex data. When it comes to mashups, Mashroom [WYH09] and MashSheet [HPB10, HPD11] explore this approach. Mashroom builds Web applications by combining content coming from different Web sites. To this end, it resorts to an expressive data structure and a set of defined mashup operators. The data structure allows users to express complex data objects while mashup operators are visualized in the formula bar. MashSheet extends conventional spreadsheet paradigms to facilitate Web services "mashup" in a spreadsheet environment. MashSheet is a collection of operators that supports orchestrating Web services, manipulating and visualizing data created by the services.

Figure 2.5 depicts the distribution of research contributions with respect to the "programming paradigm" feature over the years. Visual programming is by far the most popular approach (53%), where the other approaches fall behind: programming by demonstration (30%), Model-based (9%), DSL (4%) and spreadsheets (4%). Worth mentioning, the boost of programming-by-demonstration in 2011 although it faded over the years.

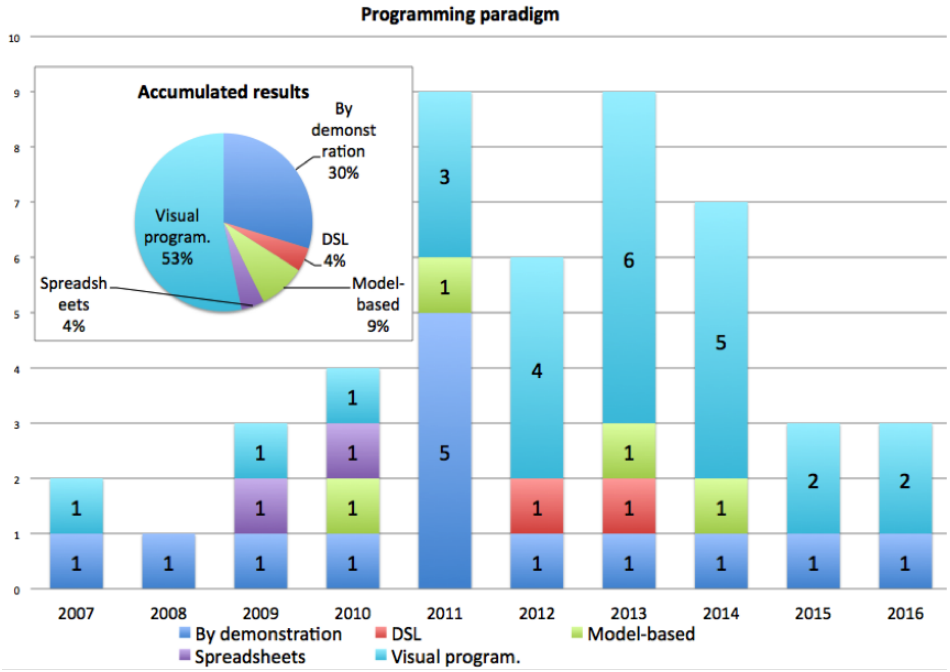

Figure 2.5: "Programming paradigm" in research contributions over the years

2.4 User and usage challenges with WA tools

Each tool cited in this chapter has its own idiosyncrasies and their use will reveal very specific challenges. But beyond the use of a particular tool, WA challenges users to revise what they know about the web and how to program applications. When it comes to WA, users should be aware of a number of aspects, namely:

- WA is mainly a browser-based technology. Regardless of the technology employed to store and run the augmentation scripts, the adaptation only affects how a web site is displayed in the user's personal machine. Users must understand that their adaptation is personal and that will not be visible by other visitors of the same web site.

- WA is mainly a single browse technology. Changes performed by the

user will only occur on the browser where the augmentation has been performed. The same user performing the same actions on another computer will not see the augmentation. It thus requires replication of the augmentation multiple times if the users are using multiple execution platforms (e.g. desktop computers, smartphones).

- Similar to other EUD technologies, WA require the adaption of the code produced by someone else. This has multiple implications for assessing the code of Web pages before to adapt them [GPSP16, GYK11].

- WA is constricted within the DOM hierarchy. Users should be aware of manipulation of DOM elements imposes a certain order of access to contents. For instance, elements might appear visually together but be arranged in separated DOM nodes. This might imply having different ancestors. This, in turn, prevents these "alongside elements" from being selected as a single DOM element. This constrain is imposed by the DOM element hierarchy [BFR+16]. Notice that the DOM hierarchy itself does not need to be made visible but manipulated through metaphors and witty interactive tools. But no matter the tool, it is constricted within the DOM hierarchy.

- WA is fragile upon Web-site upgrades. Web sites evolve overtime and with the evolution of a web site some elements resulting from the augmentations may disappear and/or be replaced by other elements that directly affect the way WA scripts operate. Thus, whilst some scripts will be resilient to maintenance of web sites, other scripts will stop working once a Web site is upgraded. This makes the use of WA a more suitable technique when user's needs are volatile [FURS16].

- WA does not create brand-new applications but enhances existing ones. The inclusion of contents from other web sites raises some pragmatic questions about the type of relationship created between

web sites [FFR+15]. The simplest approach is the clone&own of elements. This implies that changes in the source element will not propagate to its clones. Alternatively, it is also possible to keep a dynamic binding with the source element so that changes in the source ripple throughout its clones.

2.5 Conclusions

This chapter has presented the principles behind *Web Augmentation* and highlighted how this technology shares multiple similar objectives as *End-User Development*. Indeed, as it allows users to recycle, reuse and exploit material that can be obtained from other web sites it supports the construction (by the end-users themselves) of more usable and more adapted web application. One of the biggest challenges is how treat dynamic states of Web applications, which means contents that evolves over time. Whilst this remains an unsolved issue that should be addressed by future research, it is possible to envisage various copy and paste strategies to address the problem.

In the study of WA tools, it has been observed a prominence of tools that run exclusively on the client side. This is not surprising as one of the advantages of using a client-side approach is the faster execution that has a huge impact on the user performance while interacting with the web application making it possible to provide immediate feedback to the users. Moreover, users do not need to understand sever side functioning and to deal with complex installations on a remote web server (for which they, most of the time, have no access rights). Whilst client-side approach is not a panacea, we suggest that this is still a suitable strategy for giving end-users more autonomy on the scripts they want to develop.

WA techniques has huge potential for end-user programming in particular for allowing users to recycle and reuse material that can be obtained from other web sites. One of the biggest challenges is how treat dynamic states of Web applications, which means contents that evolve over

time.

In the next chapter *WebMakeup* is going to be more deeply explained. *WebMakeup* is classified as a client-side visual programming tool that can integrate different websites or augment a single one enabling content, behaviour and presentation modification. Additionally, all augmentations can be shared with other users.

Chapter 3

Web Page End-User Personalization

<hr />

3.1 Introduction

Modding is a slang expression that is derived from the verb “modify”. Modding refers to the act of modifying hardware, software, or virtually anything else, to perform a function not originally conceived or intended by the designer [Wik]. Web Modding sits within the field of *Web Augmentation* [Bou99]. The rationales for modding should be sought in the aspiration of users to contextualize to their own situation the artefact at hand. This ambition is not limited to video games, cars or computer hardware. The need also arises for the Web. As an example, consider a TV-guide website (e.g. *www.tvguia.es*). For a given user, favourite channels might be scattered throughout the channel grid, hence, forcing frequent scrolling. In addition, users might move to other websites (e.g. *www.filmaffinity.com*) to get more information about the scheduled movies. If *tvguia* is recurrently visited, this results in a poor user experience. Traditionally, this is addressed through Web Personalization,

i.e. a set of techniques for making websites more responsive to the unique and individual needs of each user [CDA00b]. Similar to other software efforts, traditional personalization scenarios prioritize the most demanded requirements while minority requests are put aside. However, as a significant portion of our social and working interactions are migrated to the Web, we can expect an increase in "long-tail" personalization petitions. These idiosyncratic petitions might be difficult to foresee or too residual to be worth the effort. **"Web modding"** moves the power to the users. Web modding (hereafter, just modding) aims at Web content being consumed in ways other than those foregone by Web masters. Rather, users are empowered to rearrange Web content "after manufacture", e.g. removing content of no interest (leading to less cluttered pages while reducing scrolling) or placing new content obtained from somewhere else (reducing moving back and forth between sites so that a single viewing context is provided). The research question is how to achieve this empowerment.

This question admits different answers depending on the target audience. We frame our work along three main requirements: available time (30'), available expertise (no programming experience), and sparking motivation (improving the Web experience). This rules out fine-grained, absorbing programmatic approaches, and demands more declarative and abstract means. This is what Domain-Specific Languages (DSLs) are good for. DSLs are full-fledged languages tailored to specific application domains by using domain-specific terms. Domain abstractions are closer to how users conceive the problem, facilitating engagement, production and promptness. First, this work characterizes *Web modding* (Section 3.2). This work's contribution rests on the three pillars of DSLs applied to Web modding, i.e. ascertaining the right concerns (Section 3.3), finding appropriate DSL constructs to capture those concerns (Section 3.4), and finally, developing suitable editors that ease the production of DSL expressions (Section 3.5). Section 3.6 shows how to share web mods and section 3.7 explains how to solve mod problems with dynamic wen content. Finally, section 3.8 shows how create efficient web mods and

Figure 3.1: Number of users of *WebMakeup*

section 3.9 evaluates the proposed solution.

The later is realized through *WebMakeup*, a *Google Chrome* extension that turns this browser into an editor for defining Web mods. *WebMakeup* is available at the *Chrome Web Store*: `https://chrome.google.com/webstore/detail/alnhegodephpjnaghlcemlnpdknhbhjj` and a demo video is available at `https://vimeo.com/204338864`. Figure 3.1 shows the evolution of the number of users *WebMakeup* has had from May 2017 to July 2017 with an average of 68 users. Mods are exported as *Google Chrome* extensions that once installed, will transparently customize the page next time is visited. We start by characterizing Web Modding.

3.2 Characterizing Web Modding

Web Modding sits in between Web Personalization [RSG01] and Web Mashup [YBCD08]. As a personalization technique, modding aims at improving the user experience by customizing Web content. There are also important differences. In Web Personalization, the website master (the "who") decides the personalization rules (the "how"), normally at the inception of the website (the "when"), preferentially using a server-centric approach (the "where"). By contrast, modding aims at empowering end-users (the "who") to rearrange Web content once in operation (the "when") by acting on the DOM tree (runtime realization of HTML pages) (the

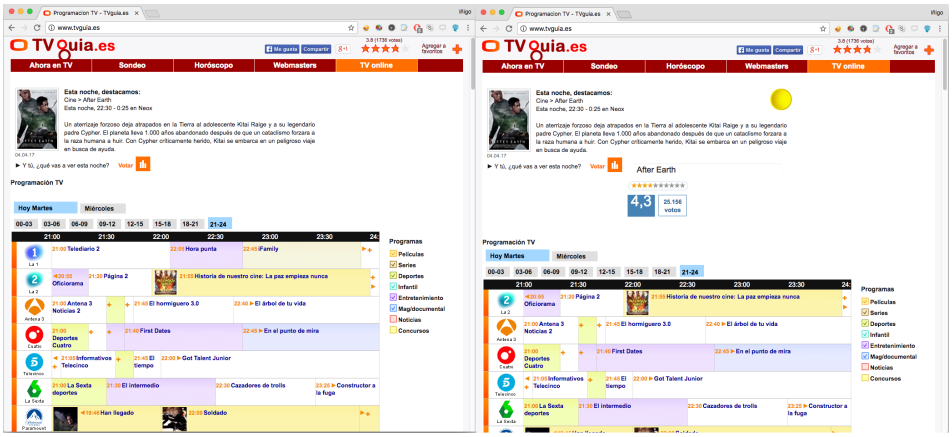

Figure 3.2: *www.tvguia.es* before (left) and after (right) being modded: channel *"La 1"* is removed & *filmaffinity* ratings are introduced.

"how") at the client side (the "where"). Nevertheless, modding also shares similitudes with mashups: both tap into external resources. However, and unlike mashups, modding does not create a bright new website. Rather, it sticks with the modded website. Just like modding a car does not build a new car, modding a website does not create a new website but just operates on the browser side to change its DOM tree.

Web modding pays off for websites frequently visited but unsatisfactory Web experience. As an example, consider *www.tvguia.es*. This website provides the channel grid plus the-movie-of-the-day recommendation (see Figure 3.2 (left)). A user might just focus on some few channels, hence a thorough channel grip becomes a nuisance. In addition, content from other websites about the recommended movie might be of interest. Figure 3.2 (right) depicts a modded version: channel *"La 1"* is removed whereas additional content about the recommended movie is obtained from *www.filmaffinity.com*. The fragment extracted from *filmaffinity* is referred to as a *widget*, in this case, the *filmAffinity* widget.

The bottom line is that mod scenarios are characterized as being idiosyncratic, situational, and, potentially, short-lived, aiming not so much at synergistically combining third-party data (as mashups do) but

improving the user experience of existing websites. Since these scenarios are very dependent on Web consumption habits and user interests, modding necessarily has to be do-it-yourself (DIY). This implies keeping the modding effort on a scale within the time and the skills of end users. This scale is a main driver in finding a balance between expressiveness (what can be modded) and effort (the cost of developing the mod). Our target is for *Web Modding to be conducted by users with no programming skills in around 30 minutes.*

Implementation wise, modding implies browser-based programming. Modding is already possible for skilful JavaScript programmers but certainly outside the scope of end users [Pil05]. This rules out fine-grained, absorbing programmatic approaches (e.g. *Chickenfoot* [BWR+05], *Co-Scripter* [LHML08]), and calls for coarser grained, light-weight component-based standpoints. Unfortunately, most works on Web components (e.g. widgets) favours a programmer perspective, addressing the definition [W3C], implementation [EBG+07, HT08] and cloning of Web components [MSCC13, FWRG11a]. A higher-level of abstraction is needed. DSLs come to the rescue. DSLs are full-fledged languages tailored to specific application domains by using domain-specific terms [Fow10]. To increase the chances for DSLs to be adopted, three main landmarks stand out: ascertaining the right concerns, finding appropriate constructs to capture those concerns, and finally, developing appropriate editors that intuitively permit users to come up with DSL expressions. Next sections address each of these landmarks for modding.

3.3 Ascertaining The Right Concerns

Web Modding sits within the field of *Web Augmentation* [Bou99], i.e. conducting changes upon the runtime representation of HTML pages (i.e. DOM trees) at the time the page is loaded into the browser. Those changes can affect the content, rendering, layout or dynamics of the page. Among the numerous uses of *Web Augmentation*, modding focuses on

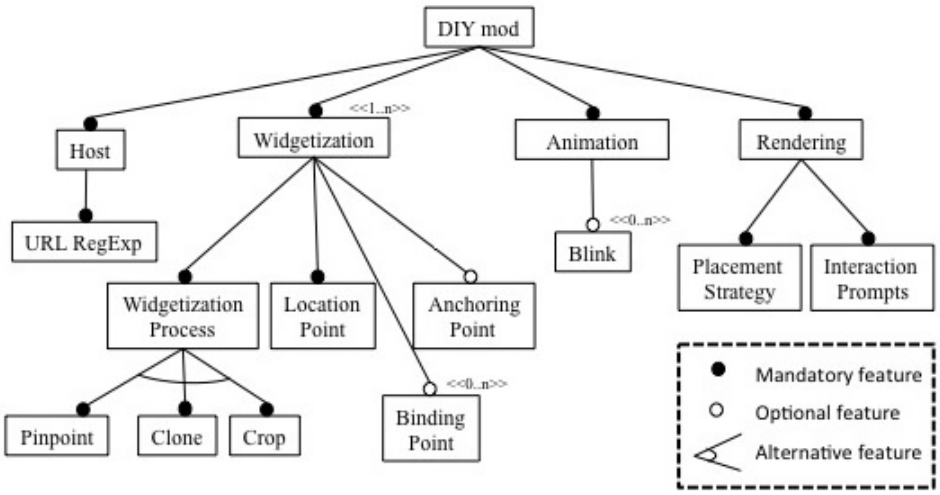

Figure 3.3: Feature diagram for DIY Web Modding.

performing a function not originally conceived or intended by the host designer [DA15]. Finally, DIY modding addresses the empowerment of end-users to mod by themselves. As in other areas of end-user design, more (expressiveness) can be less (usage). Therefore, DIY modding is necessarily going to be less expressive (i.e. more domain-specific) than general modding. We focus on improving the user experience through content rearrangement, i.e. content removal (leading to less cluttered pages) and content cloning, i.e. taking content from somewhere else (providing a single viewing context while cutting down moving back and forth between browser tabs). This sets the domain.

Along DSL good practices [MHS05], concerns raised during DIY modding are captured as a feature diagram [KCH+90]. A feature diagram represents a hierarchical decomposition of the main concepts (i.e. features) found in the domain. The diagram also captures whether features are mandatory, alternative or optional. Figure 3.3 depicts the feature diagram for the domain "DIY modding". Issues include, **hosting** (i.e. setting the ambit of the modding), **widgetization** (i.e. the definition of widgets whose addition and removal shape the modding), **animation** (i.e. defining possible dynamics among the widgets), and finally, the **rendering**

directives for the mod. Next paragraphs delve into the details (bold font is used for the features).

3.3.1 Hosting

A *mod* is a set of changes conducted upon the runtime representation of an HTML page at the time the page is loaded. Therefore modding does not happen in a vacuum but within the setting of an existing website, i.e. the **host**. The host can be characterised by a URL expression or a regular expression (e.g. www.amazon.com/*) so that all pages meeting the expression are subject to the mod. The expressiveness much depends on the target audience. For our purpose, we limit **url regexp** to those ending by "*". More complex expressions are not supported.

3.3.2 Widgetization

Modding is about customizing HTML content. HTML pages are conceived as DOM documents. The granularity at which HTML customization happens influences complexity. A finer-grained approach will certainly improve expressiveness but at the cost of complexity and learnability. Therefore, we opt for a coarser grained approach: *widgets*. For the purpose of this work, a widget is a coarse-grained DOM node (a.k.a. fragment), which accounts for a meaningful mod unit.

A widget can be defined *from scratch* through HTML and JavaScript. This is not possible for non-programmers. Alternatively, 3rd parties can help. But this also contradicts our setting that is characterized as being idiosyncratic, situational, and, potentially, short-lived, hence, the introduction of 3rd parties does not payoff. We are then forced to explore a different approach: *widget mining*. That is, users do not create widgets on their own but extract them from existing pages at the time the need arises. We then do not talk about widget creation but *widgetization* of existing code. To this end, we support tree variants: pinpoint, crop and clone.

Pinpoint supports inside-the-host widgetization, i.e. the widget is obtained from the host. In this case, extraction points hold the host's URL and a structure-based coordinate, i.e. an XPath expression that pinpoints the DOM node to be turned into a widget (see later). Widget *movie-of-the-day* is a case in point. It singularizes the DOM node that holds the content for the recommended movie. However, outside-the-host widgetization is more complex. A naive approach to extract existing functionality from a web page is just copy&paste. However, since HTML, CSS, and JavaScript are all "context-dependent", moving fragments from their original scope is rarely feasible. This moves us to the other two variants.

Clone is used for outside-the-host widgetization when the fragment to be extracted is "static", i.e. it holds content and style but not functionality (no JS scripts associated). The aim is for the widget to look like the raw content in the original page. Here, widgetization is achieved through cloning. Since style needs to be replicated, cloning is not limited to the selected DOM node but also its ancestors' CSS styles are inherited[1] (see Figure 3.4). Since code is replicated, what if the original is upgraded? How are changes propagated to the replica? To this end, we introduce *refreshTimer,* a parameter that sets the refresh polling time to five possible values: onload (i.e. the widget is calculated every time the host page is loaded), onblink (i.e. the widget is calculated every time the user click on the triggering widget), daily, onload, weekly or never.

So far, we assume widgets to be obtained from a single HTML fragment (**singleCloned**). However, the content of interest might be spread across different nodes. An interesting case is that of the Deep Web. Deep Web sources store their content in searchable databases that only produce results dynamically in response to a direct request. Here, the "meaningful functional unit" (i.e. the node to be widgetized) includes two fragments (**complexCloned**): the request fragment and the response fragment. The *filmAffinity* widget illustrates this situation. The "functional unit" includes

[1] *HTMLClipper* (http://www.betterprogramming.com/htmlclipper.html) is used to propagate replication from content to the associated CSS-like directives.

not only the ranking table (i.e. the output) but also the search entry form to type the movie title. Hence, creating *filmAffinity* implies two extractions: one to collect the ranking table; another to obtain the entry form[2]. Last but not least, so-created widgets are parameterized by the form entries. This permits to fix some form entries (e.g. set "Gone with the wind" as the movie title) or even better, bind the entry to some data which is dynamically extracted from the hosting page at runtime (so called "binding points", see later).

Crop is used for outside-the-host widgetization when the fragment is "dynamic", i.e. it holds scripts. In this scenario, cloning does not work. Functionality is difficult to extract in an automatic way (refer to [MSCC13] for the difficulties on extracting JS code). Here, we resort to pixel-based cropping. Using iframes, it is possible to load the source webpage on the background. Next, the desired fragment can be addressed by referencing the height and width w.r.t the cropping start coordinates.

Once DOM nodes are turned into widgets, they start exhibiting some additional characteristics. Widgets can have parameters and a state (i.e. visible or collapsed). But most importantly, widgets might hold reference points, i.e. directives that refer to some location in terms of Web coordinates. We distinguish tree kind of reference points:

- **Location points**, which indicate from where the widget was obtained. They contain a Web coordinate plus the framing page.

- **Anchoring points**, which refer to the new setting where the widget is to be rendered, i.e. the position (i.e. before or after) w.r.t. a given Web coordinate.

- **Binding points**, which denote how widget parameters can be bound to content from the host. It holds the name of the parameter and the host's Web coordinate. As an example, consider *filmAffinity*. This

[2]Labelling a newly created *widget* with an existing name, makes the extraction engine glue them together and be offered as a unit (provided they come from the same page).

widget needs to be recalculated every time *guiaTV*'s movie-of-the-day changes. To this end, *filmAffinity* holds the *title* parameter. This parameter holds a binding point to the DOM node in *guiaTV* that keeps the title of the recommended movie. At runtime, the movie-of-the-day is recovered, and *filmAffinity* is dynamically computed after the current title.

Previous paragraphs refer to Web coordinates. A Web coordinate is a means to address content within a DOM tree. Traditional mechanisms include structure, attribute, visual or coordinate-based references or expressions [LCRT14]. Unfortunately, these mechanisms introduce a coupling with the current DOM structure. This makes coordinates fragile, i.e. they might no longer locate the expected fragment upon upgrades on the website. Website upgrades are a likely scenario. To achieve robust reattachment of mods, we introduce a redundant specification whereby all, extraction points, anchoring points and binding points, are characterized by distinct ways of locating the same fragment. More information about this in chapter 4.

3.3.3 Animation

Modding is about rearranging content. But this rearrangement does not need to happen in a single shot. Specifically, *widgets* can be in two states: visible or collapsed. When visible, widgets have the capacity to respond to events, such as keystrokes or mouse actions. When collapsed, widgets leave no trace in the screen. A *widget* has an initial state, i.e. the state at the time the hosting page is loaded (e.g. if visible, the widget is rendered as soon as the page is loaded). This state might be amenable to be changed by interacting with other *widgets*. A common approach for describing GUI dynamics is through statecharts [DF13]. However, statecharts are far too complex for our target audience. A simpler mechanism is needed.

Broadly, state changes can be described as event-condition-action rules. First studies, however, demonstrate that rules were a too fine-

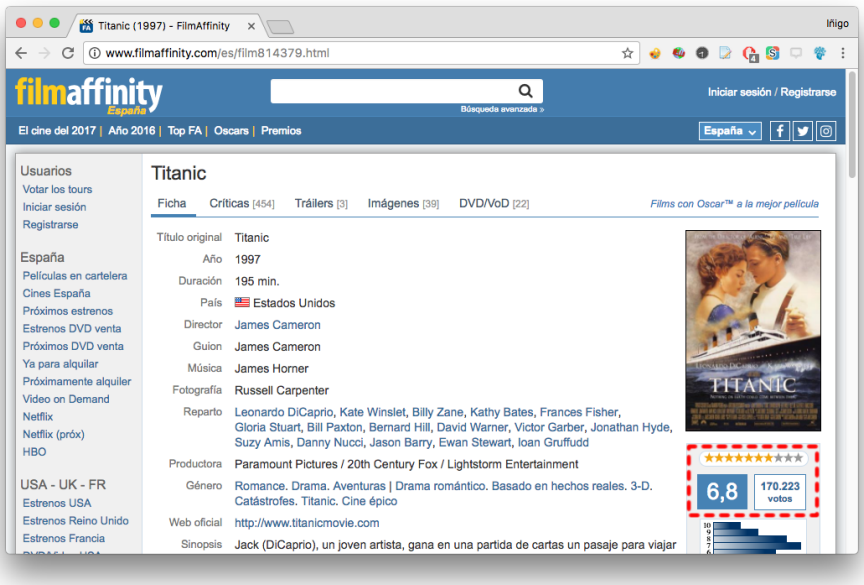

Figure 3.4: Facing website updgrades: redundant addressing for the filmAffinity widget

grained specification. Needed are higher abstractions that permit to capture recurrent patterns as a single construct. Based on previous evaluations, we noticed a recurrent animation pattern. Let's illustrate it with two widgets: *movieOfTheDay* and *filmAffinity*. Consider the later is to be made visible or collapsed upon mouse in/mouse out *movieOfTheDay*. This can be captured through a pair of rules:

> **ON** mouse-in *movieOfTheDay* **WHEN** *filmAffinity*.state = "collapsed" **DO** *filmAffinity*.state = "visible"
>
> **ON** mouse-out *movieOfTheDay* **WHEN** *filmAffinity*.state = "visible" **DO** *filmAffinity*.state = "collapsed"

We found this pattern so common that decided to introduce a DSL primitive for it: the blink. A ***blink*** accounts for a directed relationship between two widgets W1 and W2. We say *"W1 blinks W2"*, if acting upon W1 (e.g. clicking) causes W2 to change its state (from visible to collapsed

or vice versa, depending on the W2 current state). Previous example can now be expressed as *"movieOfTheDay blinks filmAffinity on clicking"*. So far, we limit animation to *blinks*. *Blink* events are limited to *mouse-in* (being *mouse-out* its *blink* counterpart), *doubleclick* (being *doubleclick* also its *blink* counterpart) and *click* (being *click* also its *blink* counterpart). Other events were considered but they add unnecessary complexity to end-users and they were not included.

Additionally, *WebMakeup* provides the possibility of adding some animation pattern:

- **click2erase**. This pattern involves only one widget. It accounts for a single blink. On clicking on the widget, this is gone for the current session.
- **click2alternate**. This pattern involves two widgets which are shown alternatively. It accounts for two blinks in which initially only one is visible. Clicking on the visible widget, one is substituted by the other widget.
- **conjunction**. These patterns involve three widgets or more: the triggering widgets, and two triggered widgets that are shown simultaneously.
- **disjunction**. These patterns involve three widgets: the triggering widgets, and two triggered widgets that are shown in alternation.
- **incremental**. This pattern involves "n" widgets which are gradually presented as the user clicks. It accounts for "n-1" blinks.
- **domino**. It leverages the previous pattern so that clicking on the last widget collapses all its predecessors except the triggering widget.

These patterns are available through the namesake tab. Pattern definition is achieved using a similar approach to PowerPoint's SmartArts (see Figure 3.5). Keeping the ALT key pressed down, select the involved widgets. As

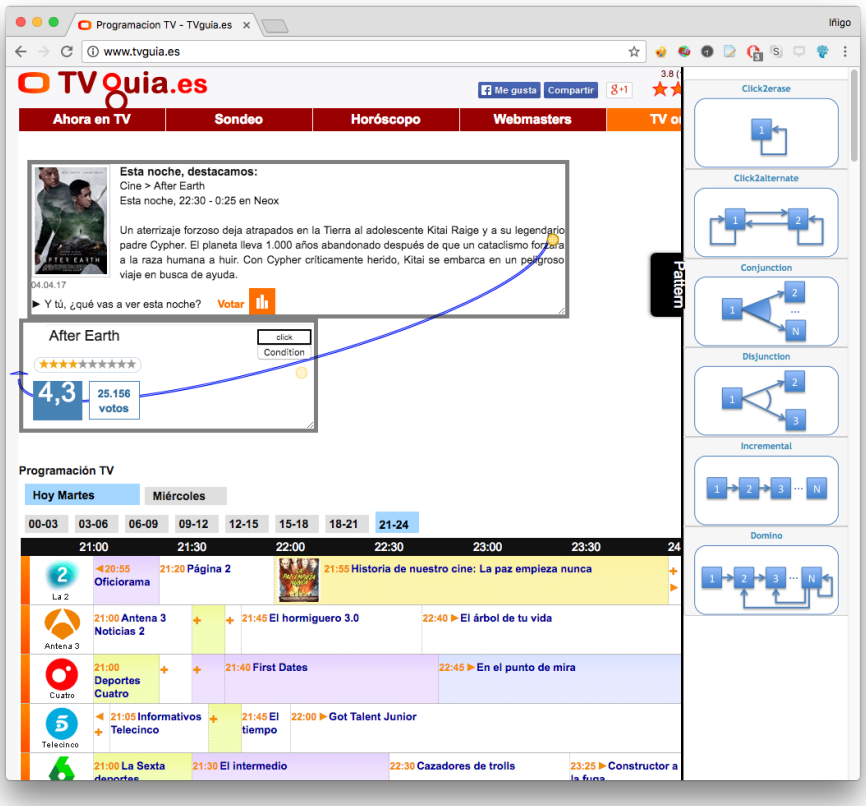

Figure 3.5: Setting patterns. Keeping the ALT key pressed down, select first the widgets, and next, the blink pattern

widgets are being selected, the widget region is shadowed, highlighting the order of the widget at hand. Once all the participating widgets are picked out, and keeping the ALT key pressed down, choose the desired behavior in the pattern tab. *WebMakeup* will automatically generate the blinks that jointly account for the pattern at hand. These patterns have been included owing to the fact that they help to define some of the most used widget event sequences by end-users in the augmentation design.

3.3.4 Rendering

Inlaying new widgets into an existing DOM structure can make the host's layout be disrupted. Specifically, HTML introduces some attributes to describe the rendering strategies for DOM nodes, namely: the layout strategy (HTML's "display" attribute) which can be arranging the content horizontally (inline) or vertically (block); minimum and maximum size intervals (HTML's attributes *minHeight*, *minWidth*, *maxHeight*, *maxWidth*); and the overflow strategy (HTML's "overflow" attribute) that indicates what to do in case the content exceeds the size intervals (i.e. make container scrollable, show the overflowed content or hide the overflowed content). Widget inlaying might disturb the page layout, causing one-dimension distortion or even worse, two-dimension distortion. We decide this concern to be hardwired within the DSL engine. Better said, the engine supports contingency actions to alleviate this situation (e.g. if container is 80% full then, *Web Augmentation* overflow strategy is set to "warn"; if container is 90% full and the widget fits inside then, *Web Augmentation* overflow strategy = "resize", etc.).

3.4 Finding Appropriate Constructs

Previous feature diagram captures main concerns to be solved during DIY modding. Next, these abstractions are realized in a language by looking into variabilities and commonalities in the feature diagram [MHS05]. Variable parts must be specified directly in or be derivable from DSL expressions. In the first case, the variants become DSL constructs. However, some alternatives can be hardwired into the DSL engine as heuristics. Being heuristics, they might fail and hence, they are not as reliable as if provided by the user. The upside is that they simplify the user's life, hence, improving learnability and development. We decided *rendering* to be hardwired into the engine. That is, widget placement is to be assisted by the DSL engine. The rest of features are set by the user

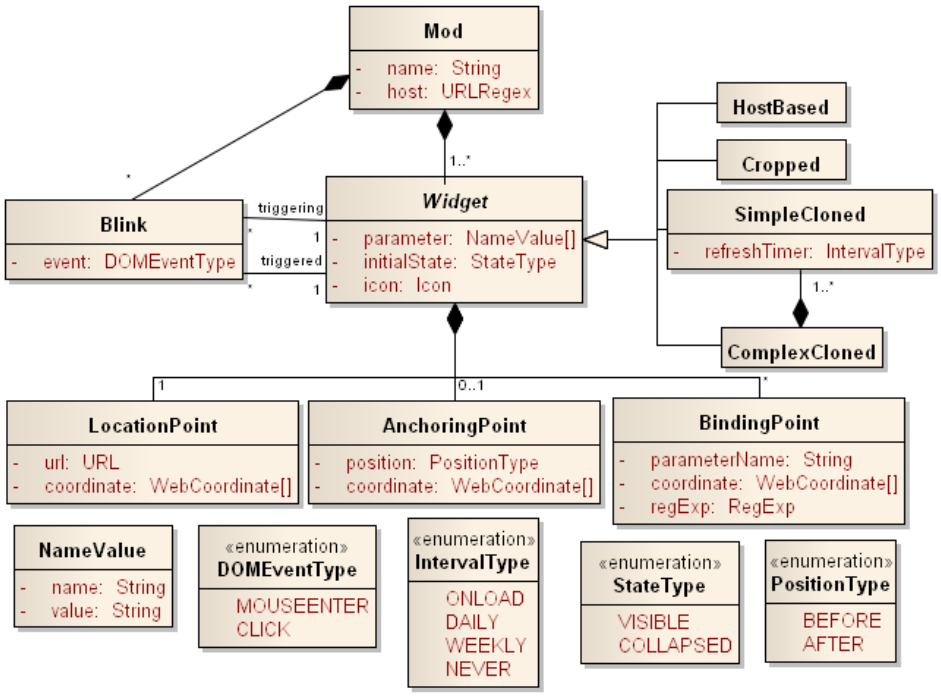

Figure 3.6: A DSL for Web Modding: abstract syntax.

through the DSL. This section introduces the DSL metamodel.

Figure 3.6 provides the metamodel for mod description. A *mod* is a set of changes conducted upon the runtime representation of an HTML page (i.e. the host). These changes are described in terms of widgets. Widgets are characterized by a **locationPoint** (i.e. how to obtain it), an **anchoringPoint** (i.e. where to locate it), and, optionally, distinct **bindingPoints** (i.e. how widget parameters can be obtained from the host's content). Each widget stands for a rearrangement operation as follows:

- If *locationPoint* exists without *anchoringPoint*, this accounts for content removal (only for host-based widgets).
- If *locationPoint* differs from *anchoringPoint*, this accounts for content displacement (only for host-based widgets).
- Otherwise, the widget captures content addition.

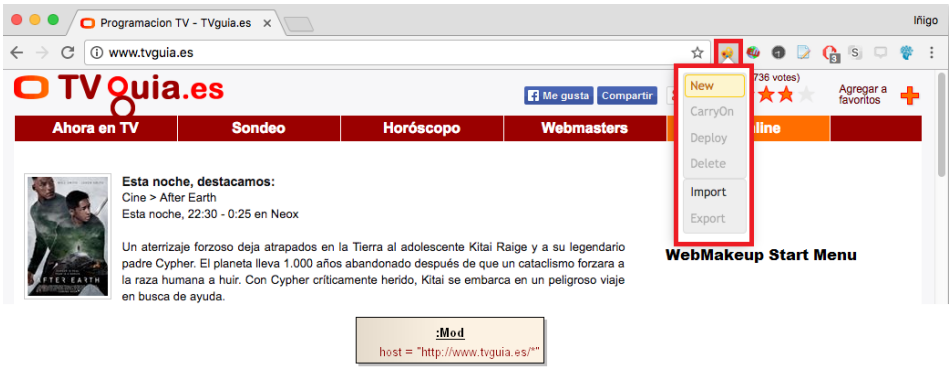

Figure 3.7: *WebMakeup*: mod initialization.

But not all contents need to be added/removed at loading time. *Blinks* permit to hand this decision over to the current user. This makes content rearrangement dependent upon user interactions. For instance, *"movieOfTheDay blinks filmAffinity on clicking"* permits to postpone till runtime the decision of rendering *filmAffinity*. If you click, you get *filmAffinity*. If complementary outside-the-host widgets exists (e.g. *filmIMDB* extracts the ratings from the IMDB website), then this content can be shown either simultaneously (e.g. *"movieOfTheDay blinks filmIMDB on clicking")* or in a cascade way (*"filmAffinity blinks filmIMDB on clicking"*). But not only additions, also removals can be left pending until interaction time: *"movieOfTheDay blinks movieOfTheDay on clicking"* permits current users decide whether they want to delete (i.e. collapse) *movieOfTheDay* by clicking on it. Next, we address how to make mods affordable to end users.

3.5 An Editor For DIY Mods

DSL acceptance is heavily influenced by the existence of appropriate editors, more to the point if targeting end users. This section outlines *WebMakeup*, an editor for DIY mods. *TVguia* is used as an example. The description goes along the creation of a *mod*, i.e. a model conforming to

Figure 3.8: *WebMakeup*: mod filling up. The *piggyBank* tab is displayed.

the metamodel presented in the previous section.

Mod creation (Figure 3.7)**.** *WebMakeup* is a plugin for *Google Chrome* browser. Its installation is reflected by the *WebMakeup* button at the right of the address bar. On clicking this button, a scrollable menu pops up. By clicking *"New makeup"*, the user initializes the mod model (Figure 3.7 (bottom)). *WebMakeup* turns the current page into the editor canvas: the pointer is turned into a camera, a grid-like structure is interspersed on top of the current DOM tree, and the *piggyBank tab* pops up.

Mod populating (Figure 3.8). A widget is a DOM node but not all DOM nodes are widgets. We need to singularize the selected DOM node

Figure 3.9: *WebMakeup*: defining *blinks*.

that accounts for a meaningful HTML fragment. Meaningfulness is not inferred by the tool but indicated by the user. To this end, and, as the user moves the cursor around the screen, the DOM node under the current cursor location is highlighted. By clicking, the user singularizes this node as a meaningful HTML fragment, i.e. a widget. A nuisance is the handling of "hidden nodes". These nodes are those that do not have a graphical counterpart and hence, they cannot be pinpointed through the cursor. For instance, a table row (<*tr*>) is graphically hidden if its graphical space is totally taken by its content. If the row does not explicitly have some graphical counterpart (e.g. a border), then all the space is occupied by the

row's content so that the cursor will always select the row's content rather than the row element itself. To overcome this problem, we resort to the keyboard. Keys *"w", "s"*, *"a"* and *"d"* help to move up, down, left and right along the DOM tree, respectively, w.r.t to the node being pinpointed by the cursor.

No matter the selection mechanism (i.e. cursor vs. keyword), the selected node is surrounded by a decorator. This decorator permits to set the initial widget state by clicking on the "eye" icon (decorators' upper left-hand side corner): visible (open eyes) & collapsed (closed eyes). The example contains two inside-the-host widgets (i.e. *movieOfTheDay* and *TVE1channel*) and outside-the-host widget (i.e. *filmAffinity*). The latter is dragged&dropped from *piggyBank*[3]. Click on this tab to expose the widgets collected from other pages (see it in display in Figure 3.8). Placement heuristics will warn or prevent from dropping widgets in certain places. In all cases, *WebMakeup* works out the Web coordinates.

Mod enhancement (Figure 3.9). At any time during editing, widgets can be:

- Deleted. Widget removal is achieved by clicking upon the *X* icon on the widget decorator. In the example, we remove *TVE1channel*. Model wise, this is reflected by deleting its anchoring point. An important remark: banners cannot be removed. Though this is a common desire among users, up to 84% of the top 100 websites rely on advertising to generate revenue. Though adverts can be a nuisance, they are the ones that pay the bill. So for the time being, we take the decision of making *WebMakeup* ad-friendly.

- Rearranged. This is conducted through drag&drop once the widget

[3]Outside-the-host widgets can be obtained at any time. To this end, the right-click contextual menu is extended with the *widgetizeIT* item. At any time, select it for a grid-like structure to be interspersed on top of the page you are looking at. As the user moves the cursor around the screen, the DOM node under the current cursor location is highlighted. By clicking, the selected node is turned into a widget and kept in the extension's variable: *piggyBank*.

is selected. Model wise, this is reflected as an update on the anchoring point.

- "Blinked". *Blinks* are graphically represented through pipes. Widget decorators have in their right-hand side a yellow circle. This circle denotes a pipe start. Click and drag from this point to expand till reaching another widget. This sets a blink from the triggering widget (the pipe's start) to the triggered widget (the pipe's end). An entry field on top of the pipe serves to indicate the blink's event. Figure 3.9 illustrates the case *"movieOfTheDay blinks filmAffinity on clicking"*.

Once the edition finishes, the mod can be executed clicking on "Deploy" in the *WebMakeup* menu button. The mod will be automatically enacted next time the host page is loaded.

3.6 Sharing

WebMakeup scripts are stored locally in the Web browser. *WebMakeup* does not support collaborative development. Nonetheless, users can export scripts into a file and next share them through email or other means. Consumers should have *WebMakeup* installed and use the "Import" option (Figure 3.7). Drag and dropping the file into the browser, the mod is installed redirecting the tab to the augmented website. Also in the scrollable menu, the entry "CarryOn" permits consumers to tune imported scripts to their own likes.

3.7 Facing dynamic web content

Dynamic content has an increasing presence among Web applications. To make mods safely in dynamic web pages, we have to take care of performing the two main steps of a mod: the identification/extraction of the content and injection of content. In order to identify/extract data, it is

needed to replay the circumstances when the data was extracted. The main problem in dynamic pages is that content could not be on page load but after some delay or user interaction. If this is the case, it is necessary to track the DOM and wait for the availability of the data. Two issues: (Rich Internet Applications) RIA-aware widgets and RIA widgets [MHBJ14]

3.7.1 RIA-aware widgets

RIA-aware widgets are those whose availability is conditioned to RIA-generated content. Widget description is extended with a precondition. For the widget to be enacted, the precondition needs to be meet. For RIA-aware widgets the precondition can be the presence of a RIA-generated fragment. Alternatively, we can describe preconditions in terms of user events in the understanding that RIA-generated fragments are the cause of user interaction. We then define preconditions in terms of user interactions so this requires to be able to track RIA-generated fragments back to the user interaction that caused them.

RIA-aware widgets is a characterization transparent to the user. Designers define both in the very same way. It is the engine's duty to ascertain what are the RIA-aware widgets. Broadly, a RIA-aware widgets keep some kind of relationship to a RIA-generated fragment. This relationship can be due a either binding or blink association.

3.7.2 RIA widgets

RIA widgets are those that come from a RIA fragments in a website. RIA widgets are cropped widgets. There are two different types:

When the user have to surf the website to reach the point in which the widget is (example: VisualEconomy website[4]) When the widget itself has all the information but the user must click on it to see the information he is interested in (example: tutiempo website[5]).

[4]http://www.visualeconomy.com/

[5]https://www.tutiempo.net/donostia-san-sebastian.html

Detecting whether an HTML fragments is RIA based

The mechanism to track DOM changes is the Mutation Observer. This observer notifies each modification but many of them in one shot then making it quick instead of notifying all the changes one by one. When the observer detects a change in the website, *WebMakeup* tries to insert all the widgets that have not been inserted before. If all widgets have been inserted, it does not do anything but if one at least has not been inserted, it tries to insert it according to its location. Observer detects changes do to dynamic elements or user interactions and thank to them, new content could have been inserted.

Some dynamic websites generate a dynamic ID for the web elements. This means that every time a user visits this website, apparently it is the same but the structure of the website could be different an automatically generated. The ID of the elements is the most robust part to identify a node [LCRT13] and most locator algorithms use it to locate nodes. Unfortunatelly, if automatic ID generation happens, all this algorithms fail trying to locate the node despite the fact that it is in the website. That is why different node locations algorithms must be implemented. More detailed information in section 4.4.

Extracting widgets from RIA-based fragments

A widget is characterized by a URL and a locator. This is fine for readily-available fragments where URLs denote the page from where to extract the fragment. In a RIA setting, this is not enough. RIA-based fragments are not readily available, but some previous user interactions are required. These user interactions need then to characterize the widget as they describe the setting when the extraction took place.

The strategy is that first of all, mineIT operates as usual. Once, the user selects the fragment to be extracted, *WebMakeup* figures out whether this fragment stands for a RIA fragment (e.g. by computing whether there exist the widget in the initial page and the current page). If so, the starting page

is reloaded, and the user is prompted to replay the interaction sequence that leads to the desired fragment. Meanwhile, *WebMakeup* records the interaction untill it detects the desired fragment is available. When so, it checks out with the user, and generates the RIA widget. When the widget is inserted in the website, it automatically repeats all the click done by the user and the widgets shows only the desired element.

3.8 Efficiency

The more widgets end-users insert in an augmentation, the more resources the computer will need to carry out the augmentation. That is why *WebMakeup* engine optimizes the memory consumption and the efficiency in order not to overload unnecessarily computer resources.

3.8.1 Guideline: Reduce memory consumption the Number of Active Event Listeners

Augmenters perceive Web Content than through listeners. Hence, augmenters make extensive use of listeners. Broadly, a listener is a triplet `<node,eventType,handler>` where handler is a function to be enacted when an occurrence of eventType occurs upon node. Worth noticing, the circumstances that make the listener logic apply might not be exclusively described by the an-event-happens-on-a-node. Within the handler, flags might need to be checked to ensure certain circumstances apply. This happens for Web Content that might act in two different modes. Here, besides the triggering of the event occurrence itself, handlers might need to check whether this occurrence happens when in the right mode. Augmenters might also exhibit this behavior where some mods might depend on some configuration parameters being set or not.

A naïve approach can fix the set of listeners (i.e. the binding between these three elements) at page loading time, attached them to the root node, and hide flag conditions into the handler code. This certainly works. The

DOM event model ensures that no matter where the event occurrence happens it will bubble up to the root node, hence, enacting the handlers. In addition, the handler would account for both the required flags and the events being risen in the appropriate nodes. Fair enough. The issue is that all the JavaScript objects referenced by the handler are retained while a listener is registered. This costs might be affordable when listeners are sparsely used. Unfortunately, augmenters are listener intensive, and the lack of a strategy for listener de-registration might end up in memory depletion. Reducing the number of listeners registered, reduce the number of handlers and the JS objects they refer to.

Therefore, augmenters should strive to minimize their burden. This guidelines aligns with the suggestion given by Mozilla[6]. We conceived listeners as `<node,eventType,handler>` bindings. Listener overhead might be reduced through the technique dynamic binding. It aims at reducing the number of active listeners at a given moment by dynamically managing `<node,eventType,handler>` triplets. Flag conditions are moved from handlers to a listener manager which regulates the creation and destruction of those bindings as the circumstance arises. If browsing is in mode One then, create listeners that apply to model One. If browsing is in mode Two then, delete mode-One listeners and create those for mode Two. In this way, we prevent the happening of spurious event payloads which will cause the triggering of handlers whose logic is posteriorly called off as flags do not apply. This technique makes listener management a bit more complex. Nonetheless, this effort pays off in terms of reducing the number of spurious events and the number of active listeners, reducing the needs for memory and CPU.

[6]https://developer.mozilla.org/en-US/Add-ons/Performance_best_practices_in_extensions

3.8.2 Guideline: Make Efficient Rendering of the Augmentation

Causes for sluggish Web interaction include reflows and repaints[7]. Repaint is what happens whenever something is made visible when it was not previously visible, or vice versa, without altering the layout of the document (e.g. adding an outline to an element, changing the background color). Repaint is expensive because the browser must verify the visibility of all other nodes in the DOM tree. A bigger penalty is incurred by reflows where the layout is affected. This happens whenever the browser window size is changed, whenever a style is changed that affects the layout, whenever the className property of an element is changed, or whenever the DOM tree is manipulated. Augmenters tend to involve extensive DOM manipulation. Therefore, augmenters might cause frequent reflows and repaints, hence, impacting GUI responsiveness.

Keep the number of repaints/reflows to a minimum is a good option to avoid unnecessary repaints and finally, time consumption. The strategies on node modification, the one-shot strategy, rather than conducting changes in a piece-meal fashion, this strategy advices to aggregate all changes in a single step (if possible) hence, involving a single repaint/reflow.

3.8.3 Optimization. Experiment design

Blinks are by large the main feature that impact both performance and storage.

For the performance, blinks involve the switching between visible and invisible states. Each stage transition causes a rendering and a painting. Painting is also conducted from scratch and hence it is almost independent on the number of blinks. In our experiment from 2 to 30 blinking widgets, painting overhead ranged between 61 and 65 ms independently the number

[7] https://dev.opera.com/articles/efficient-javascript/?page=3#reflow

#widgets	Time(ms)
0	Rendering 69,430ms Painting 56,918ms Total 862,570ms
2	Rendering 74,443ms Painting 62,274ms Total 998,959ms
3	Rendering 75,418ms Painting 61,991ms Total 1049,743ms
5	Rendering 75,969ms Painting 63, 001ms Total 1090,814ms
10	Rendering 76, 566ms Painting 64,473ms Total 1180,365ms
15	Rendering 76,991ms Painting 62,459ms Total 1192,369ms
20	Rendering 78,166ms Painting 65,329ms Total 1231,127ms
30	Rendering 84,796ms Painting 64,716ms Total 1232,167ms

Table 3.1: Optimization experiment results

of widgets. By contrast, rendering is conducted in a single-widget bases. Hence, there exists a direct relationship between the number of blinks and the rendering overhead. Table 3.1 depicts the results of our experiment with 2, 3, 5, 10, 15, 20 and 30 blinking widgets. To limit the impact of the browsing context, each point stands for the average time of ten execution for the number of widgets at hand. The rendering time increases with the number of widgets, the more widgets the mod has, the more it needs. Rendering and painting are clearly lower when there is no mods in the website.

For the storage, blinks also impact storage as for the number of listeners required. Each blink accounts for a number of listeners between 10 and 14 listeners. Hence, 30 blinks might involve up to 420 listeners. However, thanks to the efficient algorithm implemented in *WebMakeup*, the number of listeners is always lower due to the fact that unnecessary listeners have been deactivated.

3.9 Evaluation

The main objective of *WebMakeup* is to make Web modding accessible to end-users. In this regard, the matter is mostly about affordance, that is, making Web modding accessible to an ample audience, the most possible. In this backdrop, the quality of use becomes predominant that

is "the user's view of the quality of a system containing software, and is measured in terms of the result of using the software, rather than properties of the software itself" [ISO]. ISO-25010 provides a framework to evaluate quality in use. This section provides a preliminary evaluation of *WebMakeup* along the ISO-25010's quality-in-use dimensions: (i) effectiveness (i.e., the capability of the software product to enable users to achieve specified goals with accuracy and completeness), (ii) productivity (i.e., the capability of the software product to enable users to expend appropriate amounts of resources in relation to the effectiveness), (iii) safety (i.e., the capability of the software product to achieve acceptable levels of risk), and (iv) satisfaction (i.e., the capability of the software product to satisfy users). Effectiveness and productivity can be measured objectively: number of completed tasks and minutes to complete them, respectively. On the other hand, while safety is an objective measure, our target population (i.e., people with no previous knowledge of JavaScript, HTML and CSS) may not have the means to perform such evaluation. Thus, we decided to evaluate trustworthiness, that is, how trustworthy participants perceived *WebMakeup* to be. Both trustworthiness and satisfaction were assessed through specifically designed questionnaires.

At the time of this writing, *WebMakeup* has been available as a *Chrome extension* for more than three year. According to the Chrome extension information, *WebMakeup* has 68 average weekly users. *WebMakeup* design has been driven by the aim to afford end-users to develop their own mods.

A group of end-users with no technical qualification was sought to evaluate *WebMakeup*. The call to participate in the study was issued among citizens of San Sebastian with no programming knowledge. The evaluation included questions about background in related technologies (e.g., JavaScript, XPath, etc.) to discard members of the group with technical knowledge. Evaluation results are available at `https://docs.google.com/spreadsheets/d/19Jvtlw1KFY_04NwpeZTMRIenmftFpOKut-9F4fSZqTc/edit?usp=sharing`.

3.9.1 Research Method

The study was conducted in a laboratory of the Computer Science Faculty of San Sebastian. All participants used computers with the same features (i.e., Intel Core 2 Duo 2.40GHz, 4GB RAM and Windows 7) and a clean installation of Chrome.

Procedure. All participants were handed out a sheet with instructions for each task (e.g., what Web to access, when to take note of the time, etc.). The study was divided in two tasks. The first part consisted on watching a video[8] about *WebMakeup* which describes all they need to the tasks of the second part. The second part consisted on four different tasks end-users must complete. The first task was the easiest and it was getting more and more laborious at the end.

Once the participants had watched the video they had to install *WebMakeup* in their computers. Afterwards, they started with the second part of the study, four different exercises to evaluate the different possibilities *WebMakeup* offers.

- First task: delete and move. The mail goal of the first task was that users were able to delete and move different elements of the website. With the objective of doing this task, participants have to visit the city council website and delete theoretically useless elements and move an important element from the bottom of the website to the top of it.

- Second task: insert a link and modify the content of an element of the website. The aim of this task was to evaluate the ability of users to insert new elements in the website (default elements). Moreover, this task intended to appraise what ease was to modify the content of any element of the website.

- Third task: creating and inserting widgets (cropped) from a different website and blinking it. The participants had to surf to "Pesa", a

[8] https://vimeo.com/204338864

Basque bus company and search the timetable. Afterwards, users had to clone the timetable through the right button of the mouse and selecting "MineIT". Additionally, participants had to insert a button (default widget) and blink this button with the widget inserted before. The idea was that this widget appeared when the users clicked on the button and search the timetable of all the buses of the company when the user desires.

- Fourth task: creating and inserting widgets (simple and complex widgets) from different websites, updating simple widgets, binding complex widgets and creating a blink pattern. This task was the most sophisticated and complete. First of all, the user had to create a weather widget of the city where he lived. After that, he had to create a complex widget with the valuation of the film in filmAffinity web page. Once the user creates both widgets he had to insert them in "tvguia", a Spanish TV guide. To complete the task the participant had to bind filmAffinity widget and the film of the day that the website recommend and update the weather widget to be updated everyday because the weather widget changes it content everyday. Finally, the participant had to create a "conjunction" pattern with the film recommended by the website, the filmAffinity and weather widgets to show the last two widgets when the user clicks on website recommendation.

3.9.2 Subjects

Twenty-six people participated in the evaluation of *WebMakeup*. The majority were male (57,7%) (see image 3.10 left) and the age of all of them was between 20 and 35 years. Concerning browsing behaviour, in the last year 10 participants have never installed a plug-in in their browser (see image 3.10 right). We also gathered information about their background on programming skills and 23 of participants have never programme or they have not had programming classes (see image 3.11 left). Despite this,

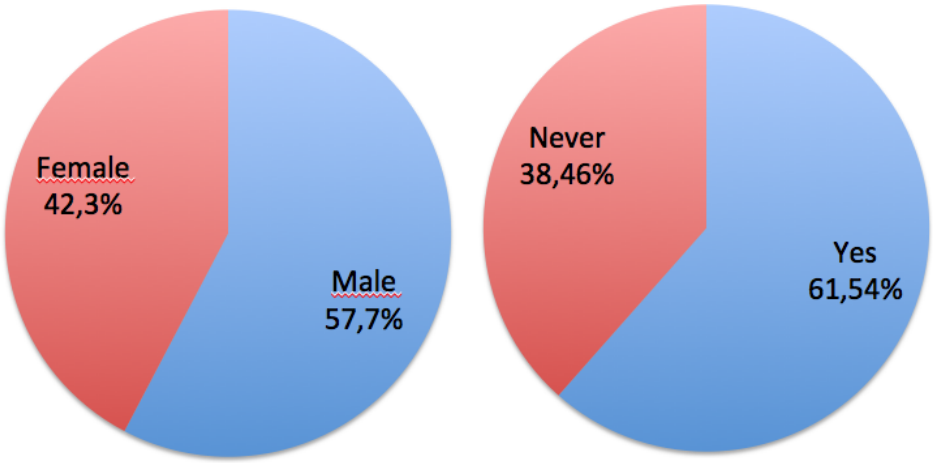

Figure 3.10: Left: Gender; Right: Users that have installed a plug-in

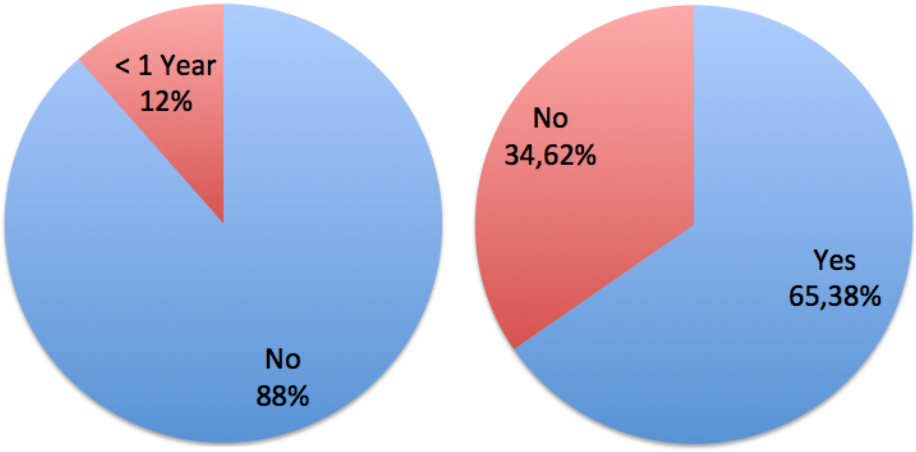

Figure 3.11: Left: Users with programming skills; Right: Users that have used editing programmes

17 participants have used editing programmes like Photoshop (see image 3.11 right). 10 participants surf the net more than an hour everyday in their free time (see image 3.12 left) and 18 of the participants cannot surf the net more than an hour in their job or university (see image 3.12 right). Most of the participants had visited the city council website (20) although 21 participant had not visited "tvguia" website before.

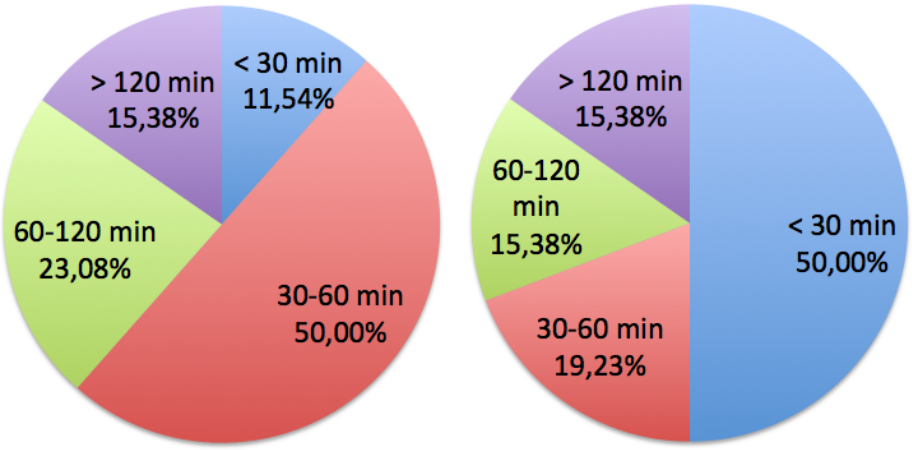

Figure 3.12: Left: Time surfing on the Internet in their free time; Right: Time surfing on the Internet in their job

3.9.3 Instrument

An online questionnaire have been used to know users' opinion about *WebMakeup*. The questionnaire consisted of five parts: background, effectiveness, usefulness, satisfaction and the time needed to finish each task. With the intention of evaluating effectiveness, the questionnaire contained the proposed tasks so every participant could indicate if they had performed them, whereas productivity was measured using the minutes taken in such tasks.

3.9.4 Data analysis

Descriptive statistics were used to characterize the sample and to evaluate the participants' experience using *WebMakeup*. Moreover, test analyses were performed to assess differences among groups of users. PASW Statistics 18 for Windows[SPS] was employed to perform the different analyses.

3.9.5 Results

The four tasks were successfully completed by all participants with the only help of the previously provided explanation. In the last part of the exercise, the fourth task, some participant had problems to remember how they had to use the pattern. We had to remember them that "alt" key must be pushed to define a pattern.

Besides the questionnaires an open question was included. From the 26 participants, 13 commented. The opinions about *WebMakeup* were positive. One participant suggests that all the buttons have the option to show a comment explaining what they do. Another participant says that delete button should ask the user if he agrees with that action. A participant suggest including a help option or an user manual. The rest of participants have made comment about *Webmakeup* standing out that they consider *WebMakeup* useful for them and recommendable for all their acquaintance.

Table 3.2 shows the valuation of users regarding the usefulness of *WebMakeup* and table 3.3 shows their opinion about the usability.

3.9.6 Effectiveness

All the participants were able to finish all the tasks during the evaluation. This shows that *WebMakeup* is easy to use for end-users despite some of them had some doubts because the video was long and they do not remember exactly how to do a certain part.

3.9.7 Productivity

Productivity is measured as the number of minutes required for each task: first task took in average 5,46 minutes. The faster participant needed 2 minutes to finish the task and the slowest one 24 minutes. This participant needed more time to do this task than the others when nobody else needed more time to do the first task than in the others. In the second task participants needed in average 7,54 minutes and between 3 and 14 minutes

Questions	User valuation	Deviation
Removing content from websites improves my Web experience	4,577	0,6433
Moving content from websites improves my Web experience	4,423	0,8086
Collecting content into a single website improves my Web experience	4,500	0,7071
Inserting my own links into a single website improves my Web experience	4,423	0,8566
Showing and hiding elements improves my Web experience	3,962	0,9157
Parameterizing content from websites improves my Web experience	4,192	0,7494
Reducing the need for going back and forth between browser tabs improves my Web experience	4,231	0,9511
I plan to use WebMakeup in the future	4,346	0,8458
I find WebMakeup interesting enough to tell to my friends	4,462	0,7060
To scroll a website is awkward	3,808	1,0206
When I visit a website frequently I have to scroll to find the content I want on the screen	4,192	0,6939
When I use a browser I have to change the tab frequently	4,192	0,7494
Tabs-switching is awkward	3,885	0,9089
Collecting content into a single website improves my Web experience	4,385	0,6972

Table 3.2: Evaluation questions and end-users answers (usefulness)

Questions	User valuation	Deviation
It was easy for me to widgetize fragments out of the donostia.eus page	3,962	0,7736
It was easy for me to delete elements of the donostia.eus page	4,423	0,5778
It was easy for me to move elements of the donostia.eus page	4,231	0,6516
It was easy for me to insert link or buttons in the donostia.eus page	3,846	0,7845
It was easy for me to change/add a new value to a existing element in the donostia.eus page	3,808	0,8010
It was easy for me to drag&drop widgets from the piggy-bank tab to the desired position in the webpage	4,038	0,8237
It was easy for me to create widgets out of existing pages (filmAffinity)	3,615	1,0983
It was easy for me to create widgets out of existing pages (eltiempo.es)	4,154	0,7317
It was easy for me to create widgets out of existing pages (Pesa)	3,769	1,1066
It was easy for me to set a simple-blink among selected widgets	3,192	1,1321
It was easy for me to bind widgets together (i.e. feeding widgets parameters from data in the TVguia page)	3,192	1,0961
It was easy for me to set a blink pattern among selected widgets	3,192	1,0206
It was easy for me to create a new makeup design	3,885	0,7656
It was easy for me to deploy a makeup design	4,385	0,6972
It was easy for me to restore an existing makeup design	3,923	0,7442
During the use of WebMakeup, I have always known how to do the things I was required to do	3,154	1,1204
There have been NO errors during the use of WebMakeup	3,615	0,8521
WebMakeup is fast enough	4,308	0,7884
In general, I am satisfied with the things that I have made with WebMakeup	4,115	0,7114

Table 3.3: Evaluation questions and end-users answers (usability)

de faster participant and the slowest one. For the third task they needed 8,23 minutes in average and between 3 and 17 minutes to finish the task. Finally, the last task was longer and they needed in average 13,65 minutes. In spite of the fastest participant needed 5 minutes to finish, the slowest one needed 31 minutes.

3.9.8 Satisfaction

Satisfaction is the capability of the software product to satisfy its users [ISO]. In this case, the product is the *WebMakeup* engine, and its ability to develop a working *WebMakeup*.

We found no statistically significant differences in many aspects evaluated in the evaluation but there some significant differences between people who install plug-ins. There is a significant difference with the idea that tab switching is irritating (p=0,01) and with the concept of scrolling that is tedious (p=0,023). There is also a compelling difference between people who use the net more or less than an hour in their job in the third task of the evaluation (p=0,016). Finally, another important differencecan be found between people who knew "tvguia" website and not in the second task (p=0,023).

3.10 Conclusions

Webies 2.0 no longer take the Web as it is but imagine fancy ways of customizing it for their own purposes. This work presents our vision for DIY modding along three main requirements: available time (30'), available expertise (no programming experience), and spark motivation (improving the Web experience). These requirements ground a coarse-grained, light-weight approach to DIY modding that is so far limited to content rearrangement. A fully-working editor, *WebMakeup*, demonstrates the feasibility of this vision. Evaluation is encouraging about the potentiality of Web Modding to improve the Web experience.

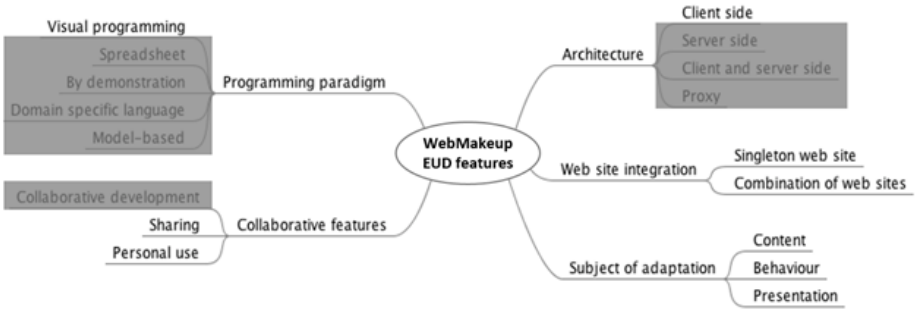

Figure 3.13: *WebMakeup End-User Development* features

WebMakeup does not require users to write a single line of code to modify Web pages. All programming is achieved through selecting DOM elements and interacting with widget decorators. Figure 3.13 highlights *End-User Development* features of *WebMakeup*.

Chapter 4

Generating Robust Locators

<hr />

4.1 Introduction

Browser extensions are plugs-in that extend the functionality of a web browser in some way [Bro]. Here, we focus on those extensions that target the Web Content, i.e. the Web Content is transcoded to change its appearance, layout or data. These downloads are accounted for hundreds of thousands. This kind of browser extensions (hereafter referred to as just "extensions") fall within the general category of code transcoding, that is, transforming content or a program on the fly to other formats. This normally implies locating the content to be transformed using *Web locators*, i.e. mechanisms for uniquely identifying an element on the Web Content [RLS+13]. Unfortunately, locators are fragile upon Web Content upgrades, i.e. changes in the layout, appearance of content of the Web page can make locators stop working. For instance, XPath expressions are a popular way to realize locators. These expressions might rely on the content and structure of the Web Content to pinpoint the target element. However, if the Web Content structure changes, the XPath expression might fail to select the appropriate target element, jeopardizing

the extension. In other words, adaptive maintenance (i.e. the one due to changes in the environment) is a main headache for extension developers.

Extensions maintenance introduces two specifics worth mentioning:

- extensions are often not developed by website owners themselves but third parties who might or might not have the resources to keep the pace with the upgrades of the underlying website,
- failures might make users give up using the extension. A large proportion of extensions aim at improving the Web Experience in terms of shortcuts and content additions. Hence, they are not essential in the sense of users not being able to accomplish their tasks, but they just save users some clicks. Here, the effort of communicating the error and re-installing the extension might dilute the benefits of the extension.

This makes robustness a key non-functional feature for browser extensions. Since locators tend to be "the weak link", locator robustness becomes a main must to preserve the enhanced Web Experiences that extensions bring. This sustains our tenant that investing in robust locators will payoff during maintenance. This work then addresses the following problem-based research question.

How can we enhance Web locator robustness despite their fragility upon Web Content upgrades?

Web Augmentation tools have a serious problem with locators and hence, *WebMakeup*. Web modifications designed with *Web Augmentation* tools are varied from static websites that are modified once a year to newspaper websites that are updated every moment. Additionally, some websites structures are updated frequently and locator robustness decrease significantly. That is why efforts have been done to enhance robustness.

Robustness is the ability of a computer system to cope with errors during execution [Rob]. Different techniques are proposed based on different aspects of the underlying Web page, namely content, attributes, structure, coordinate or visual appearance (Section 4.2) and their

robustness is compared in Section 4.3. Section 4.4 shows an improvement of common coordinate-based locators. If we start from the fact that website upgrades are a frequent event, extensions will sooner than later fail. Failure can be faced using preventive or curative approaches [Dro03]. In a preventive approach, we engineer-in robust properties that will anticipate and prevent the occurrence of defects in the first place. Redundancy is a common approach to address robustness. The human body has two kidneys: if one fails, the other will continue working, and the person will live. Here, we investigate two approaches to redundancy. First, *Kidney locators* (Section 4.5). Here, alternative code is provided that possess equivalent functionality (i.e. retrieve the same element), so that if a locator is broken, another returning the same DOM element can replace it. Second approach is based on adding contingency data, i.e. information about the nodes' ancestors so that this data can be used to regenerate the broken locator (Section 4.6).

No matter the redundancy approach, designers should be vigilant to two main drawbacks: the additional development effort and the eventual impact on performance. For Web testing, locator robustness might be a matter of maintenance whose gains should be balanced against development overheads. For browser extensions however, locator robustness could become a question of survival since developers might not enjoy maintenance allowance. Here, the aim becomes keeping the extension up and running for as long as possible, no matter the development effort. This chapter focuses on the second drawback: performance. Therefore, the introduced algorithms are judged in terms of effectiveness (i.e. locator robustness improvement) and efficacy (i.e. performance penalty).

4.2 Locators: theme & variations

A Web locator (hereafter just "locators") is a mechanism for uniquely

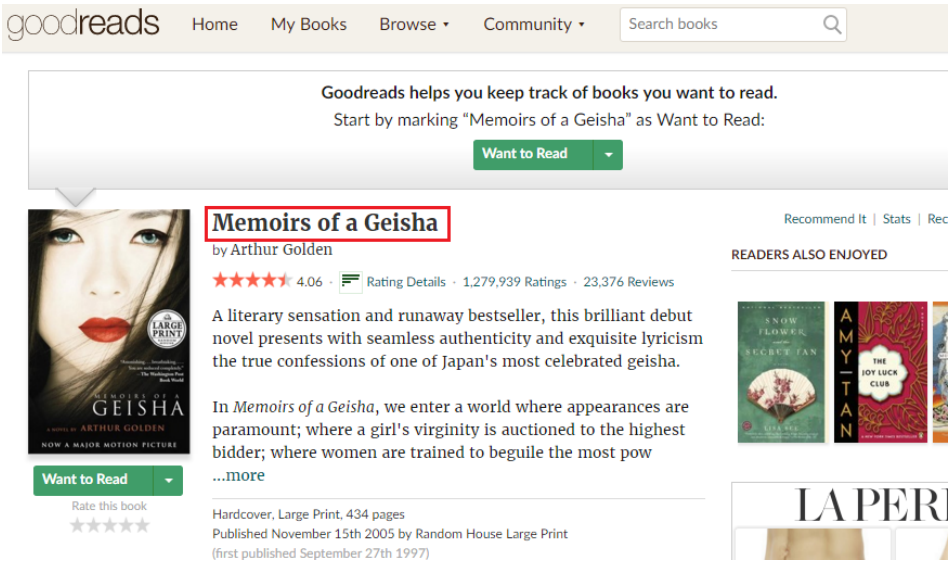

Figure 4.1: Running example: find expression that singles out the book's title in a Goodreads page.

identifying an element on the Web Content i.e. in the Document Object Model (DOM) [RLS+13]. Locators are used in a broad range of scenarios, namely, Web testing [CP08, Lad10, MBH10], Web harvesting (a.k.a web data extraction) [FMFB14], Web annotation [UCI+06], Web automation [MPR+09] or *Web augmentation* [DA15]. This ample usage of locators might well explain the different proposed techniques, namely, structure-based, coordinate-based, visual-based, attribute-based, and content-based. This section outlines these techniques along four main criteria: robustness (i.e. fragility upon DOM changes), evaluation performance (i.e. speed in identifying DOM elements), storage requirements (i.e. data to be stored) and expressiveness (i.e. ways to single out the DOM element). As a running example, we consider the Web Content in Figure 4.1. Specifically, we are interested in locating the title of the book in this *Goodreads* page. Table 4.1 outlines the comparison between different techniques to locate the title. The aim is not to be exhaustive but just to provide a glimpse of locators' diverse approaches.

	Creation time (ms)	Evaluation time (ms)	Storage (Bytes)	Expressiveness (visibility & structure)
Coordinate	1	1	16	Visible & Leaf
Structure	3-4	1	30	Visible/Hidden & Any
Visual	6	8	6200	Visible & Any
Attribute	0-1	0-1	10	Visible/Hidden & Any with attribute node
Content	1	1	40	Visible/ Hidden & Text Node

Table 4.1: Comparison of Locator Realization Techniques. *Evaluation and creation time* is being measured based on the following mechanisms: *jsDom* for structured-based [jsD], *jsDom* for attribute-based, *Hypothesis* for content-based [Hypa], *dom-to-image [dom] and Resemble [Res]* for visual-based and finally, *WebMakeup* for coordinate-based [Web]. *Expressiveness* is described in terms of the kind of node that can be located. Specifically, whether the node needs to be visible or not, and the node structure (a leaf node, a node with attributes, a text node).

Structure-based. This approach relies on the DOM structure, i.e. the locator is expressed as an XPath expression [SLRT14]. For the sample case, this expression is

//[@id="bookTitle"]*

This approach is fragile upon changes in the DOM structure, though the use of relative paths reduces considerably this risk. Expressiveness wise, the target node does not need to be visible. Different studies indicate the robustness of structure-based locators [LCRT13, LSRT14, LCRS13, LSRT15, LCRT14]. On the downside, they are one of the least performing approaches.

This locator is mainly used for *Web Augmentation* [DAA13, Web10, DSAT12, FWRG11b], web testing and web automation [LCRS13, MPR+09]. The automation means that either the method can generate the process model automatically, or the method can locate the correct services if an abstract process model is given [RS04]. Structure-based locator and

attribute locators can be used in web annotation and web extraction when all the text of a node is wanted [FB11c, FB11a, GSCJ14].

Attribute-based. These locators rely on attributes *'id', 'class'* and the like. For the sample case, these attributes include:

> *ID: "bookTitle"; Name: "bookTitle"; Attributes: "itemprop:name"; Text: "Memoirs of a Geisha"*

In stricto sensu, the attribute playing the locator role must be unique. Combination of attributes is considered structure-based locators. High robustness and performance (less than a millisecond) are the main advantages of attribute-based locators. Main drawback is that these attributes are not always available or they are not unique. Usage scenarios are those of structure-based locators with which they are jointly used.

Coordinate-based. First generation locators were coordinate locators [LCRT14, LSRT15]. This approach relies on node position, i.e. the locator is expressed as a coordinate. This makes this approach fragile upon screen resolution and font size when the element is caught. Moreover, CSS attribute changes might modify the initial position of the element. Moreover, this locator is only able to find the leaf nodes of the DOM due to the fact that in certain point of the website the system only finds a leaf node. In order to enhance this type of locator, we have implemented a new method to find all type of nodes and not only leaf nodes (see subsection 4.4).

For the sample case, the locator is:

> *[(x:325, y:195), (x:780, y:223)]*

Expressiveness wise, the approach is limited to leaf visible elements. On the upside, this approach space performance and the algorithmic complexity to find the node is low.

This approach can be mainly found in web testing and web automation [LSRT15]. This technique is deprecated owing to its weak robustness even with small changes. This worsen with AJAX websites where node

attributes are automatically generated, putting locator robustness at stake [BMM12].

Visual-based. This approach rests on image recognition [SLRT14]. They work as if web pages were photos, and next, they try to find the position of the desired fragment within this photo. For the sample case, this fragment would be:

Its robustness is similar to structure-based locators [LCRT14]. They are good for stable websites, which explains their popularity among web testing and web automation practitioners. Examples include Sikuli [YCM09] for searching and automating GUI elements, and Lixto [BFG01] for extracting web content. Moreover, some other studies [LSRT15] reveal that visual locators have been used for web testing. Nonetheless, *Web Augmentation*, web annotation and data extraction should not use visual locator because website changes are frequent or the system wants a certain node in similar websites of the same domain but with different content. Main drawbacks include the need for the node to be visible, performance and storage requirements.

Content-based. Here, Web Content is regarded as a string. Locators are then expressed as a substring delimited through two other substrings. For the sample case, this leads to the following triplet:

["irs of a Geisha by Arthur Golden", "Memoirs of a Geisha"
,"by Arthur Golden Memoirs of a Ge"]

No wonder this approach is very fragile upon changes on content. Expressiveness wise, these locators are restricted to nodes that have content, no matter whether they are visible or not. Buttons and the like with no content might not be located this way. On the other hand, they are the only ones that permit to pinpoint data within node content. This makes

Attributes/Structure	Simple	Complex
Few	*BBC*	*Goodreads*
Many	*FOX*	*Wikipedia*

Table 4.2: Website sample representatives arranged along two dimensions: attribute usage & structure complexity.

this approach very valuable for Web annotation where the content to be annotated tends to be embedded as part of a node's content [Hypb].

This locator is mainly used for web annotation and web harvesting (is the process of automatically collecting information from the Web [FMFB14]) thanks to the possibility it provides of selecting data within text node content. Pankow [CHS04] uses a range of relatively rare, but informative, syntactic patterns to mark-up candidate phrases.

4.3 Locator robustness

Web Content upgrades impact all locator approaches to a greater or lesser extent. If Web Content is upgraded, locators might need to be revised. But, which is the likelihood of Web upgrades to impact browser extensions? In other words, should you be the developer of an extension for a given website, how often would have been you forced to maintain your extension as a result of this website's upgrades? To get a glimpse, we analyse the evolution of four websites which use browser extensions: *www.fox.com, www.wikipedia.org, www.goodreads.com,* and *www.bbc.com*. These websites are selected as website representatives based on two dimensions: structure complexity and attribute usage (see Table 4.2). The rationale is that these characteristics ground most of the locator approaches. Using Wayback Machine[1], we picked up the Web Content for three-month intervals from January 2014 to March 2016. This accounts for nine Web Content samples for each of the aforementioned

[1] https://archive.org/

websites.

The experiment tested whether extensions working for the websites' versions in January 2014 would have still been working two years later. The likelihood of extensions to be broken is measured as the rate of locators, which fail to recover the node. That is, an extension is broken if any of its locators no longer pinpoints the right node. These locators are obtained at sample *Si*, and checked at sample *Si+1*. Figure 4.2 depicts the outcome for attribute locators, coordinate locators and structure locators, respectively[2]. The number of locators depends on the approach expressiveness: structure locators can pinpoint any node, coordinate locator cannot find hidden nodes, and attribute locators are limited to nodes that hold ID-like attributes.

At the onset, around 10% of locators obtained at January 2014 fail to recover the appropriate node in March 2014. This rate is reasonable since we are considering the whole set of DOM nodes. For Wikipedia, this rate keeps steady during the whole timeframe, meaning that this website did not undergo big changes. However, websites like Fox or BBC conducted main upgrades that cause a sharp decrease in the number of successful locators. It is worth noticing how diagrams tend to be quite similar no matter the locator approach being followed. This seems to imply that big DOM disruptions affect all approaches alike. For the sample cases, it is more likely that extensions on top of either the BBC website or the Fox website would require some maintenance at the risk of stop working[3].

This experiment confirms previous insights on locator robustness, i.e. that attribute-based locators and structure-based locators outperform coordinate locators as for robustness. This is aligned with previous results. Table 4.3 summarizes previous findings as partial order relationships between the different locator approaches. ID-based locators

[2]Content-based and visual locators are decidedly a bad option when confronted with frequent website upgrades, as tend to be the norm rather than the exception.

[3]In some months, the number of successful locators is more than in the previous month. This is as a result of small changes in the website that makes possible to find an element that previously had been impossible due to website upgrades.

Figure 4.2: Rate of locators successfully recovering DOM nodes upon website upgrades. Outcome for the attribute-based approach (top-left), the coordinate-based approach (top-right) and the structure-based approach (bottom). Data available at http://tinyurl.com/hud5vxn

Findings	**Supporting Evidence Reference**
structure locator < attribute locator < ID-based locator	[LCRT13, LSRT14]
structure locator < ID-based locator	[LCRS13]
visual locator < DOM-Based locator	[LCRT14]
coordinate locator < DOM-based	[LSRT15]
coordinate locator < Visual locator	[LSRT15]

Table 4.3: Robustness among locator approaches. *"A1 < A2"* indicates that evidences exist that approach *A1* is less robust than *A2* for the case studies presented at the companion reference.

are certainly the most robust among attribute locators [LCRT13, LSRT14]. Unfortunately, IDs are not always available and other node attributes should be used. Attribute-based locators are shown to be more robust than structure-based locators [LCRS13]. These locators are in turn more robust than visual locators for the cases studies in [LCRT14]. Finally, coordinate locators are the least stable ones according to the findings in [LSRT15]. The bottom line is that ID-based locators bring together the best of both worlds: easy definition and robustness. However, this technique is only available if the target element has an ID. And this is frequently not the case. Hence, fragility is inherent to Web locators. This work introduces redundancy in locator definition as the means to improve locator resilience.

4.4 Improving coordinate-based locators

Nowadays coordinate-based locators are an out-dated type of locator. Coordinate-based locator gets the node of a concrete pixel of the screen and as a consequence the node is always a leaf node. With a minimum change on the website, this node can be moved and it will extremely difficult to find the element because the node will have been moved. Stocco et al. [SLRT14] say that they were the first type of locator and now a second and a third generation of locator are being used, structure-based locators and visual-based locators. Furthermore, [SLRT14] says that coordinate-based locators are obsolete because they are remarkably fragile. However there are some scenarios in which coordinate-based locators are a good option because of the DOM changes, for example Ajax web pages because the ID can be generated automatically when the website is reloaded and structure and attribute based locators cannot find the node.

With the idea of improving this type of algorithm and with the intention of including it in the *Kidney algorithm* (section 4.5) we have developed a coordinate-based locator. Thank to our algorithm coordinate-based locators can locate any type of nodes if they are visible, they do not have to be leaf nodes. Our approach consists on a union of traditional coordinate-

based locator with a structure-based locator.

When we are defining a coordinate-based locator, we have to save the coordinates of middle point of the node in the screen an its domHash. DomHash is a string based on all tagNames of the node. The domHash string starts with the root tagName and continues with the tagName of its first child and after that with the tagName of the second children of root node. This process continues with all children of the root node. At the end, if the first child of the root node has children, we repeat the same process with all tagNames. All tagNames start with the "<" character and end with the ">" to separate all tagNames and detect them unequivocally. Additionally, we add "#" character to denote text in the node.

When we are executing the coordinate-based locator we extract the node we have in the point saved before. If this node has the same domHash, we have extracted the correct node. Otherwise we get the father of the node and we check if this node has the same domHash. We repeat the process until we get the correct node, we get the root of the webpage or until the node domHash is longer than the domHash saved in the defining process similarly [ESZ16] does. Continuing with the example of subsection 4.2, the coordinate-based locator sample would be:

[(x:325, y:195), (x:780, y:223),"<H1>#</H1>"]

4.5 Kidney locators

Kidney locators single out DOM elements by combining three different approaches: structure-based, attribute-based and coordinate-based. Hence, *Kidney locators* hold three *component locators*. Kidney locator has been implemented in *WebMakeup* [DAA+14]. If one component locator fails, others can keep delivering the right element. In addition, *Kidney locators* attempt to regenerate themselves so that the failed locator can be re-created based on those components that still work.

Visual locators are not useful for *Web Augmentation* and for that

Algorithm 4.1 Kidney Algorithm.

```
function kidney (KidneyLocator: locator[]):Node {
var strNode, attNode, coorNode = null;
strNode=KidneyLocator['structure'];
if(strNode!=null){
        KidneyLocator=updateLocators(strNode);
        return strNode;
}
else{
        attNode=KidneyLocator['attribute'];
        if(attNode!=null){
                KidneyLocator=updateLocators(attNode);
                return attNode;
        }
        else{
                coorNode=KidneyLocator['coordinate'];
                if(coorNode!=null){
                KidneyLocator=updateLocators(coorNode);
                        return coorNode;
                }
                else{
                        return null;
                }
        }
}
}
```

reason, Kidney locator has been implemented with this three approaches. Moreover, *dom-to-image [dom] and Resemble [Res]* used to implement visual locators, only the first iteration cost is extremely high to be used in a web extension (more than 5.000 ms). Next iterations will be 8 ms (see table 4.1). Something similar happens with content-based locators, they are fragile within text content changes. *Web Augmentation* can be used in web pages with frequent content changes like newspapers, blogs, TV guides, social networks, etc. and for that reason, content-based locator has not been included in *Kidney locators.* It must be taken into consideration that

structure-based locator that *Kidney locator* uses does not include unique attributes like ID because in this case, structure and attribute-based locator would be the same.

Algorithm 4.1 outlines the code. The algorithm takes a *Kidney locator* as input, and returns the desired node. Two main design decisions were taken w.r.t. (1) the order in which component locator will be evaluated, and (2), the moment when component locators are regenerated. Structure-based locator will always be available. However, attribute-based locators and coordinate locators depend on the target node having ID-like attributes or nodes being visible, respectively. As for the former, the decision is based on efficiency.

Regarding the efficiency, it is important to execute each algorithm in the best order to avoid unnecessary executions and to reduce the time needed for each node location. We have calculated (See Table 4.1) the time needed for each locators to be created and to search the node on the website. Attribute-Based locators execution varies depending on the type of the locator the algorithm is using. For example ID locators are executed in less than one millisecond but CSS locators need one millisecond. We have thought in the worst case to calculate the necessary time to locate the node.

As for second design decision, locator regeneration timing, there are two different options: (1) when a locator succeed, update the others always even if we do not know they succeed, (2) check all locators and update the locators that have failed. We have calculated the time needed considering what happens in all combination of locators when a locator works and the others fail and it has been taken into account the time needed in each order combination of execution of each locator. Based on this data and trying to balance both criteria (robustness and efficiency), Table 4.4 shows that the best order is SAP (S $\leftrightarrow$ Structure-Based locator, A $\leftrightarrow$ Attribute-Based

Updating always	Time (ms)	Updating after a fail	Time (ms)
APS	22	APS	18
ASP	18	ASP	18
PAS	21	PAS	18
PSA	18	PSA	18
SAP	15	SAP	18
SPA	16	SPA	18

Table 4.4: Comparison of different possible orders for the Kidney algorithm

locator, P $\leftrightarrow$ Pixel-Based locator) due to its lowest average time to be executed and the second option of robustness (updating always all locators after a success). Additionally, it needs the lowest time to be executed in the most common options (the probability of failure with regard to the robustness of each locator). The most common options are the cases when Structure-Based locators and Attribute-Based locators work both or one of them because these locators are the most robust.

4.5.1 Validation

Effectiveness. We analyse to what extent *Kidney locators* outperform traditional structure locators. To this end, we conduct the very same experiment that in Section 4.3 but now using *Kidney locators*. Figure 4.3 provides the output. The figures account for an approximate 10% increase in the rate of success. Unfortunately, the combine usage of different locator mechanisms was not able to survive a big disruption in the website design (e.g. BBC and Fox).

Efficiency. Performance wise, Figure 4.4 compares structure-based locators and *Kidney locators*. Kidney locator (discontinuous line) aprox. doubles the time of structure locators (continuous line). Nevertheless, for most extensions (having below 30 locators), *Kidney locators* only incur in

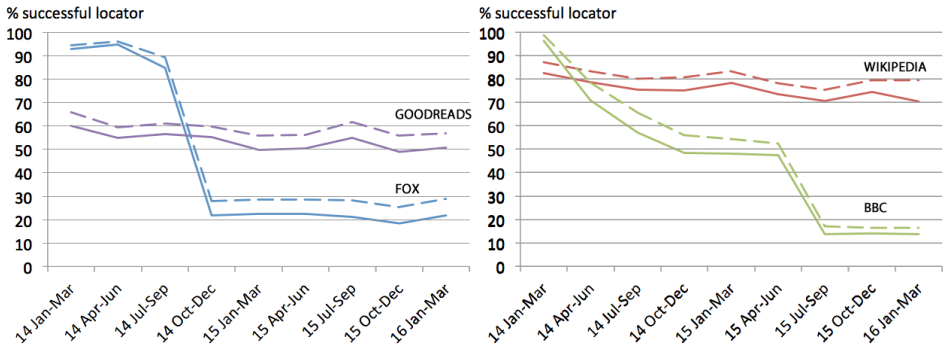

Figure 4.3: Rate of locators successfully recovering DOM nodes upon website upgrades. Structure locators (continuous line) versus *Kidney locators* (dotted line) for the four website case studies. Data available at http://tinyurl.com/hud5vxn

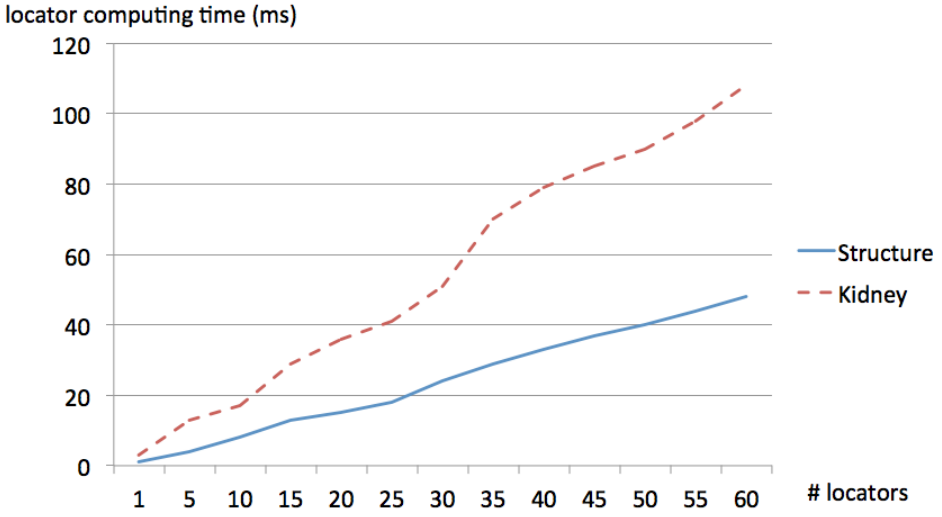

Figure 4.4: Time to recover locators. Kidney locator (discontinuous line) aprox. doubles the time of structure locators (continuous line).

a 20 ms performance plus. This penalty might be bearable if we get a 10% robustness gain in return.

4.6 Regenerative locators

Lets imagine that despite the fact that *Kidney locator* improves locator robustness, it fails and *Web Augmentation* extension stops working. This is not acceptable and hence, a new algorithm has been developed in order to avoid web extension failure, *Regenerative locator*.

A *Regenerative locator* is a structure-based locator supplemented with contingency data about the target node (i.e. the node to be located) and its ancestors. Contingency data includes: *ID, Name, CSS, Position, Text, TagName*, etc. When a locator fails, this contingency data might help to restore the broken locator.

Figure 4.5 (left) provides an example. This figure depicts the *Boston Globe* headlines on 11 March 2015: *"MIT officer's death is described during Tsarnaev trial"*. A structure locator that singles out this node would be

".//div[@id='main'] //div[1]/div/div/div[1]/div[1]/h2".

One year later, the *Boston Globe* website has a similar appearance (Figure 4.5 (right)) but its underlying structure has changed as unveiled by the DOM counterpart: an <*a*> element now appears on top of the <*h2*> element. This breaks the headline locator. This is when regenerative locators come into play. The headline locator can be regenerated by resorting to its ancestors, whose information is kept as part of the locator state (contingency data).

Regenerating means being able to single out the targeted node again. This is achieved by departing from the target element type (e.g. <*h2*>) and gradually enriching the expression with additional predicates till the XPath expression returns a single node [4]. Figure 4.6 shows a case example.

[4]This success criteria rules out locators that select multiple DOM elements. According

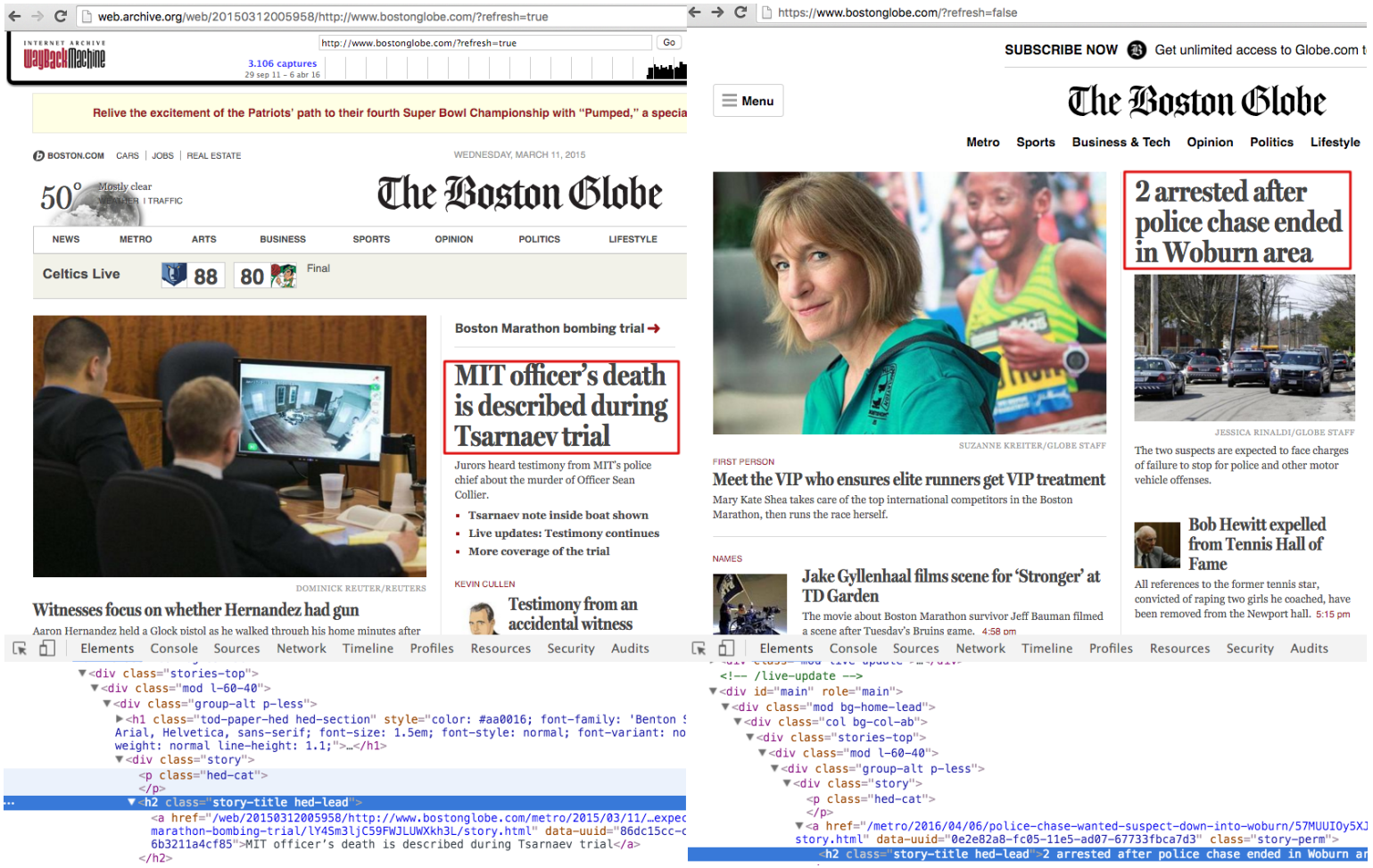

Figure 4.5: *The Boston Globe* website: screenshots and their DOM counterparts for headlines on March 2015 (left side) and March 2016 (right side). Notice the upgrade: <*H2*> moves from being a child of a <*DIV*> element to become a child of a <*a*> element.

The algorithm starts by adding attributes from the node itself. First attempt: *contains(@class, 'story-title')* that returns four nodes. Second attempt: *contains(@class, 'hed-lead')* that returns five nodes. Third attempt: *contains(@class,'story-title') & contains(@class, 'hed-lead')* that returns no node. Since the *H2* attributes alone do not single out a single node, the algorithm moves one level up: DIV. The algorithm supplements the previous expressions by adding parent-level restrictions (e.g. *DIV[contains(@class,'story-title')]*) on the hope that this additional restriction can reduce the number of possible outputs to just one. This process continues till an expression returns a single node. For our case study, this expression is the one highlighted at the bottom right-hand corner of Figure 4.5 (right).

Algorithm 4.2 shows the code. First, attributes of the desired node (ID, name, text, position...) are used to obtain a first XPath expression (*combineAttributes*). After that, the *AdaptiveXpath* algorithm links all previous XPath with the new combination of attributes using the *concatPredicateList* function. The algorithm checks all different combinations of attributes one by one through the *apply* function[5]. The *OneResultXpath* algorithm returns the node with a unique XPath result. Otherwise, if there is not XPath expression with a unique node as a result, the *deleteCombinationsWithoutResult* function deletes all combinations that have no result. That is, if an XPath expression has found no node, there is no point in adding additional predicates to this expression. After deleting those useless XPaths, the algorithm adds the different combinations of its father's attributes. The process iterates till either the XPath expression delivers a unique result or all ancestors have been considered.

to [BPM15], this kind of locators account for 22% while the other 78% correspond to locators used to select a single element.

[5]The first iteration does not consider the *ID* since this attribute already delivers a single output.

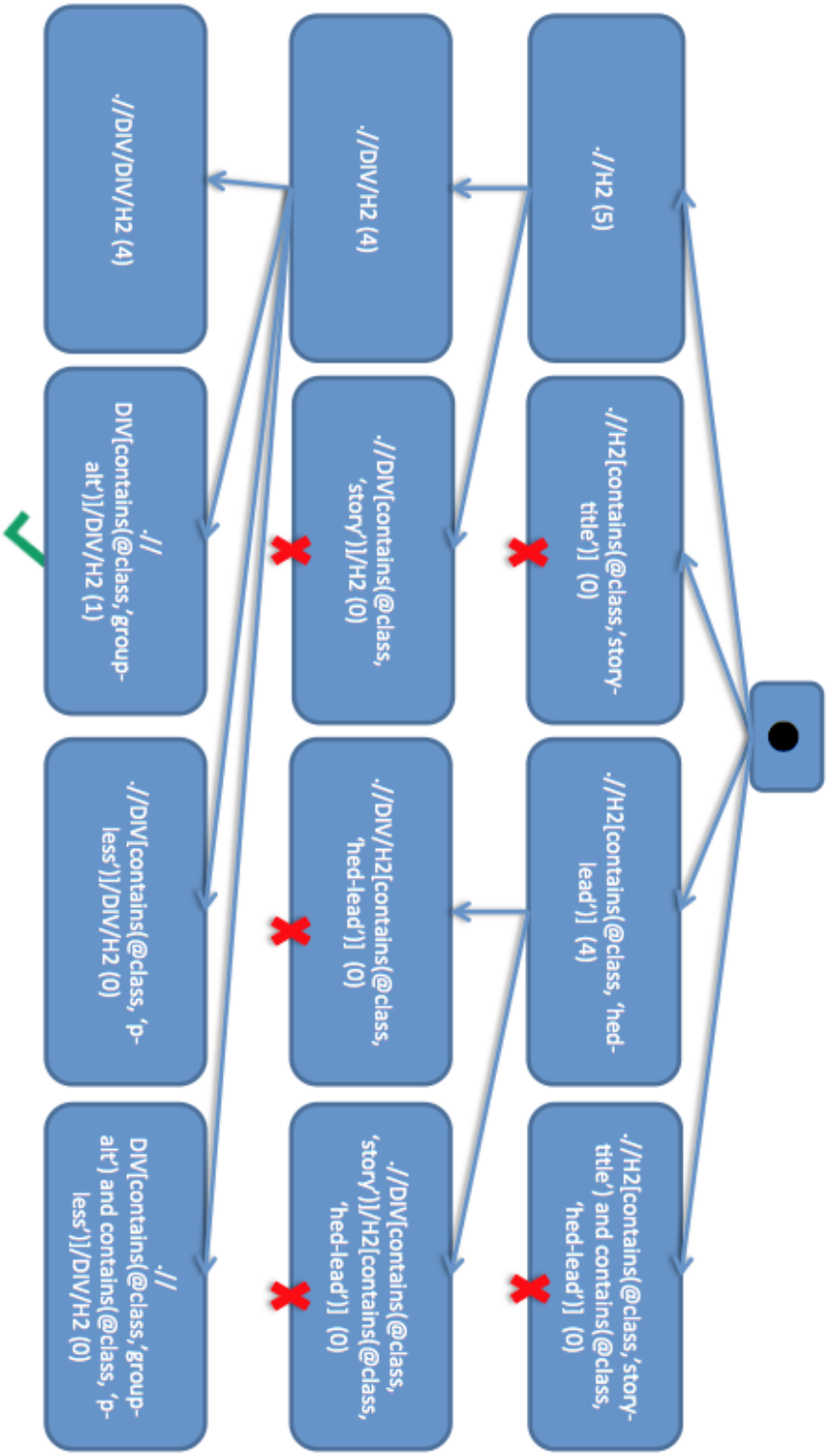

Figure 4.6: The *regenerative* algorithm at work: the type of the target node is gradually enriched with contingency data till the XPath expression delivers a single node.

Algorithm 4.2 The *regenerative* algorithm: a guess-and-check algorithm to regenerate structure-based locators based on contingency data.

```
function adaptiveXpath (ancestorList):XPath{
    var ancestorPosition = 0;
    var fullPredicateList = null;
    while(ancestorList.length > ancestorPosition){
        var currentPartialPredicates =
                combineAttributes(ancestorList[ancestorPosition]);
        fullPredicateList =   concatPredicateList(fullPredicateList,
                                                  currentPartialPredicates);
        var numberOfXpathResultList = apply(fullPredicateList);
        var validXPath = oneResultXpath(numberOfXpathResultList);
        if(validXPath!=null){
            return validXPath;
        }else{
            fullPredicateList=deleteXPathsWithoutResult(fullPredicateList,
                                                  numberOfXpathResultList);
            ancestorPosition++;
        }
    }
}
```

4.6.1 Validation

Effectiveness. We compare traditional structure-based locators (XPath expression) and regenerative locators (XPath expression + contingency data) w.r.t the experiment in Section 4.3. Figure 4.7 depicts the output. Results are encouraging since the approach seems to recover even for disruptive changes as those of the BBC website.

In addition, Table 4.5 provides specific examples as for the recall and the precision metrics. Here, we selected eight websites that enjoyed browser extensions, and took samples of the very same page every three months based on Wayback Machine[6]. For each website, we took one of its extension, and dug out its locators. Table 4.5 indicates the day the sampling started together with the initial XPath. Samplings were taking every three months till this XPath broke. This data was recorded together with the new regenerated expression returned by the algorithm. This new XPath was put to test for the successive samplings. The process repeats till last sampling. The outcome: XPath expressions failed in 15 occasions though 13 of them could be regenerated. This accounts for a 86,66% recall success. However, recall is not enough. Table 4.5 also indicates that some the regenerated XPath expressions were false positives, i.e. the XPath recovers a unique element but not the target element (false positives are shown in bold). This was the case for two XPaths. This reduces success to 73,33%.

Efficiency. The complexity of the *regenerative* algorithm is $O(m^n)$ where "m" and "n" stand for the number of attributes and ancestors, respectively. Function *combineAttributes* generates all combination of attributes. In the best case, this function is executed once. However, in the worst case, *concatPredicateList* will expand the quantity of XPath at each iteration, specifically, the growth is the number of combinations of attributes multiplied by previous attribute combination. This means that for each iteration, we combine previous combination of attributes with the

[6]Obtaining the very same page was not always easy. Some websites were rather stable (e.g. Wikipedia, Fox News) whereas others suffer more frequent updates (e.g. Amazon). This limits the moment to which we could go back taking samples.

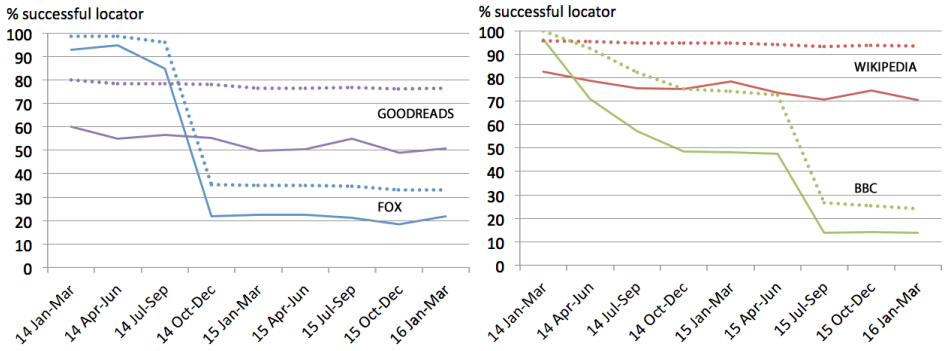

Figure 4.7: Structure-based locators (continuous line) *versus* Regenerative locators (dotted line): Regenerative locators outperform traditional locators as for locator resilience. Data available at http://tinyurl.com/hud5vxn

last iterated attributes. The *ancestorList* contains a list with all attributes of all the ancestors that an element has. The list length determines the number of times attributes will be combined. The larger the number of ancestors the element has, the longer the time spent in combining attributes. This leads to a complexity of $O(m^n)$.

Table 4.5 displays the times for the feasibility cases. The last column holds the time it tooks to obtain the regenerated XPath expression. This time depends on the number of nodes' attributes as well as the number of ancestors. The larger the number of attributes/ancestors, the longer the regeneration time. For the sample cases, this value stayed below 100 ms. Notice that this penalty is paid just in case the *Kidney locator* breaks.

4.7 Conclusions

Browser extensions make extensive use of Web locators for both anchoring and scraping Web Content. Web locators are fragile upon Web Content upgrades that can make extensions no longer pinpoint the right DOM element. This is unfortunate since extension developers cannot always afford a big maintenance investment. This makes robustness a main non-

WebPage	Initial day	Initial XPath	Day XPath Fails	Regenerated XPaths	Time (ms)
Amazon	21/09/14	//td[@id=	18/03/15	//SPAN[@id=	10
				’buyingPriceContent’]	
		"buyingPriceContent"]	05/07/15	**//TABLE[3]/TBODY /TR/TD/DIV/*/***	4426
Youtube	21/02/13	//div[@id="watch7-player"]	30/07/13	//div[@id=’player-api’]	337
IMDB	18/02/13	//td[@id="overview -top"]/div[3]	27/03/16	**//TD[2]/DIV**	47
FilmAffinity	27/07/13	//h1[@id="main-title"]/a	01/07/14	//h1[@id="main-title"]/span	3
			09/11/14	.//a/span	7
			18/01/15	//h1[@id=’main-title’]/span	8
Wikipedia	10/1/13	//table[@id="toc"]	01/08/13	//div[@id=’toc’]	623
FoxNews	01/01/13	//div[@id="section"] /div[2]/div	01/07/15	//div[@id=’big-top’]	7
BBC	01/01/13	//div[@id="promo2"]	01/07/14	//div[@id=’news’]	72
			01/01/15	//div[@id=’blq-main’] /div/div[2]/div[1]	71
			01/01/16	–	31
NYTimes	01/01/13	//*[@id= "ledePackageRegion"]	01/07/13	//div[@id=’main’]/div[2] /div[2]/div/div[1]/div[1]	88
			12/01/14	//div[@id=’main’]/div[2] /div[3]/div/div[1]/div[1]	42
			01/07/14	–	26

Table 4.5: XPath regeneration results. **Bold** is used for false positives.

functional requirement for browser extensions. This chapter explores the use of redundancy as a means to improve locator robustness. By introducing both alternative locators and additional contingency data, so called *Kidney locators* seek to make locators more robust.

First evaluations indicate an increase in robustness effectiveness (10%), though efficiency is penalized (almost double). That said, considering that extensions hardly have over 30 locators and each locator accounts for approximately 1ms, the efficiency penalty will generally be below 30ms. Moreover, the regenerative addendum might improve robustness as drawn from the case study with a 70% success in regenerating broken locators. Our hope is that these insights will prolong the correct functioning of browser extension, decreasing maintenance efforts.

Chapter 5

Conclusions

<hr />

5.1 Overview

This dissertation addresses web *personalization* through *End-User Development* for *Web Augmentation* thanks to a Chrome browser extension. Furthermore, it provides a robust system to ensure that augmentations designed by end-users will be executed correctly for long in spite of website upgrades.

In this chapter the main results are reviewed, their limitations are stated and new areas for future research are suggested.

5.2 Results

This dissertation defines a solution to end-users to personalize the Web. Specifically:

- Chapter 2 revisits the concept of *Web Augmentation* and provides an overview of the main features that characterize *Web Augmentation* technologies. In this study, some of the most important *End-User Development* and *Web* conferences were examined to conduct the

concept of *Web Augmentation* and provides an overview of the main features that characterize *Web Augmentation* technologies. Despite all their advantages, *Web Augmentation* technologies present some limitations that might result in challenges on the user side. These aspects are also presented and discussed highlighting directions for future work in that domain.

- Chapter 3 advocates for empowering people to tune the Web for their own purposes. We resort to a Visual Programming approach to empower end-users to augment a website by removing, moving and adding content from other sites to avoid scrolling, cut and pasting, tab switching and clicks. No collaboration of the websites is required. These ideas are borne out through the *WebMakeup*, a client-side Chrome browser extension. Being an end-user tool, evaluation is conducted through a set of usability experiments.

- Chapter 4 describes different web locators techniques that are used to find web nodes. Moreover, two different locators (*Kidney locator* and *Regenerative locator*) are described in order to enhance locator robustness. *Kidney locator* employs different web locator techniques until one of these techniques have a positive result and it updates the information needed to retrieve a web element when one of these techniques have failed. If *Kidney locator* fails, *Regenerative algorithm,* a structure-based locator supplemented with contingency data about the target node and its ancestors, reacts to help to restore the broken structure-based locator. Furthermore, the aim of coordinate-based locator improvement has been to use them to find any type of web node and not only leaf nodes. All this reduce browser extensions maintenance cost and avoid users give up using extensions due to location failures.

5.3 Publications

Part of the work presented in this thesis have been already presented and discussed in distinct peer-reviewed forums. The author has contributed to the following publications:

Selected Publications

- Oscar Díaz, Cristóbal Arellano, Iñigo Aldalur, Haritz Medina, Sergio Firmenich. End-user browser-side modification of web pages. In Web Information Systems Engineering – WISE 2014 – 15th International Conference, Thessaloniki, Greece, October 12-14, 2014, Proceeding, Part I (2014), pp. 293-307. Related to chapter 3.

- Iñigo Aldalur, Marco Winckler, Oscar Díaz and Philippe Palanque. Web Augmentation as a Promising Technology for End User Development, In Fabio Paternò and Volker Wulf, ed., New Perspectives in End-User Development (Springer, 2017). Related to chapter 2.

- Iñigo Aldalur, Oscar Díaz. Buildng Robust Web Locators Through Redundancy: A Case for Browser Extensions. In Proceedings of the 9th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2017, Lisbon, Portugal, June 26-29. Related to chapter 4.

International Conferences/Workshops

- Oscar Díaz, Iñigo Aldalur, Cristóbal Arellano, Haritz Medina, Sergio Firmenich. Empowering Users to Mod Websites. Rapid Mashup Development Tools – First International Rapid Mashup Challenge, RMC 2015, Rotterdam, The Netherlands, June 23-26, 2015, Revised Selected Papers, pp. 82-97

- Oscar Díaz, Iñigo Aldalur, Cristóbal Arellano, Haritz Medina, Sergio Firmenich. WebMakeup: An End-User Tool for Web Page Customization. XIX Jornadas en Ingeniería del Software y Bases de Datos (JISBD 2014), Cádiz, Spain, September 16-19, 2014

5.4 Research Stage

One of the outstanding benefits of performing a Ph.D. is the possibility of working together with international and well-regarded professionals, and above all, learning from them. The author visited the group *ICS* from the *University Toulouse III Paul Sabatier,* in France, leaded by *professor Philippe Palanque*. The author was under the supervision of *Dr. Marco Winckler* from April to June of 2015. The benefits of the visit have been at least two. On one hand, being with highly qualified people makes learning about different topics, or realize about different perspectives of the topics you are familiarized with. On the other hand, it makes learning about different working habits, methodologies and ways to organize and interact. During this research, the author acquired a deepen knowledge in online communities which influenced the content of the Chapter 2.

5.5 Assessment and Future Research

In this dissertation, the author proposes three scenarios where *End-User Development* for *Web Augmentation* has been conducted. During the development of the solution some limitations were detected. Such limitations mark the direction of future work.

Study about End-User Development features for Web Augmentation and Mashups

- Including more references from different resources. In this study, articles from some *End-User Development* and *Web* conferences

were taken into account. Journal articles and other conferences have not been considered in this study. Despite the fact that main characteristics have been extracted, it might be interesting to enhance the study including more references from different sources.

Empowering people to customize web content

- Enlarging *WebMakeup* expresiveness. Most websites have a huge amount of information and users yearn to avoid unnecessary content. This quantity of information complicates the possibility to find the needed content and the user spends for a long time searching for it. For that reason, if a user adds new content, *WebMakeup* would avoid showing unnecessary information based on variable information adding a condition to show only information when the widget has more than certain value.

- End-users cannot modify the style of the widget and they are not able to add functionality to widgets. That is why widgets with advanced behaviour can be added to *WebMakeup* functionality in order to solve the problem such as, filtering information based on a criteria, creating cropped widgets that *WebMakeup* is not able to create due to the fact that some functionalities cannot be maintained, APIs supply with information that cannot be obtained from websites and *WebMakeup* does not access to Web APIs and it is not supported to update widgets from websites in which the user has to be logged. These widgets might be developed by programmers and uploaded to a web repository to be downloaded and just used by end-users.

Enhancing web locator robustness

- Evaluation of *Kidney locator* and *Regenerative locator* must be improved due to the fact that they have not been tested with a large number of websites and browser extensions. With regard to *Regenerative locator*, browser extensions we used to check it were

Web Augmentation extensions that utilize a single locator. It was extremely difficult to find this type of extension and moreover, it was more difficult to test them using The WayBack Machine website because some websites are not saved in the The WayBack Machine.

- *Regenerative locator* sometimes generates false positive results. It is important to research how these results can be detected and avoid them.

- Improving locator generation. The study of GUI wizards (e.g. Firepath [Fir], XPath Helper [Xpab], XPath Checker [Xpaa]) and Programming by Demonstration techniques (e.g. LED [BPM15]) to facilitate the obtaining of *Kidney locators*, and hence alleviating their development overhead.

5.6 Conclusions

End-User Development for *Web Augmentation* has proven to be an adequate paradigm to improve the satisfaction of end-users. It is impossible for webmasters to foresee, develop and maintain their websites with content to satisfy everybody. *End-User Development* for *Web Augmentation* alleviates this situation by allowing end-users to make their own adaptations.

This dissertation presents three scenarios. The first scenario presents a summary of *Web Augmentation* techniques for *End-User Development*. The second scenario presents a Chrome browser extension for *Web Augmentation* for end-users. In the third scenario different locator mechanisms are mentioned and additionally, two different locators are included to enhance locator's robustness. Part of the content of this thesis has already been presented in different venues. To conclude, the author enumerated some limitations of the work, which can serve as future lines of research.

Appendix A

Evaluation test

WebMakeup: your Web, your decision

This is an experiment to collect information about your WebMakeup tool opinion which aim is to facilitate end-users to reorganize Web content. We express our gratitude for your collaboration.

The experiment consist of two different parts:

- mandatory part (in the lab): watch a video, 4 tasks and a brief questionnaire
- optional part (at home): 1 task and a brief questionnaire

Before anything:

- Watch this video: https://vimeo.com/146228732
- Install WebMakeup[1]: you can search in Google "Chrome store webmakeup"

[1] https://chrome.google.com/webstore/detail/webmakeup/alnhegodephpjnaghlcemlnpdknhbhjj

Mandatory part

First, you have to access to the questionnaire[2]

PLEASE, FILL IN SECTION 1 OF THE QUESTIONNAIRE RELATED TO THE DEMOGRAPHIC DATA

Task 1

- Before starting, write the beginning time:
- In the browser, access next Web page: http://www.donostia.eus/
- Click on Webmakeup browser icon and select "New" in the menu
- The task should carry out next:
  - Delete the upper element of the right column
  - Move "Tramites destacados" from the lower part to underneath notice section
- When you have it ready, you can see the result clicking on "Deploy" in the WebMakeup browser menu
- You can see the result in the figure A.1
- Write down the time at the end of the task:

PLEASE, FILL IN TASK 1 DATA IN SECTION 2

[2]https://docs.google.com/forms/d/198R02VsR-E2E1Imbw-AYeNO7yiYEaazAFnwXQpquPGk/viewform

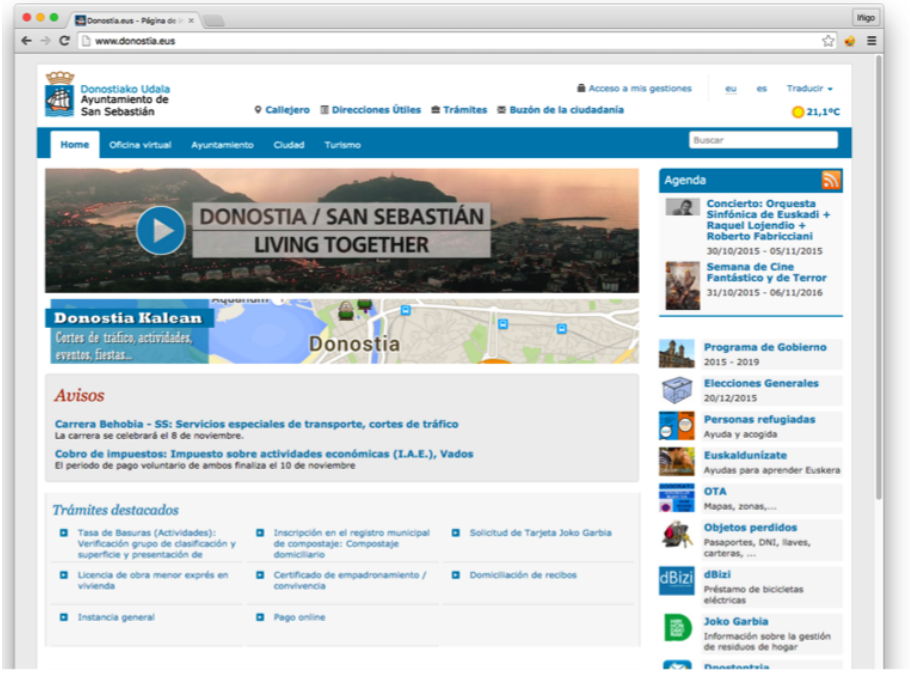

Figure A.1: donostia.eus website after the first task

Task 2

- Before starting, write the beginning time:
- You should continue in the same website: http://www.donostia.eus/
- Click on WebMakeup browser icon and select “CarryOn” in the menu
- This task have to carry out next:
  - Extend PiggyBank tab and insert a link
  - Insert http://www.kursaal.eus/es/ URL in the link
  - Insert “Kursaal” as a text of the link
  - Widgetize the upper menu of the website

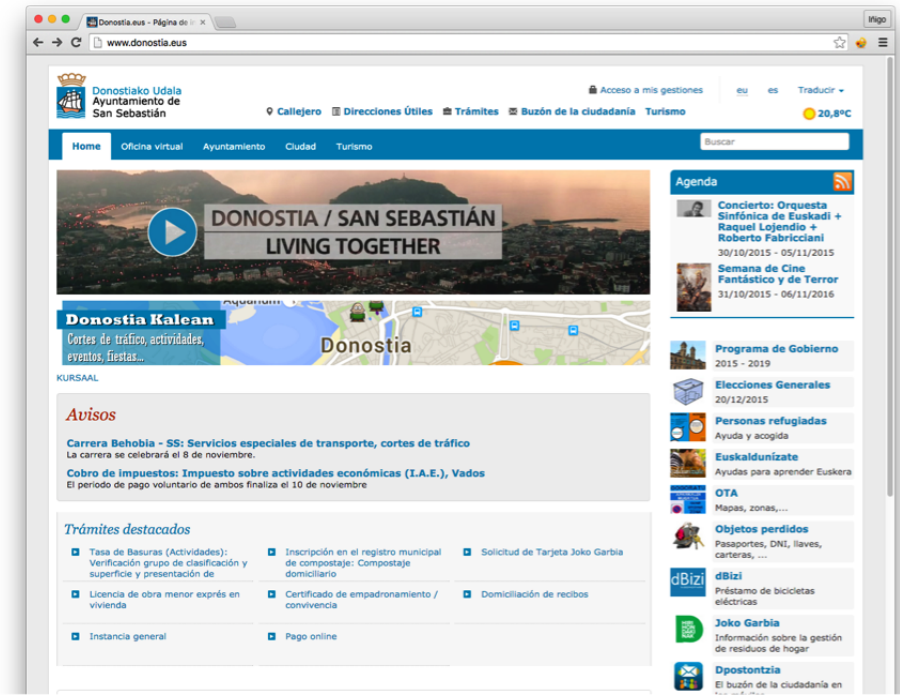

Figure A.2: donostia.eus website after the second task

  - Double click on it
  - Add a new element inserting the http://www.sansebastianturismo.com/es/ Web direction
  - Insert “Turismo” as a text
  - Double click on the widget and the element will be inserted in the menu

- When you have it ready, you can see the result clicking on “Deploy” in the WebMakeup browser menu
- You can see the result in the figure A.2
- Write down the time at the end of the task:

PLEASE, FILL IN TASK 2 DATA IN SECTION 2

Task 3

- Before starting, write the beginning time:
- In the browser, access to http://www.pesa.net/pesa/es/horarios website
  - Select a date
  - Select Arrasate as a origin city
  - Select Donostia as a destination city
- We are going to create a widget in the Pesa website in order to look up bus schedule information. Click with the right button of the mouse and select "MineIt". Select the timetable element like in figure A.3 (use the keyboard if it is necessary) and give "Pesa" name to the widget
- In the browser, access next Web page: http://www.tvguia.es/
- Click on Webmakeup browser icon and select "New" in the menu
- The task should carry out next:
  - Extend PiggyBank tab and insert "Pesa" widget
  - Extend PiggyBank tab and insert a button
  - Hide "Pesa" widget
  - Join both widget with a blink
- When you have it ready, you can see the result clicking on "Deploy" in the WebMakeup browser menu
- You can see the result in the figure A.4
- Write down the time at the end of the task:

PLEASE, FILL IN TASK 3 DATA IN SECTION 2

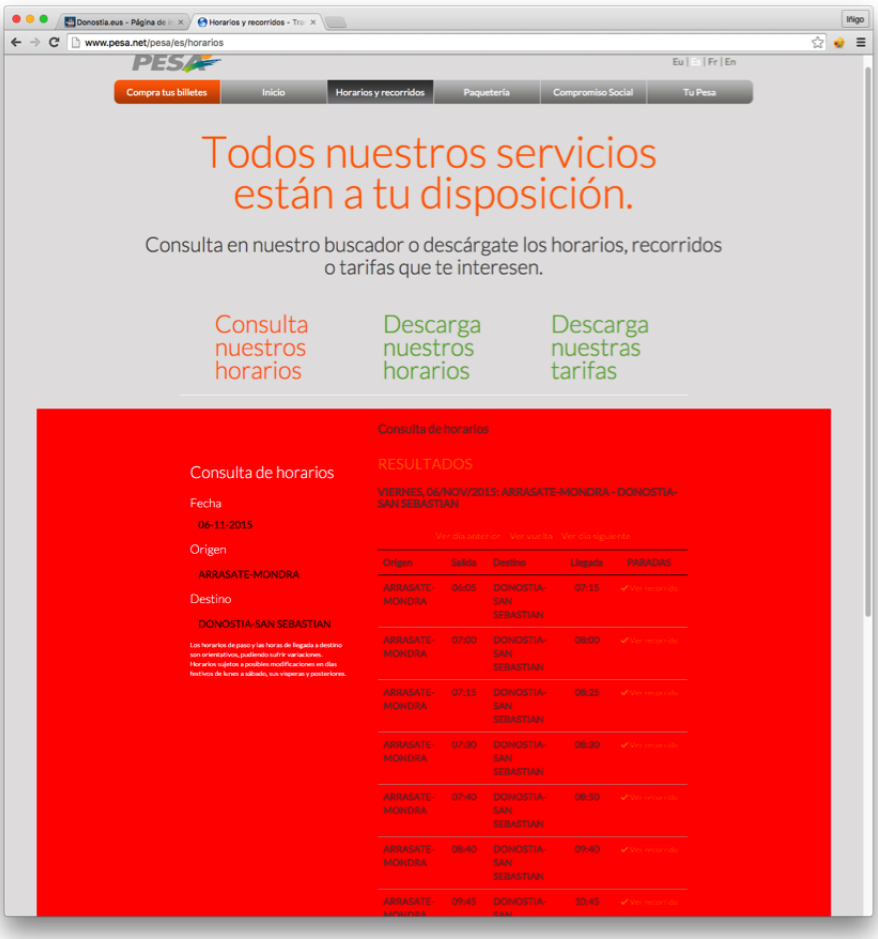

Figure A.3: Creating a widget in "Pesa" bus company website

Task 4

- Before starting, write the beginning time:
- Click on WebMakeup browser icon and select "CarryOn" in the menu
- Open a new tab in the browser and access a weather website[3]

[3] http://www.eltiempo.es/donostia-san-sebastian.html

Figure A.4: Pesa timetable in the tvguia.es website

  - Create a widget with the time of tomorrow at 13:00 (see figure A.5)

- Open a new tab and access to filmAffinity[4] website
  - Search whatever film
  - Create a widget with the search bar and the scoring element of the film (insert the same name to both widgets) (see figure A.6)

[4] www.filmaffinity.com

Figure A.5: Creating a widget in a weather web page

- Click on www.tvguia.es tab
- The task should carry out next based on task 3:
  - Delete the blink between the button and “Pesa” widget
  - Delete the Button
  - Extend PiggyBank tab and insert “filmAffinity” widget
  - Double click on “filmAffinity” widget, copy the title of the film in the main element of the website and paste it in the widget input. In order to end the binding process, double click on the widget
  - Extend the PiggyBank tab and insert the “weather” widget
  - Double click on “weather” widget
  - Choose “Day” option in the “Poly frequency” menu
  - Double click again in the “weather” widget

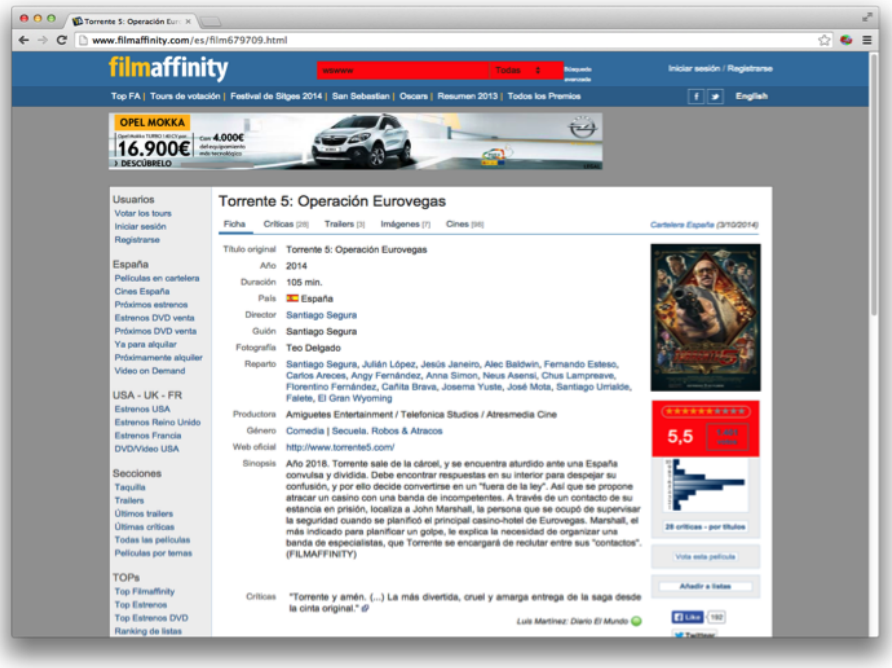

Figure A.6: Creating a widget in filmAffinity website

  - Widgetize "Esta noche destacamos..." main element in the upper part of the website (Recommendation widget)
  - Establish a CONJUNCTION blink pattern between (1) "recommendation" (2) "filmAffinity" (3) "weather" (4) "Pesa" (keep click "ALT" keyboard button during the process and select the widgets in the order mentioned)
  - Delete "Pesa" widget (the widget and the blink will disappear)

- When you have it ready, you can see the result clicking on "Deploy" in the WebMakeup browser menu
- You can see the result in the figure A.7
- Write down the time at the end of the task:

PLEASE, FILL IN TASK 4 DATA IN SECTION 2

Figure A.7: FilmAffinity and weather widgets in tvguia website

PLEASE, FILL IN SECTION 3 AND SECTION 4

Optional part

Now, it is your turn. Think about your Web routines. Are there any of them that could be of benefit to WebMakeup modding? If it comes to your mind, implement it and let us know your opinion about it answering a questionnaire[5].

We hope this experience has been interesting for you. We sincerely thank you for your participation owing to without it, we would not know if this tool is useful. We would be happy to answer any questions you may have about WebMakeup. Do not hesitate to contact us. Our email is inigo.aldalur@ehu.eus

[5]https://docs.google.com/forms/d/1D-ZPGALH6erP-crwilSKwx9g4tqfFZcSLW76t73QLFo/viewform

WebMakeup evaluation questionnaire

A.1 General Information

1. Gender.
   (a) Female.
   (b) Male.
2. Age
   (a) Less than 20
   (b) Between 20 and 39
   (c) Between 40 and 59
   (d) More than 60
3. Indicate how often you use *edition programmes*
   (a) At least once every week
   (b) At least once every month.
   (c) At least once every year.
   (d) Never.
4. Have you ever used a browser extension?

   (a) Yes

   (b) No

5. How many browser extensions have you installed during the last year?

6. How many websites do you visit every day?

   (a) 1-5

   (b) 6-10

   (c) 11-20

   (d) More than 20

7. On average, how many tabs do you have opened simultaneously?

   (a) 1

   (b) 2-5

   (c) 6-10

   (d) More than 10

8. How much time do you spend surfing the Internet in your work?

   (a) Less than 30 minutes

   (b) Between 30 and 1 hour

   (c) Between 1 hour and 2 hour

   (d) More than 2 hour

9. How much time do you spend surfing the Internet in your free time?

   (a) Less than 30 minutes

   (b) Between 30 and 1 hour

   (c) Between 1 hour and 2 hour

(d) More than 2 hour

10. Have you ever studied any programming language?

    (a) No, I have not got any programming experience
    (b) Yes, for less than a year
    (c) Yes, for more than a year

11. Have you ever visited www.tvguia.es website?

    (a) Once a week
    (b) Once a month
    (c) Once a year
    (d) Never

12. Have you ever visited www.donostia.eus website?

    (a) Once a week
    (b) Once a month
    (c) Once a year
    (d) Never

A.2 Time needed to fulfil the tasks

A.2.1 First task

1. What time did you started the task?
2. What time did you finished the task?

A.2.2 Second task

1. What time did you started the task?
2. What time did you finished the task?

A.2.3 Third task

1. What time did you started the task?
2. What time did you finished the task?

A.2.4 Fourth task

1. What time did you started the task?
2. What time did you finished the task?

A.3 Efficacy

1. Mark all the tasks you have finished with
   (a) First task
   (b) Second task
   (c) Third task
   (d) Fourth task

A.4 Usefulness

1. When I visit a website frequently I have to scroll to find the content I want on the screen(1- Totally disagree, 5 - Totally agree).
2. To scroll a website is awkward(1- Totally disagree, 5 - Totally agree).
3. When I use a browser I have to change the tab frequently (1- Totally disagree, 5 - Totally agree).
4. Tabs-switching is awkward (1- Totally disagree, 5 - Totally agree).
5. Removing content from websites improves my Web experience (1- Totally disagree, 5 - Totally agree).

6. Moving content from websites improves my Web experience (1- Totally disagree, 5 - Totally agree).

7. Inserting my own links into a single website improves my Web experience(1- Totally disagree, 5 - Totally agree).

8. Collecting content into a single website improves my Web experience(1- Totally disagree, 5 - Totally agree).

9. Parameterizing content from websites improves my Web experience(1- Totally disagree, 5 - Totally agree).

10. Showing and hiding elements improves my Web experience(1- Totally disagree, 5 - Totally agree).

11. Collecting content into a single website improves my Web experience(1- Totally disagree, 5 - Totally agree).

12. Reducing the need for going back and forth between browser tabs improves my Web experience (1- Totally disagree, 5 - Totally agree).

13. I plan to use WebMakeup in the future(1- Totally disagree, 5 - Totally agree).

14. I find WebMakeup interesting enough to tell to my friends(1- Totally disagree, 5 - Totally agree).

A.5 Usability

1. It was easy for me to widgetize fragments out of the donostia.eus web page(1- Totally disagree, 5 - Totally agree).

2. It was easy for me to delete elements of the donostia.eus web page(1- Totally disagree, 5 - Totally agree).

3. It was easy for me to move elements of the donostia.eus web page(1- Totally disagree, 5 - Totally agree).

4. It was easy for me to insert link or buttons in the donostia.eus web page(1- Totally disagree, 5 - Totally agree).

5. It was easy for me to change/add a new value to a existing element in the donostia.eus web page(1- Totally disagree, 5 - Totally agree).

6. It was easy for me to drag&drop widgets from the piggy-bank tab to the desired position in the web page.(1- Totally disagree, 5 - Totally agree).

7. It was easy for me to create widgets out of existing pages (filmAffinity)(1- Totally disagree, 5 - Totally agree).

8. It was easy for me to create widgets out of existing pages (eltiempo.es)(1- Totally disagree, 5 - Totally agree).

9. It was easy for me to create widgets out of existing pages (Pesa)(1- Totally disagree, 5 - Totally agree).

10. It was easy for me to set a simple-blink among selected widgets(1- Totally disagree, 5 - Totally agree).

11. It was easy for me to bind widgets together (i.e. feeding widgets parameters from data in the TVguia web page).(1- Totally disagree, 5 - Totally agree).

12. It was easy for me to set a blink pattern among selected widgets(1- Totally disagree, 5 - Totally agree).

13. It was easy for me to create a new makeup design(1- Totally disagree, 5 - Totally agree).

14. It was easy for me to deploy a makeup design(1- Totally disagree, 5 - Totally agree).

15. It was easy for me to restore an existing makeup design(1- Totally disagree, 5 - Totally agree).

16. During the use of WebMakeup, I have always known how to do the things I was required to do(1- Totally disagree, 5 - Totally agree).

17. There have been NO errors during the use of WebMakeup(1- Totally disagree, 5 - Totally agree).

18. WebMakeup is fast enough(1- Totally disagree, 5 - Totally agree).

19. In general, I am satisfied with the things that I have made with WebMakeup(1- Totally disagree, 5 - Totally agree).

20. We appreciate any comments, suggestions, etc. that you wish to include.

Appendix B

WebMakeup examples

Example spectrum

Table B.1 shows different examples that illustrate all the widgets and blinks spectrum of possibilities *WebMakeup* offers.

EXAMPLE	pre-set widget	hostBased widgets	Cropped widgets	Simple widgets	Complex widgets	Blink/ Pattern
DBLP-Scholar					✓	
Ecobolsa-Bolsa Madrid		✓			✓	✓
EHU-Pesa	✓		✓			✓
Linguee-Wordreference	✓				✓	✓
EasyChair	✓					
NYT-Weather				✓		
Open Science Framework		✓				

Figure B.1: WebMakeup example spectrum table

DBLP-Scholar

This example uses a complex widget to show the information from Google Scholar of the authors the user looks for in the DBLP website.

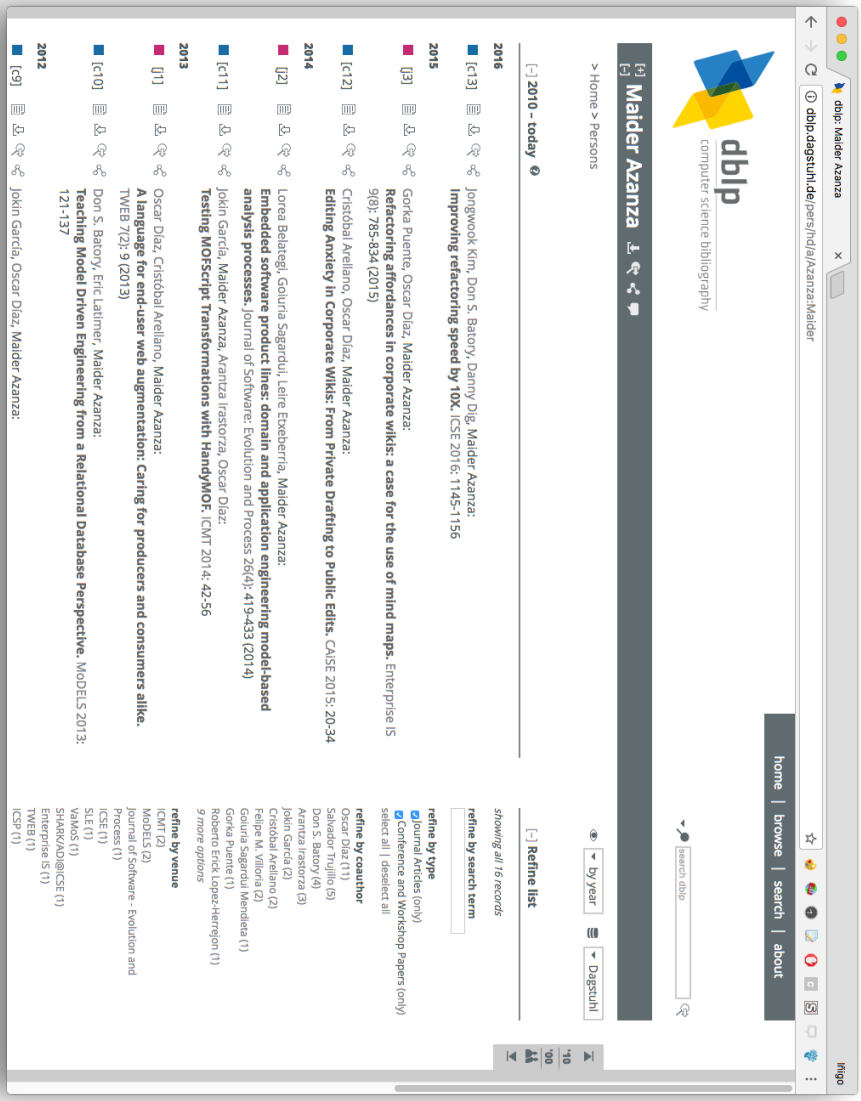

Figure B.2: DBLP

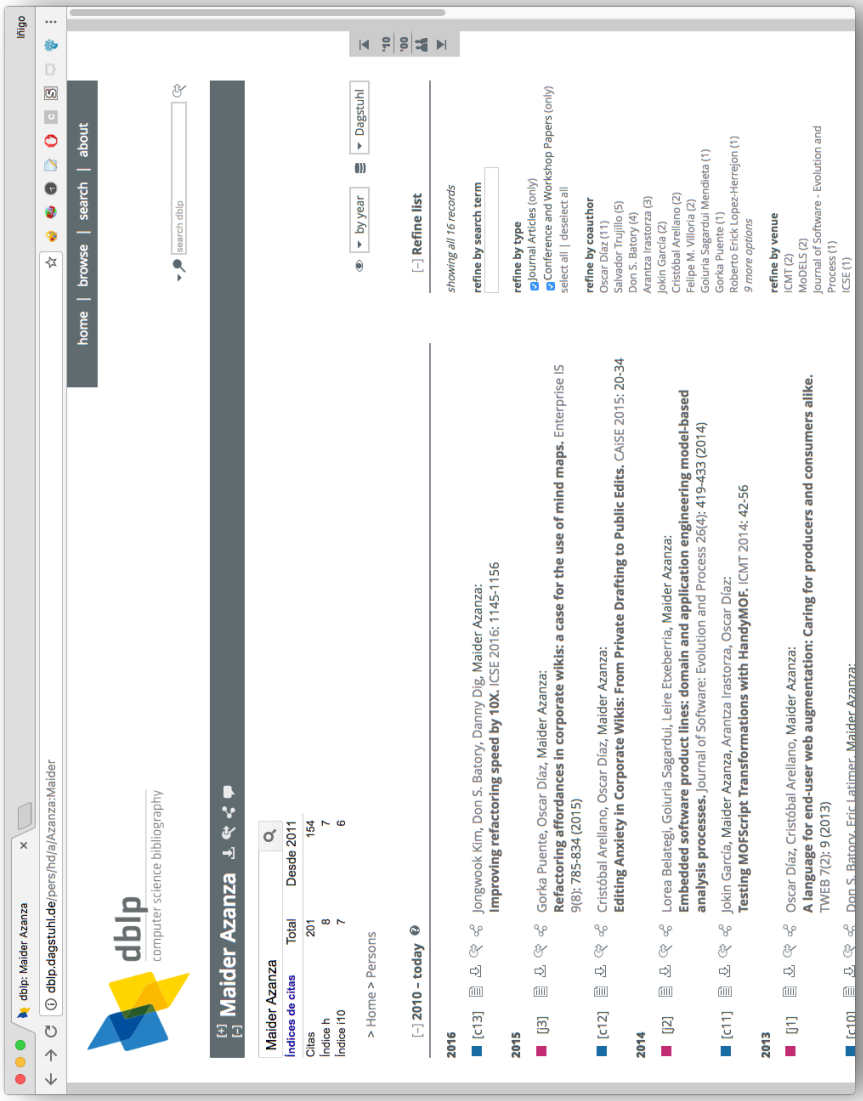

Figure B.3: DBLP after Google Scholar addition

Ecobolsa-Bolsa Madrid

This example employs a complex widgets to show more information from the Spanish stock exchange that the host website does not provide.

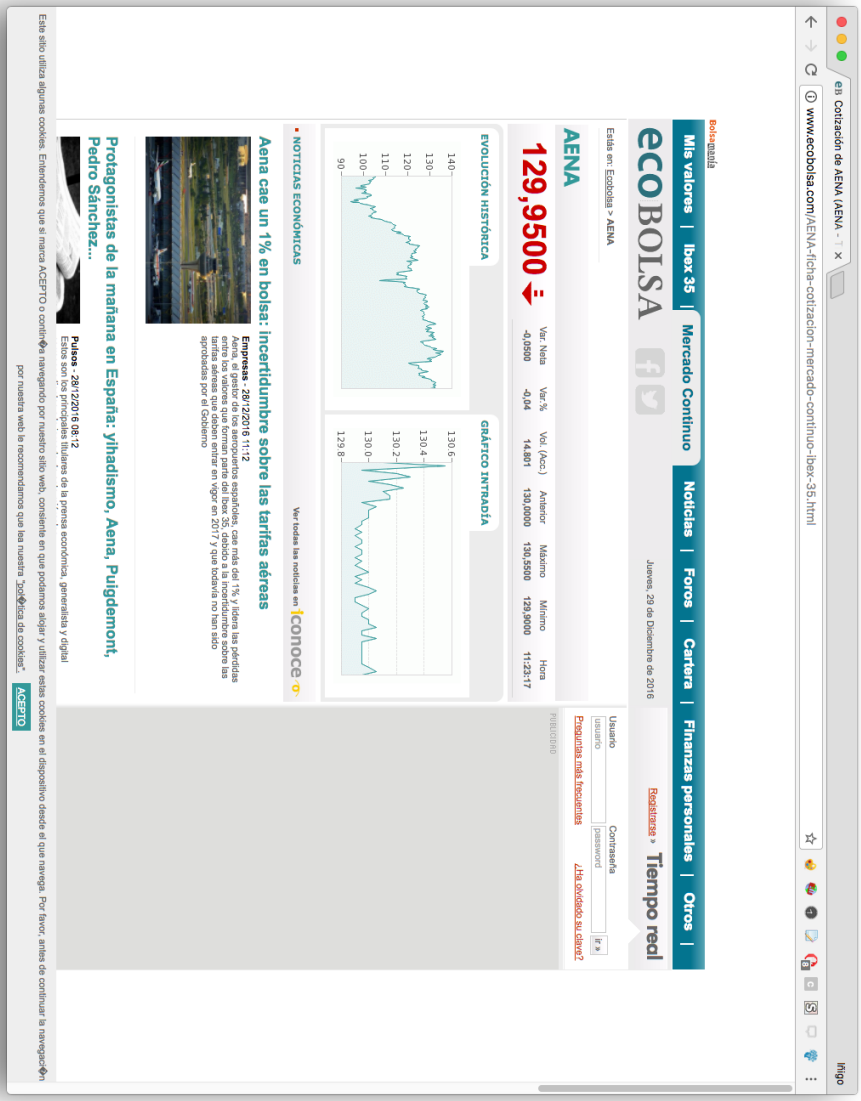

Figure B.4: Ecobolsa

Additionally, it makes use of a blink to join the company (host based

widget) with the complex widget to see the extra information only when the user desires.

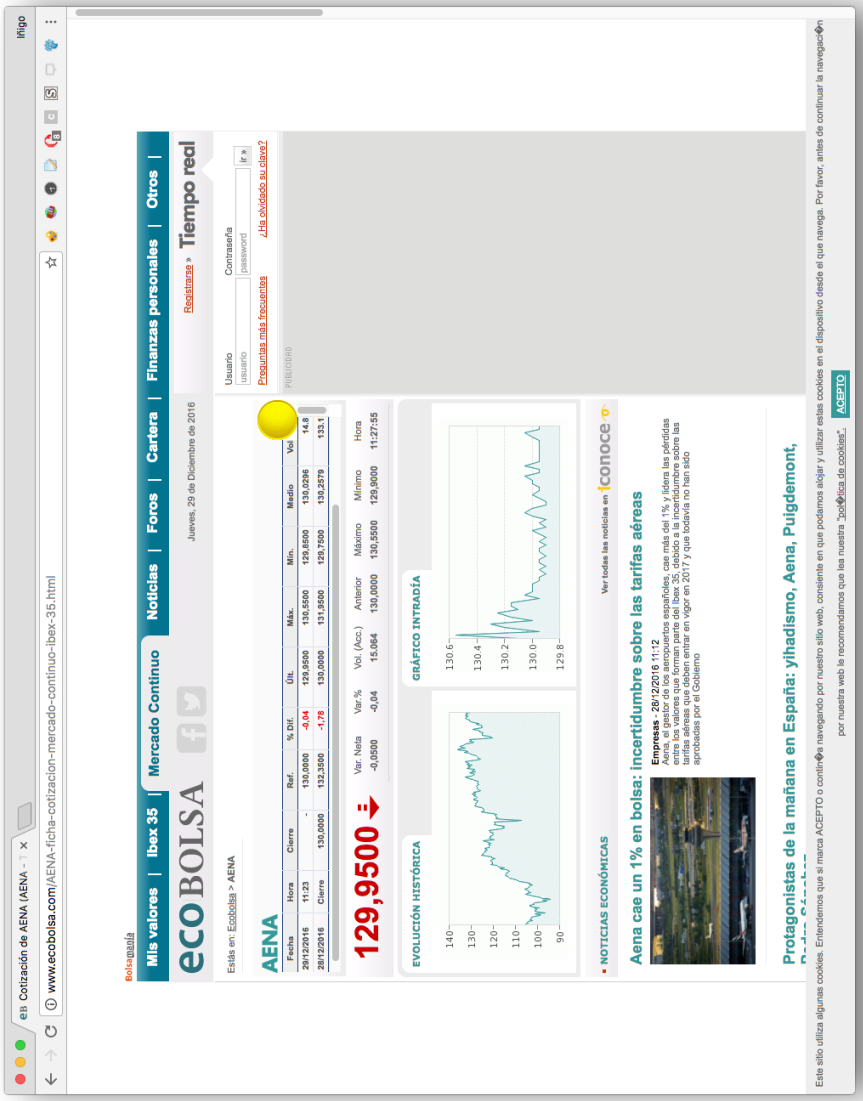

Figure B.5: Ecobolsa after Bolsa Madrid addition

EHU-Pesa

This example uses a cropped widget to display information Pesa (Basque bus company) timetable in the University of the Basque Country website.

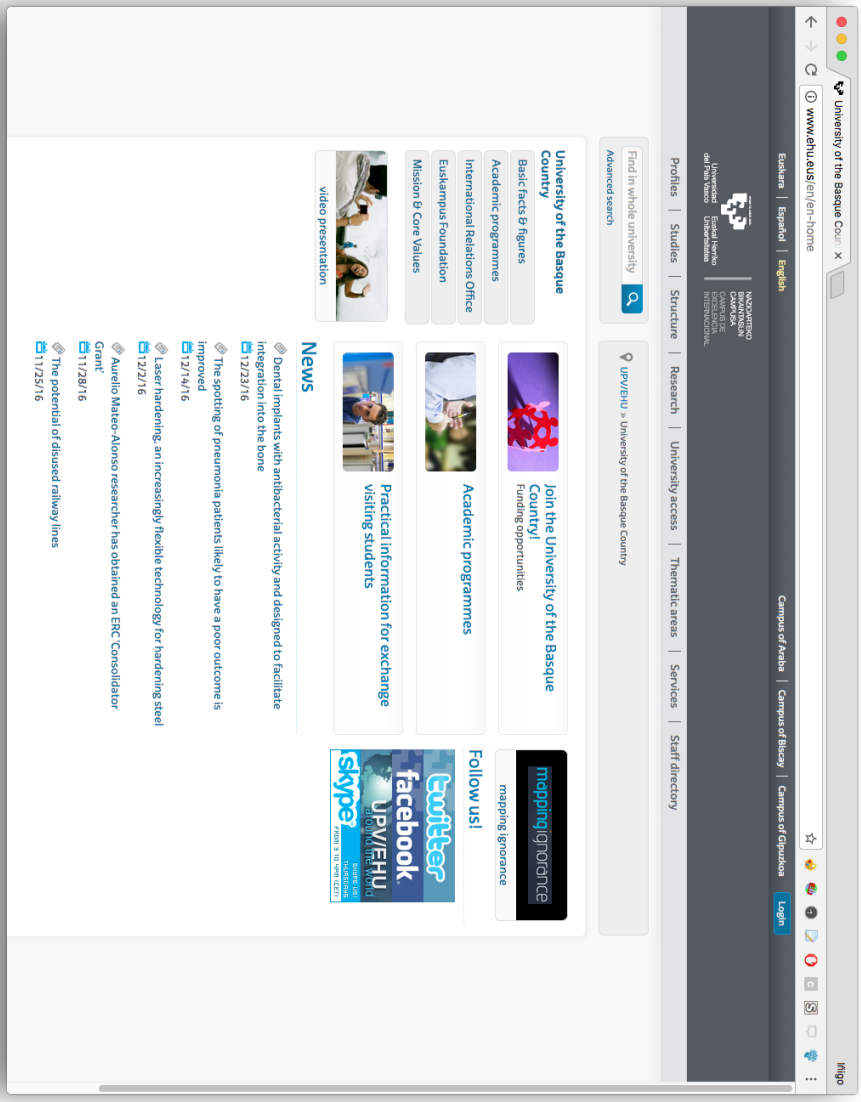

Figure B.6: EHU

Moreover, a button (pre-set widget) is added and blinked to Pesa widget

to show the widget only when the user needs it.

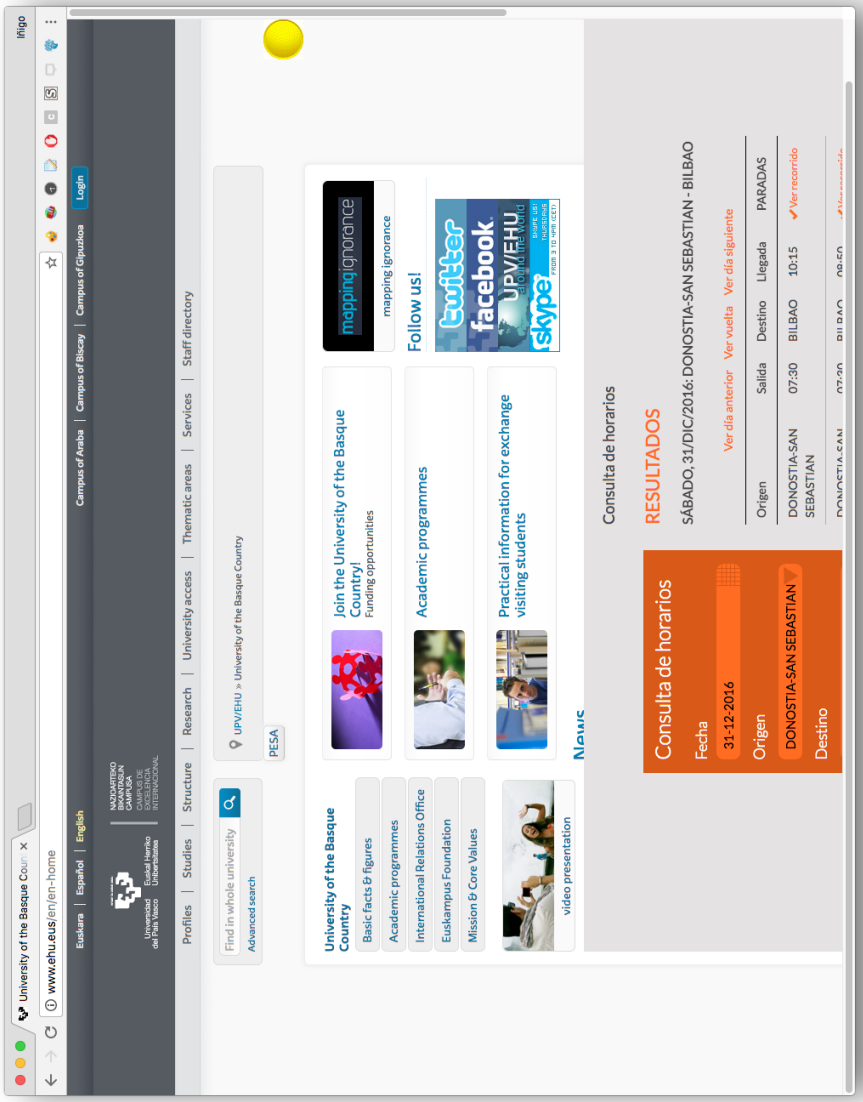

Figure B.7: EHU after Pesa addition

Linguee-WordReference

This example uses a complex widget to add more definitions to Linguee website from Wordreference web page. A button is added and blinked to show the information only when the user wants it.

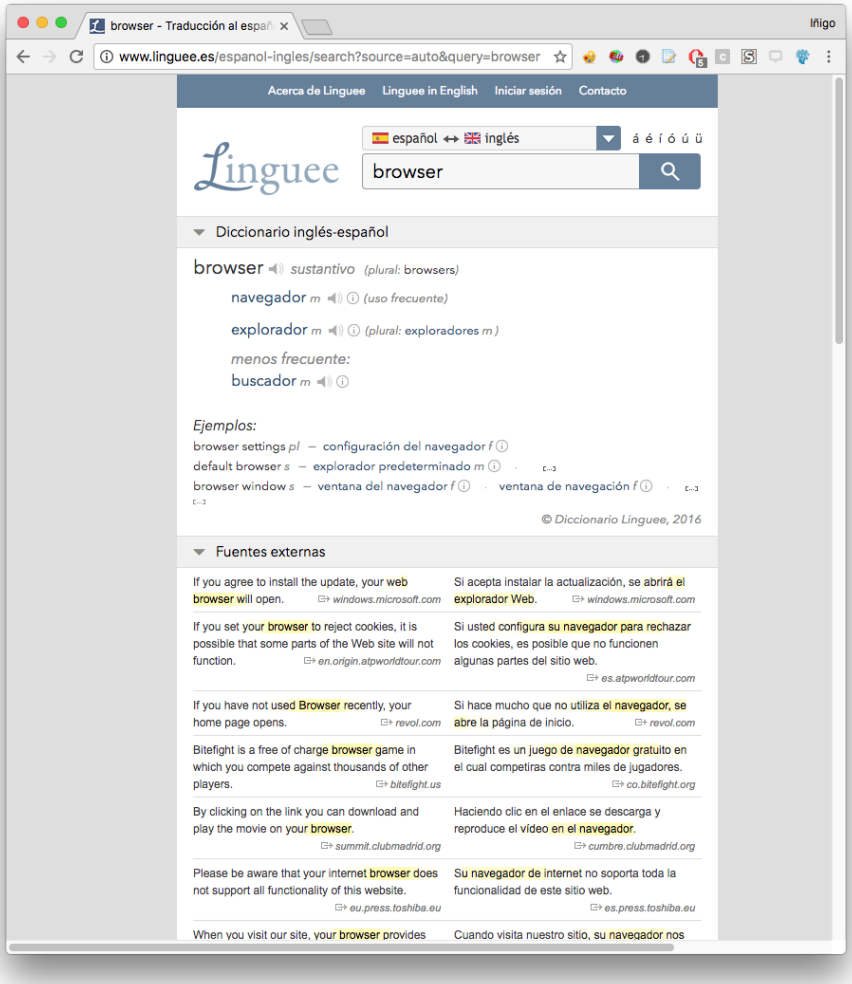

Figure B.8: Linguee

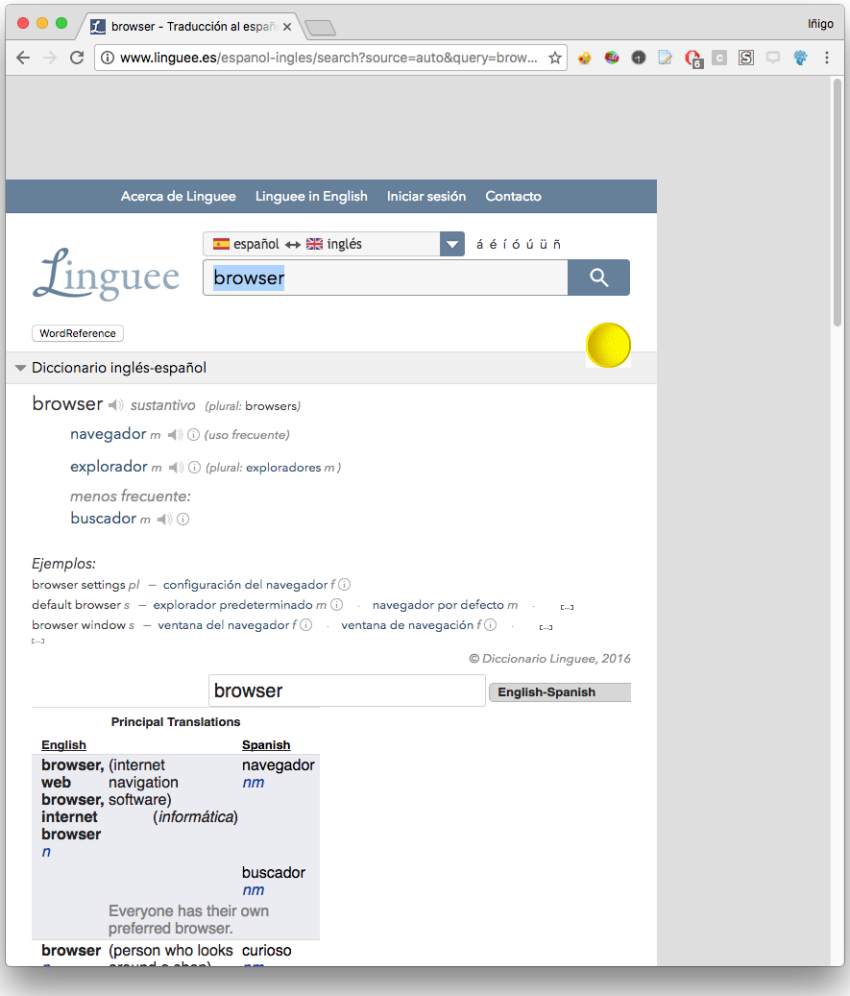

Figure B.9: Linguee after Wordreference addition

EasyChair

This example makes use of a link (pre-set widget) to join EasyChair website with the URL to justify the absence in the university when the users goes to a conference.

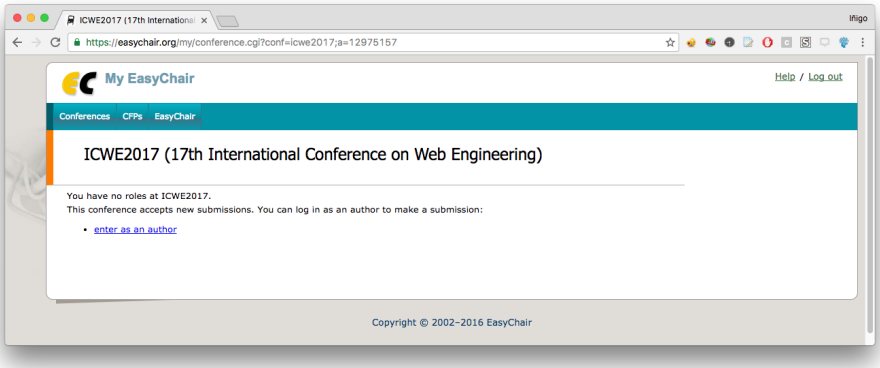

Figure B.10: EasyChair

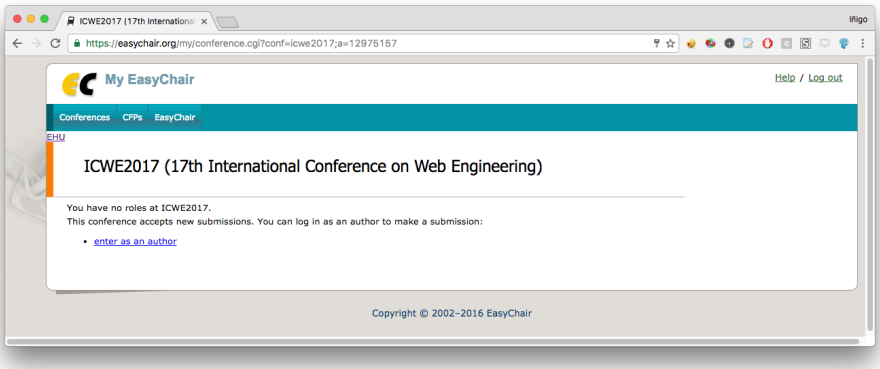

Figure B.11: EasyChair after EHU link addition

NYTimes-Weather

This example employs a simple widget to add weather information from the New York city and it must be taken into account that this widget will update its content every time the user visits the website.

Figure B.12: New York Times

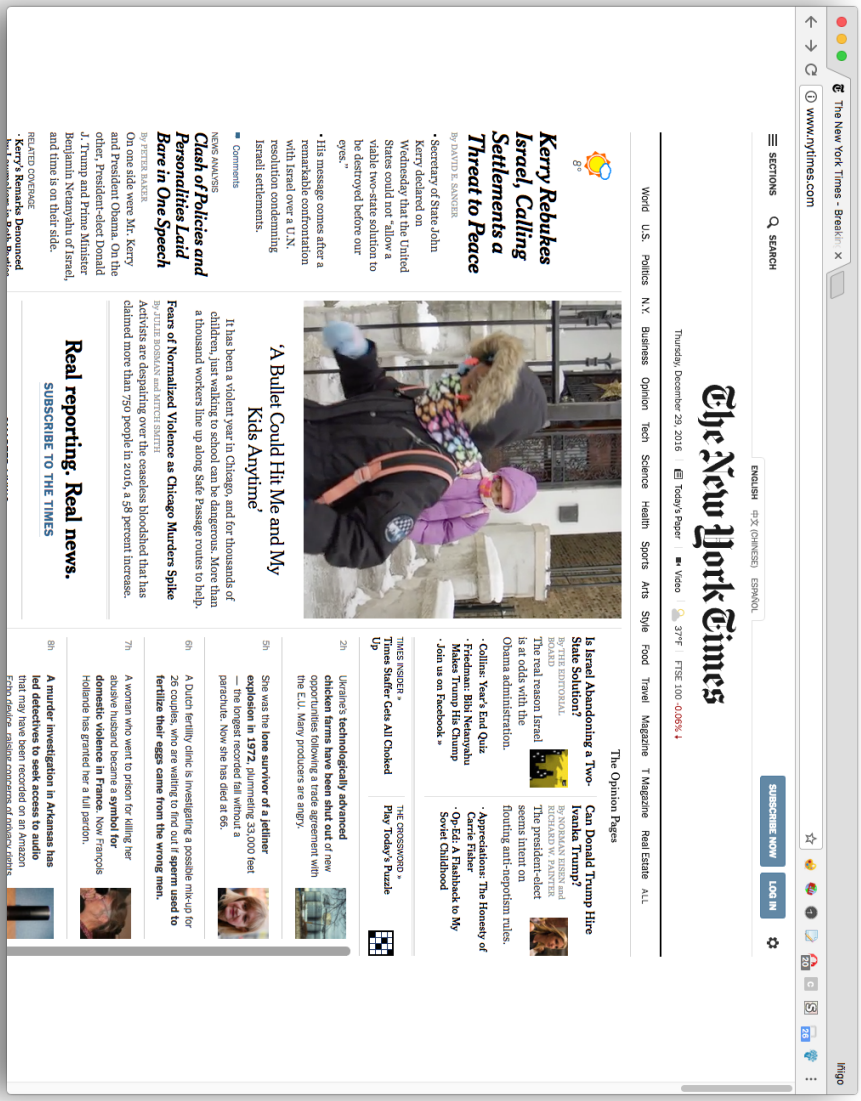

Figure B.13: New York Times after weather addition

Open Science Framework

This example moves the 'Citation' element from the top right column to the left column widgetizing this element in order to have this element is a more accessible place to the user.

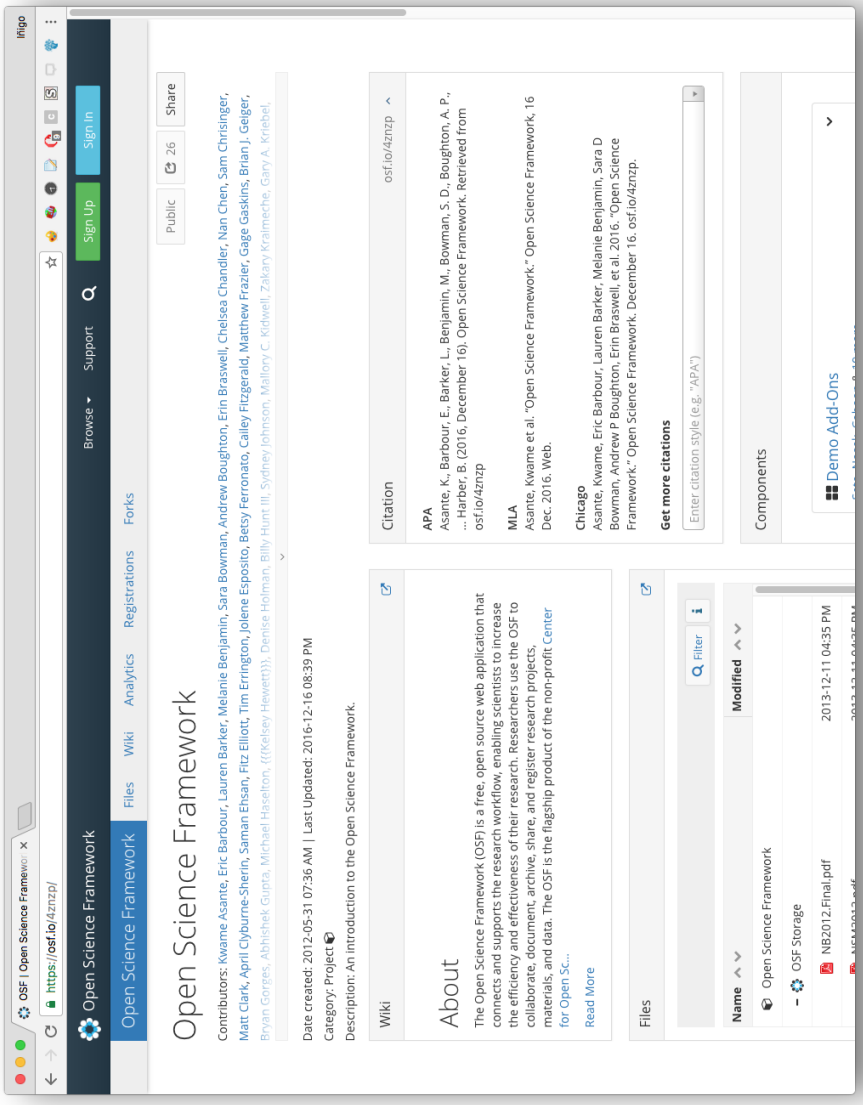

Figure B.14: Open Science Framework

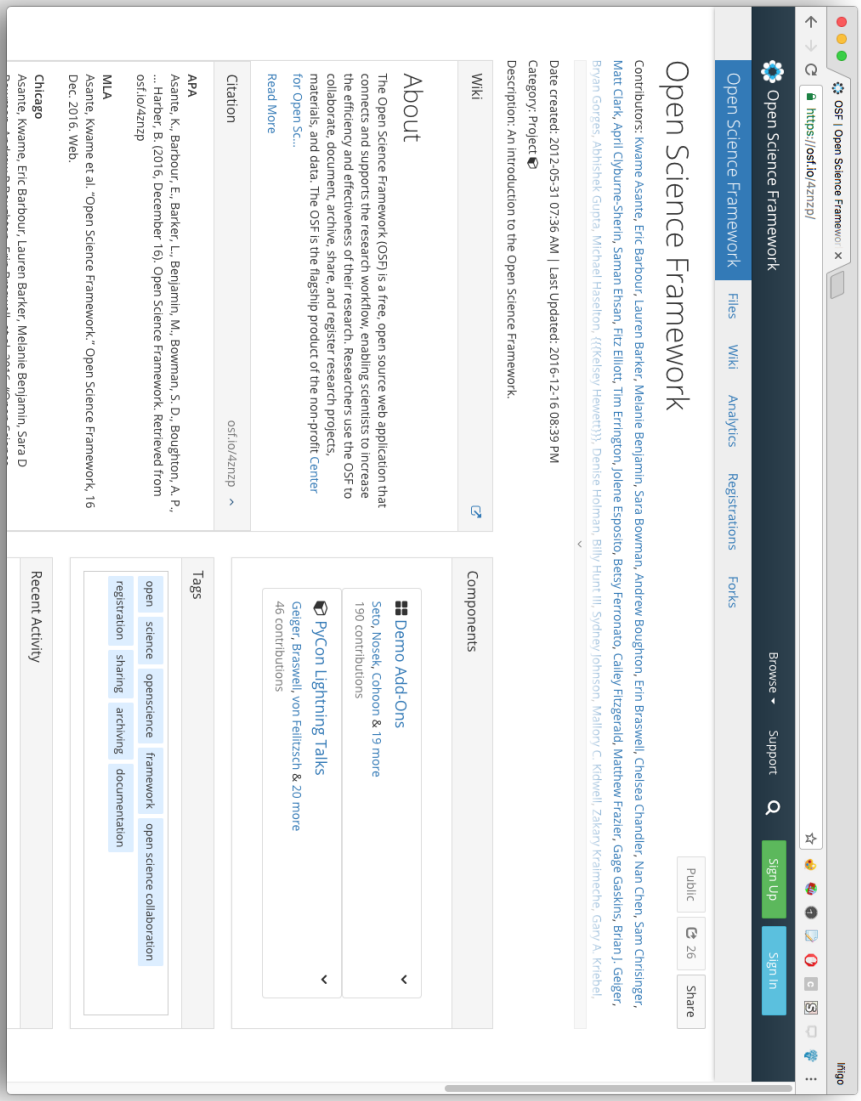

Figure B.15: Open Science Framework after element position change

Bibliography

[AAF+05] Maristella Agosti, Hanne Albrechtsen, Nicola Ferro, Ingo Frommholz, Preben Hansen, Nicola Orio, Emanuele Panizzi, Annelise Mark Pejtersen, and Ulrich Thiel. Dilas: a digital library annotation service. In *Proceedings of the International Workshop on Annotation for Collaboration - Methods, Tools and Practices, La Sorbonne, Paris, France, 2005, November 23-24*, pages 91–101, 2005.

[ABC+13] Carmelo Ardito, Paolo Bottoni, Maria Francesca Costabile, Giuseppe Desolda, Maristella Matera, Antonio Piccinno, and Matteo Picozzi. Enabling end users to create, annotate and share personal information spaces. In *End-User Development - 4th International Symposium, IS-EUD 2013, Copenhagen, Denmark, June 10-13, 2013. Proceedings*, pages 40–55, 2013.

[ACD+14] Carmelo Ardito, Maria Francesca Costabile, Giuseppe Desolda, Rosa Lanzilotti, Maristella Matera, Antonio Piccinno, and Matteo Picozzi. User-driven visual composition of service-based interactive spaces. *J. Vis. Lang. Comput.*, 25(4):278–296, 2014.

[ACD+15] Carmelo Ardito, Maria Francesca Costabile, Giuseppe Desolda, Markus Latzina, and Maristella Matera. Hands-on actionable mashups. In *End-User Development - 5th*

International Symposium, IS-EUD 2015, Madrid, Spain, May 26-29, 2015. Proceedings, pages 295–298, 2015.

[AD13] Cristóbal Arellano and Oscar Díaz. Lightweight end-user software sharing. In *End-User Development - 4th International Symposium, IS-EUD 2013, Copenhagen, Denmark, June 10-13, 2013. Proceedings*, pages 241–246, 2013.

[AGJ+16] Kumaripaba Athukorala, Dorota Glowacka, Giulio Jacucci, Antti Oulasvirta, and Jilles Vreeken. Is exploratory search different? A comparison of information search behavior for exploratory and lookup tasks. *JASIST*, 67(11):2635–2651, 2016.

[AKG10] Anne Aula, Rehan M. Khan, and Zhiwei Guan. How does search behavior change as search becomes more difficult? In *Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Atlanta, Georgia, USA, April 10-15, 2010*, pages 35–44, 2010.

[AN15] Mohammed AlSada and Tatsuo Nakajima. Parallel web browsing in tangible augmented reality environments. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems, Seoul, CHI 2015 Extended Abstracts, Republic of Korea, April 18 - 23, 2015*, pages 953–958, 2015.

[AP11] Saeed Aghaee and Cesare Pautasso. End-user programming for web mashups - open research challenges. In *Current Trends in Web Engineering - Workshops, Doctoral Symposium, and Tutorials, Held at ICWE 2011, Paphos, Cyprus, June 20-21, 2011. Revised Selected Papers*, pages 347–351, 2011.

[AP14] Saeed Aghaee and Cesare Pautasso. End-user development of mashups with naturalmash. *J. Vis. Lang. Comput.*, 25(4):414–432, 2014.

[ATD08] Eytan Adar, Jaime Teevan, and Susan T. Dumais. Large scale analysis of web revisitation patterns. In *Proceedings of the 2008 Conference on Human Factors in Computing Systems, CHI 2008, 2008, Florence, Italy, April 5-10, 2008*, pages 1197–1206, 2008.

[BCL+04] Paolo Bottoni, Roberta Civica, Stefano Levialdi, Laura Orso, Emanuele Panizzi, and Rosa Trinchese. MADCOW: a multimedia digital annotation system. In *Proceedings of the working conference on Advanced visual interfaces, AVI 2004, Gallipoli, Italy, May 25-28, 2004*, pages 55–62, 2004.

[BDRR11] Christian Brel, Anne-Marie Dery-Pinna, Philippe Renevier-Gonin, and Michel Riveill. Ontocompo: A tool to enhance application composition. In *Human-Computer Interaction - INTERACT 2011 - 13th IFIP TC 13 International Conference, Lisbon, Portugal, September 5-9, 2011, Proceedings, Part IV*, pages 588–591, 2011.

[BFG01] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 119–128, 2001.

[BFR+16] Gabriela Bosetti, Sergio Firmenich, Gustavo Rossi, Marco Winckler, and Tomas Barbieri. Web objects ambient: An integrated platform supporting new kinds of personal web experiences. In *Web Engineering - 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*, pages 563–566, 2016.

[BMM+11] Antonio Bottaro, Enrico Marino, Franco Milicchio, Alberto Paoluzzi, Maurizio Rosina, and Federico Spini. Visual programming of location-based services. In *Human Interface and the Management of Information. Interacting with Information - Symposium on Human Interface 2011, Held as Part of HCI International 2011, Orlando, FL, USA, July 9-14, 2011, Proceedings, Part I*, pages 3–12, 2011.

[BMM12] Alberto Bartoli, Eric Medvet, and Marco Mauri. Recording and replaying navigations on AJAX web sites. In *International Web Conference on Web Engineering - ICWE 2012, 12th International Conference, Berlin, Germany, July 23-27, 2012. Proceedings*, pages 370–377, 2012.

[Bou99] Niels Olof Bouvin. Unifying strategies for web augmentation. In *HYPERTEXT '99, Proceedings of the 10th ACM Conference on Hypertext and Hypermedia: Returning to Our Diverse Roots, February 21-25, 1999, Darmstadt, Germany*, pages 91–100, 1999.

[BPM15] Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. Synthesizing web element locators. In *30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015*, pages 331–341, 2015.

[Bro] Browser extension. https://en.wikipedia.org/wiki/Browser_extension. Accessed: 2017-01-15.

[BS11] Margaret M. Burnett and Scaffidi. End-user development. http://www.interaction-design.org/encyclopedia/end-user_development.html, 2011. In: Encyclopedia of Human-Computer Interaction. Soegaard, Mads and Dam, Rikke Friis (eds.).

[BWR+05] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology, Seattle, WA, USA, October 23-26, 2005*, pages 163–172, 2005.

[CDA00a] I. Cingil, A. Dogac, and A. Azgin. A Broader Approach to Personalization. *Communications of the ACM*, 43(8):136–141, 2000.

[CDA00b] Ibrahim Cingil, Asuman Dogac, and Ayca Azgin. A broader approach to personalization. *Commun. ACM*, 43(8):136–141, 2000.

[CDM+11] Cinzia Cappiello, Florian Daniel, Maristella Matera, Matteo Picozzi, and Michael Weiss. Enabling end user development through mashups: Requirements, abstractions and innovation toolkits. In *End-User Development - Third International Symposium, IS-EUD 2011, Torre Canne (BR), Italy, June 7-10, 2011. Proceedings*, pages 9–24, 2011.

[CFB+13] Cecilia Challiol, Sergio Firmenich, Gabriela Alejandra Bosetti, Silvia E. Gordillo, and Gustavo Rossi. Crowdsourcing mobile web applications. In *Current Trends in Web Engineering - ICWE 2013 International Workshops ComposableWeb, QWE, MDWE, DMSSW, EMotions, CSE, SSN, and PhD Symposium, Aalborg, Denmark, July 8-12, 2013. Revised Selected Papers*, pages 223–237, 2013.

[CHS04] Philipp Cimiano, Siegfried Handschuh, and Steffen Staab. Towards the self-annotating web. In *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 462–471, 2004.

[CM12] Kerry Shih-Ping Chang and Brad A. Myers. Webcrystal: understanding and reusing examples in web authoring. In *CHI Conference on Human Factors in Computing Systems, CHI '12, Austin, TX, USA - May 05 - 10, 2012*, pages 3205–3214, 2012.

[CMP+11] Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Gabriele Sprega, Donato Barbagallo, and Chiara Francalanci. Dashmash: A mashup environment for end user development. In *Web Engineering - 11th International Conference, ICWE 2011, Paphos, Cyprus, June 20-24, 2011*, pages 152–166, 2011.

[CMP13] Cinzia Cappiello, Maristella Matera, and Matteo Picozzi. End-user development of mobile mashups. In *Design, User Experience, and Usability. Web, Mobile, and Product Design - Second International Conference, DUXU 2013, Held as Part of HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013, Proceedings, Part IV*, pages 641–650, 2013.

[CNG+12] Olexiy Chudnovskyy, Tobias Nestler, Martin Gaedke, Florian Daniel, José Ignacio Fernández-Villamor, Vadim I. Chepegin, José Angel Fornas, Scott Wilson, Christoph Kögler, and Heng Chang. End-user-oriented telco mashups: the OMELETTE approach. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pages 235–238, 2012.

[CP08] Gerardo Canfora and Massimiliano Di Penta. Service-oriented architectures testing: A survey. In *Software Engineering, International Summer Schools, ISSSE 2006-2008, Salerno, Italy, Revised Tutorial Lectures*, pages 78–105, 2008.

[CPT11] Prach Chaisatien, Korawit Prutsachainimmit, and Takehiro Tokuda. Mobile mashup generator system for cooperative applications of different mobile devices. In *Web Engineering - 11th International Conference, ICWE 2011, Paphos, Cyprus, June 20-24, 2011*, pages 182–197, 2011.

[CRDC12] Soudip Roy Chowdhury, Carlos Rodríguez, Florian Daniel, and Fabio Casati. Baya: assisted mashup development as a service. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pages 409–412, 2012.

[CVM14] Lorena Castaneda, Norha M. Villegas, and Hausi A. Müller. Self-adaptive applications: on the development of personalized web-tasking systems. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings, Hyderabad, India, June 2-3, 2014*, pages 49–54, 2014.

[DA15] Oscar Díaz and Cristóbal Arellano. The augmented web: Rationales, opportunities, and challenges on browser-side transcoding. *TWEB*, 9(2):8, 2015.

[DAA13] Oscar Díaz, Cristóbal Arellano, and Maider Azanza. A language for end-user web augmentation: Caring for producers and consumers alike. *TWEB*, 7(2):9, 2013.

[DAA+14] Oscar Díaz, Cristóbal Arellano, Iñigo Aldalur, Haritz Medina, and Sergio Firmenich. End-user browser-side modification of web pages. In *Web Information Systems Engineering - WISE 2014 - 15th International Conference, Thessaloniki, Greece, October 12-14, 2014, Proceedings, Part I*, pages 293–307, 2014.

[DANP13] Tao Dong, Mark S. Ackerman, Mark W. Newman, and Gaurav Paruthi. Social overlays: Collectively making websites more usable. In *Human-Computer Interaction - INTERACT 2013 - 14th IFIP TC 13 International Conference, Cape Town, South Africa, September 2-6, 2013, Proceedings, Part IV*, pages 280–297, 2013.

[DB10] Patrick Dubroy and Ravin Balakrishnan. A study of tabbed browsing among mozilla firefox users. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems, CHI 2010, Atlanta, Georgia, USA, April 10-15, 2010*, pages 673–682, 2010.

[DBS09] Nilesh N. Dalvi, Philip Bohannon, and Fei Sha. Robust web extraction: an approach based on a probabilistic tree-edit model. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 335–348, 2009.

[DF13] Florian Daniel and Andrea Furlan. The interactive API (iapi). In *Current Trends in Web Engineering - ICWE 2013 International Workshops ComposableWeb, QWE, MDWE, DMSSW, EMotions, CSE, SSN, and PhD Symposium, Aalborg, Denmark, July 8-12, 2013. Revised Selected Papers*, pages 3–15, 2013.

[dom] Github - don-to-image. `https://github.com/tsayen/dom-to-image)`. Accessed: 2016-11-09.

[DPP07] Oscar Díaz, Sandy Pérez, and Iñaki Paz. Providing personalized mashups within the context of existing web applications. In *Web Information Systems Engineering - WISE 2007, 8th International Conference on Web*

Information Systems Engineering, Nancy, France, December 3-7, 2007, Proceedings, pages 493–502, 2007.

[DRC+12] Florian Daniel, Carlos Rodríguez, Soudip Roy Chowdhury, Hamid R. Motahari Nezhad, and Fabio Casati. Discovery and reuse of composition knowledge for assisted mashup development. In *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pages 493–494, 2012.

[Dro03] R. Geoff Dromey. Software quality-prevention versus cure? *Software Quality Journal*, 11(3):197–210, 2003.

[DSAT12] Oscar Díaz, Josune De Sosa, Cristóbal Arellano, and Salvador Trujillo. Web-based tool integration: A web augmentation approach. In *International Web Conference on Web Engineering - ICWE 2012, 12th International Conference, Berlin, Germany, July 23-27, 2012. Proceedings*, pages 431–434, 2012.

[EBG+07] Robert Ennals, Eric A. Brewer, Minos N. Garofalakis, Michael Shadle, and Prashant Gandhi. Intel mash maker: join the web. *SIGMOD Record*, 36(4):27–33, 2007.

[ESZ16] Philipp Eichmann, Hyunchang Song, and Emanuel Zgraggen. cted: Advancing selection mechanisms in web browsers. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, May 7-12, 2016, Extended Abstracts*, pages 3049–3055, 2016.

[FB11a] Emilio Ferrara and Robert Baumgartner. Design of automatically adaptable web wrappers. *CoRR*, abs/1103.1254, 2011.

[FB11b] Emilio Ferrara and Robert Baumgartner. Intelligent self-repairable web wrappers. In *AI*IA 2011: Artificial Intelligence Around Man and Beyond - XIIth International Conference of the Italian Association for Artificial Intelligence, Palermo, Italy, September 15-17, 2011. Proceedings*, pages 274–285, 2011.

[FB11c] Emilio Ferrara and Robert Baumgartner. Intelligent self-repairable web wrappers. *CoRR*, abs/1106.3967, 2011.

[FBR+16] Sergio Firmenich, Gabriela Alejandra Bosetti, Gustavo Rossi, Marco Winckler, and Tomas Barbieri. Abstracting and structuring web contents for supporting personal web experiences. In *Web Engineering - 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*, pages 77–95, 2016.

[FFR+15] Diego Firmenich, Sergio Firmenich, Gustavo Rossi, Marco Winckler, and Damiano Distante. User interface adaptation using web augmentation techniques: Towards a negotiated approach. In *Engineering the Web in the Big Data Era - 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings*, pages 147–164, 2015.

[FFRA14] Diego Firmenich, Sergio Firmenich, José Matías Rivero, and Leandro Antonelli. A platform for web augmentation requirements specification. In *Web Engineering, 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings*, pages 1–20, 2014.

[Fil06] Robert E. Filman. From the editor in chief: Taking back the web. *IEEE Internet Computing*, 10(1):3–5, 2006.

[Fir] Firepath :: Add-ons for firefox. https://addons.mozilla.org/en-us/firefox/addon/firepath/. Accessed: 2016-12-12.

[FMFB14] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. Web data extraction, applications and techniques: A survey. *Knowl.-Based Syst.*, 70:301–323, 2014.

[Fow10] M. Fowler. *Domain-Specific Languages*. Addison-Wesley Professional, 2010.

[FURS16] Darian Frajberg, Matias Urbieta, Gustavo Rossi, and Wieland Schwinger. Volatile functionality in action: Methods, techniques and assessment. In *Web Engineering - 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*, pages 59–76, 2016.

[FWRG11a] Sergio Firmenich, Marco Winckler, Gustavo Rossi, and Silvia E. Gordillo. A crowdsourced approach for concern-sensitive integration of information across the web. *J. Web Eng.*, 10(4):289–315, 2011.

[FWRG11b] Sergio Firmenich, Marco Winckler, Gustavo Rossi, and Silvia E. Gordillo. A framework for concern-sensitive, client-side adaptation. In *International Web Conference on Web Engineering - ICWE 2011, 11th International Conference, Paphos, Cyprus, June 20-24, 2011*, pages 198–213, 2011.

[GHT10] Junxia Guo, Hao Han, and Takehiro Tokuda. Towards flexible mashup of web applications based on information extraction and transfer. In *Web Information Systems Engineering - WISE 2010 - 11th International Conference,*

Hong Kong, China, December 12-14, 2010. Proceedings, pages 602–615, 2010.

[GMPP14] Giuseppe Ghiani, Marco Manca, Fabio Paternò, and Claudio Porta. Beyond responsive design: Context-dependent multimodal augmentation of web applications. In *Mobile Web Information Systems - 11th International Conference, MobiWIS 2014, Barcelona, Spain, August 27-29, 2014. Proceedings*, pages 71–85, 2014.

[GPS11] Giuseppe Ghiani, Fabio Paternò, and Lucio Davide Spano. Creating mashups by direct manipulation of existing web applications. In *End-User Development - Third International Symposium, IS-EUD 2011, Torre Canne (BR), Italy, June 7-10, 2011. Proceedings*, pages 42–52, 2011.

[GPSP16] Giuseppe Ghiani, Fabio Paternò, Lucio Davide Spano, and Giuliano Pintori. An environment for end-user development of web mashups. *Int. J. Hum.-Comput. Stud.*, 87:38–64, 2016.

[GSCJ14] George Gkotsis, Karen Stepanyan, Alexandra I. Cristea, and Mike Joy. Entropy-based automated wrapper generation for weblog data extraction. *World Wide Web*, 17(4):827–846, 2014.

[GTZ+11] Steven Gardiner, Anthony Tomasic, John Zimmerman, Rafae Aziz, and Kathryn Rivard. Mixer: Mixed-initiative data retrieval and integration by example. In *Human-Computer Interaction - INTERACT 2011 - 13th IFIP TC 13 International Conference, Lisbon, Portugal, September 5-9, 2011, Proceedings, Part I*, pages 426–443, 2011.

[GYK11] Paul A. Gross, Jennifer Yang, and Caitlin Kelleher. Dinah: an interface to assist non-programmers with selecting

program code causing graphical output. In *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, pages 3397–3400, 2011.

[Hic52] W. E. Hick. On the rate of gain of information. *Quarterly Journal of Experimental Psychology*, 4(1):11–26, 1952.

[HNPN14] Maria Husmann, Michael Nebeling, Stefano Pongelli, and Moira C. Norrie. Multimasher: Providing architectural support and visual tools for multi-device mashups. In *Web Information Systems Engineering - WISE 2014 - 15th International Conference, Thessaloniki, Greece, October 12-14, 2014, Proceedings, Part II*, pages 199–214, 2014.

[HPB10] Dat Dac Hoang, Hye-young Paik, and Boualem Benatallah. An analysis of spreadsheet-based services mashup. In *Database Technologies 2010, Twenty-First Australasian Database Conference (ADC 2010), Brisbane, Australia, 18-22 January, 2010, Proceedings*, pages 141–150, 2010.

[HPD11] Dat Dac Hoang, Hye-Young Paik, and Wei Dong. Mashsheet: Mashups in your spreadsheet. In *Web Information System Engineering - WISE 2011 - 12th International Conference, Sydney, Australia, October 13-14, 2011. Proceedings*, pages 332–333, 2011.

[HT08] Hao Han and Takehiro Tokuda. A method for integration of web applications based on information extraction. In *Proceedings of the Eighth International Conference on Web Engineering, ICWE 2008, 14-18 July 2008, Yorktown Heights, New York, USA*, pages 189–195, 2008.

[HT10] Hao Han and Takehiro Tokuda. Towards flexible and lightweight integration of web applications by end-user programming. *IJWIS*, 6(4):359–373, 2010.

[HW10] Jeff Huang and Ryen W. White. Parallel browsing behavior on the web. In *HT'10, Proceedings of the 21st ACM Conference on Hypertext and Hypermedia, Toronto, Ontario, Canada, June 13-16, 2010*, pages 13–18, 2010.

[Hypa] Github - hypothesis. https://github.com/hypothesis/browser-extension). Accessed: 2016-11-09.

[Hypb] Hypothesis. https://hypothes.is/). Accessed: 2017-01-17.

[IAD14] Jon Iturrioz, Iker Azpeitia, and Oscar Díaz. Generalizing the "like" button: empowering websites with monitoring capabilities. In *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, pages 743–750, 2014.

[ISK+12] Muhammad Imran, Stefano Soi, Felix Kling, Florian Daniel, Fabio Casati, and Maurizio Marchese. On the systematic development of domain-specific mashup tools for end users. In *Web Engineering - 12th International Conference, ICWE 2012, Berlin, Germany, July 23-27, 2012. Proceedings*, pages 291–298, 2012.

[ISO] Iso/iec25010:2011. http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=35733. Accessed: 2016-11-14.

[JP14] Paul Johannesson and Erik Perjons. *An Introduction to Design Science*. Springer, 2014.

[jsD] Github - jsdom. https://github.com/tmpvar/jsdom). Accessed: 2016-11-09.

[KAB+11] Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan F. Blackwell, Margaret M. Burnett, Martin Erwig, Christopher Scaffidi, Joseph Lawrance, Henry Lieberman, Brad A. Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. The state of the art in end-user software engineering. *ACM Comput. Surv.*, 43(3):21:1–21:44, 2011.

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University, 1990.

[KKA06] Marek Kowalkiewicz, Tomasz Kaczmarek, and Witold Abramowicz. myportal: Robust extraction and aggregation of web content. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 1219–1222, 2006.

[KLN05] Gabor Karsai, Andras Lang, and Sandeep Neema. Design patterns for open tool integration. *Software and System Modeling*, 4(2):157–170, 2005.

[KMK+10] Max Van Kleek, Brennan Moore, David R. Karger, Paul André, and m. c. schraefel. Atomate it! end-user context-sensitive automation using heterogeneous information sources on the web. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 951–960, 2010.

[KRNK13] Dejan Kovachev, Dominik Renzel, Petru Nicolaescu, and Ralf Klamma. Direwolf - distributing and migrating user interfaces for widget-based web applications. In *Web Engineering - 13th International Conference, ICWE 2013, Aalborg, Denmark, July 8-12, 2013. Proceedings*, pages 99–113, 2013.

[KS08] Bill Kules and Ben Shneiderman. Users can change their web search tactics: Design guidelines for categorized overviews. *Inf. Process. Manage.*, 44(2):463–484, 2008.

[KWG14] Michael Krug, Fabian Wiedemann, and Martin Gaedke. Smartcomposition: A component-based approach for creating multi-screen mashups. In *Web Engineering, 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings*, pages 236–253, 2014.

[Lad10] Mohamad I. Ladan. Web services testing approaches: A survey and a classification. In *Networked Digital Technologies - Second International Conference, NDT 2010, Prague, Czech Republic, July 7-9, 2010. Proceedings, PartII*, pages 70–79, 2010.

[LCRS13] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Cristiano Spadaro. Comparing the maintainability of selenium webdriver test suites employing different locators: A case study. In *Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to Testing Automation*, pages 53–58, 2013.

[LCRT13] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Paolo Tonella. Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. In *20th Working Conference on Reverse Engineering, WCRE 2013,*

Koblenz, Germany, October 14-17, 2013, pages 272–281, 2013.

[LCRT14] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Paolo Tonella. Visual vs. dom-based web locators: An empirical study. In *Web Engineering, 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings*, pages 322–340, 2014.

[LHML08] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa A. Lau. Coscripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the 2008 Conference on Human Factors in Computing Systems, CHI 2008, 2008, Florence, Italy, April 5-10, 2008*, pages 1719–1728, 2008.

[LPW06] Henry Lieberman, Fabio Paternò, and Volker Wulf, editors. *End User Development*. Human-Computer Interaction Series. Springer, 2006.

[LSRT14] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. Reducing web test cases aging by means of robust xpath locators. In *25th IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Naples, Italy, November 3-6, 2014*, pages 449–454, 2014.

[LSRT15] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. Automated generation of visual web tests from dom-based web tests. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 775–782, 2015.

[LSRT16] Maurizio Leotta, Andrea Stocco, Filippo Ricca, and Paolo Tonella. Robula+: an algorithm for generating robust xpath

locators for web testing. *Journal of Software: Evolution and Process*, 28(3):177–204, 2016.

[MBH10] Mark Harman Mustafa Bozkurt and Youssef Hassoun. Testing web services: A survey. Technical report, Department of Computer Science, King's College London, 2010.

[MGF16] Jesús López Miján, Irene Garrigós, and Sergio Firmenich. Supporting personalization in legacy web sites through client-side adaptation. In *Web Engineering - 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*, pages 588–592, 2016.

[MHBJ14] Ali Moosavi, Salman Hooshmand, Sara Baghbanzadeh, and Guy-Vincent Jourdan. Indexing rich internet applications using components-based crawling. In *Web Engineering, 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings*, pages 200–217, 2014.

[MHS05] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, 2005.

[MPR+09] Paula Montoto, Alberto Pan, Juan Raposo, Fernando Bellas, and Javier López. Automating navigation sequences in AJAX websites. In *International Web Conference on Web Engineering - ICWE 2009, 9th International Conference, San Sebastián, Spain, June 24-26, 2009, Proceedings*, pages 166–180, 2009.

[MS15] Daniele Massa and Lucio Davide Spano. Facemashup: Enabling end user development on social networks data. In *End-User Development - 5th International Symposium, IS-*

EUD 2015, Madrid, Spain, May 26-29, 2015. Proceedings, pages 204–210, 2015.

[MSCC13] Josip Maras, Maja Stula, Jan Carlson, and Ivica Crnkovic. Identifying code of individual features in client-side web applications. *IEEE Trans. Software Eng.*, 39(12):1680–1697, 2013.

[NK15] Petru Nicolaescu and Ralf Klamma. A methodology and tool support for widget-based web application development. In *Engineering the Web in the Big Data Era - 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings*, pages 515–532, 2015.

[NLHL03] Mark W. Newman, James Lin, Jason I. Hong, and James A. Landay. DENIM: an informal web site design tool inspired by observations of practice. *Human-Computer Interaction*, 18(3):259–324, 2003.

[NLN12] Michael Nebeling, Stefania Leone, and Moira C. Norrie. Crowdsourced web engineering and design. In *Web Engineering - 12th International Conference, ICWE 2012, Berlin, Germany, July 23-27, 2012. Proceedings*, pages 31–45, 2012.

[PGSP15] Raúl Peña-Ortiz, José Antonio Gil, Julio Sahuquillo, and Ana Pont. Surfing the web using browser interface facilities: A performance evaluation approach. *J. Web Eng.*, 14(1&2):3–21, 2015.

[Pil05] Mark Pilgrim. *Greasemonkey hacks - tips and tools for remixing the web with Firefox*. O'Reilly, 2005.

[Pol10] Eleanor Poley. RUMU editor: a non-wysiwyg web editor for non-technical users. In *Proceedings of the 28th International*

Conference on Human Factors in Computing Systems, CHI 2010, Extended Abstracts Volume, Atlanta, Georgia, USA, April 10-15, 2010, pages 4357–4362, 2010.

[PPH+09] Danh Le Phuoc, Axel Polleres, Manfred Hauswirth, Giovanni Tummarello, and Christian Morbidoni. Rapid prototyping of semantic mash-ups through semantic web pipes. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 581–590, 2009.

[PSJ+13] Thomas H. Park, Ankur Saxena, Swathi Jagannath, Susan Wiedenbeck, and Andrea Forte. Openhtml: designing a transitional web editor for novices. In *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013, Extended Abstracts*, pages 1863–1868, 2013.

[RAR+11] Alexander Repenning, Navid Ahmadi, Nadia Repenning, Andri Ioannidou, David C. Webb, and Krista Sekeres Marshall. Collective programming: Making end-user programming (more) social. In *End-User Development - Third International Symposium, IS-EUD 2011, Torre Canne (BR), Italy, June 7-10, 2011. Proceedings*, pages 325–330, 2011.

[RBM13] Carsten Radeck, Gregor Blichmann, and Klaus Meißner. Capview - functionality-aware visual mashup development for non-programmers. In *Web Engineering - 13th International Conference, ICWE 2013, Aalborg, Denmark, July 8-12, 2013. Proceedings*, pages 140–155, 2013.

[RDR11] Valentim Realinho, A. Eduardo Dias, and Teresa Romão. Testing the usability of a platform for rapid development

of mobile context-aware applications. In *Human-Computer Interaction - INTERACT 2011 - 13th IFIP TC 13 International Conference, Lisbon, Portugal, September 5-9, 2011, Proceedings, Part III*, pages 521–536, 2011.

[Res] Github - resemble. https://github.com/Huddle/Resemble.js). Accessed: 2016-11-09.

[Rho00] Bradley J. Rhodes. Margin notes: building a contextually aware associative memory. In *Proceedings of the 5th International Conference on Intelligent User Interfaces, IUI 2000, New Orleans, LA, USA, January 9-12, 2000*, pages 219–224, 2000.

[RLS+13] Filippo Ricca, Maurizio Leotta, Andrea Stocco, Diego Clerissi, and Paolo Tonella. Web testware evolution. In *15th IEEE International Symposium on Web Systems Evolution, WSE 2013, Eindhoven, The Netherlands, September 27, 2013*, pages 39–44, 2013.

[RMS13] Juwel Rana, Sarwar Morshed, and Kåre Synnes. End-user creation of social apps by utilizing web-based social components and visual app composition. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013, Companion Volume*, pages 1205–1214, 2013.

[Rob] Robustness. https://en.wikipedia.org/wiki/Robustness_(computer_science). Accessed: 2017-01-15.

[RPR17] Sriram Raghavan, Udaya Parampalli, and S. V. Raghavan. Re-engineering simultaneous internet sessions process-separated browsers. In *Proceedings of the Australasian Computer Science Week Multiconference, ACSW 2017,*

Geelong, Australia, January 31 - February 3, 2017, pages 70:1–70:10, 2017.

[RS04] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *Semantic Web Services and Web Process Composition, First International Workshop, SWSWPC 2004, San Diego, CA, USA, July 6, 2004, Revised Selected Papers*, pages 43–54, 2004.

[RSG01] Gustavo Rossi, Daniel Schwabe, and Robson Guimarães. Designing personalized web applications. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 275–284, 2001.

[Scra] Blasting the myth of the fold. http://boxesandarrows.com/blasting-the-myth-of-the-fold/. Accessed: 2017-06-13.

[Scrb] People do not scroll. http://uxmyths.com/post/654047943/myth-people-dont-scroll. Accessed: 2017-06-13.

[Scrc] What you think you know about the web is wrong. http://time.com/12933/what-you-think-you-know-about-the-web-is-wrong/. Accessed: 2017-06-13.

[SDW08] Michael Spahn, Christian Dörner, and Volker Wulf. End user development: Approaches towards a flexible software design. In *16th European Conference on Information Systems, ECIS 2008, Galway, Ireland, 2008*, pages 303–314, 2008.

[SER09] Kathryn T. Stolee, Sebastian G. Elbaum, and Gregg Rothermel. Revealing the copy and paste habits of end users. In *IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2009, Corvallis, OR, USA, 20-24 September 2009, Proceedings*, pages 59–66, 2009.

[SLRT14] Andrea Stocco, Maurizio Leotta, Filippo Ricca, and Paolo Tonella. PESTO: A tool for migrating dom-based to visual web tests. In *14th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2014, Victoria, BC, Canada, September 28-29, 2014*, pages 65–70, 2014.

[SPS] Spss statistics. `http://www.spss.com.hk/statistics/`. Accessed: 2016-11-14.

[SPWL14] Aju Thalappillil Scaria, Rose Marie Philip, Robert West, and Jure Leskovec. The last click: why users give up information network navigation. In *Seventh ACM International Conference on Web Search and Data Mining, WSDM 2014, New York, NY, USA, February 24-28, 2014*, pages 213–222, 2014.

[TDD+09] Michael Toomim, Steven M. Drucker, Mira Dontcheva, Ali Rahimi, Blake Thomson, and James A. Landay. Attaching UI enhancements to websites with end users. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems, CHI 2009, Boston, MA, USA, April 4-9, 2009*, pages 1859–1868, 2009.

[TS14] Ahmed A. O. Tayeh and Beat Signer. Open cross-document linking and browsing based on a visual plug-in architecture. In *Web Information Systems Engineering - WISE 2014 - 15th International Conference, Thessaloniki, Greece, October 12-14, 2014, Proceedings, Part II*, pages 231–245, 2014.

[TS15] Ahmed A. O. Tayeh and Beat Signer. A dynamically extensible open cross-document link service. In *Web Information Systems Engineering - WISE 2015 - 16th International Conference, Miami, FL, USA, November 1-3, 2015, Proceedings, Part I*, pages 61–76, 2015.

[UCI+06] Victoria S. Uren, Philipp Cimiano, José Iria, Siegfried Handschuh, Maria Vargas-Vera, Enrico Motta, and Fabio Ciravegna. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *J. Web Sem.*, 4(1):14–28, 2006.

[W3C] W3C. Requirement For Standardizing Widgets. http://dev.w3.org/2006/waf/widgets-reqs/. Accessed: 2017-04-04.

[WCB+15] Marco Winckler, Ricardo Andrade Cava, Eric Barboni, Philippe A. Palanque, and Carla M. D. S. Freitas. Usability aspects of the inside-in approach for ancillary search tasks on the web. In *Human-Computer Interaction - INTERACT 2015 - 15th IFIP TC 13 International Conference, Bamberg, Germany, September 14-18, 2015, Proceedings, Part II*, pages 211–230, 2015.

[Web] Webmakeup - chrome web store. https://chrome.google.com/webstore/detail/webmakeup/alnhegodephpjnaghlcemlnpdknhbhjj). Accessed: 2017-01-11.

[Web10] Matthew J. Webber. A stateful web augmentation toolkit. Technical report, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science, 2010.

[WH07] Jeffrey Wong and Jason I. Hong. Making mashups with marmite: towards end-user programming for the web. In *Proceedings of the 2007 Conference on Human Factors in Computing Systems, CHI 2007, San Jose, California, USA, April 28 - May 3, 2007*, pages 1435–1444, 2007.

[Wik] Wikipedia. Modding. Accessed: 2017-02-21.

[WNM11] Usman Wajid, Abdallah Namoun, and Nikolay Mehandjiev. Alternative representations for end user composition of service-based systems. In *End-User Development - Third International Symposium, IS-EUD 2011, Torre Canne (BR), Italy, June 7-10, 2011. Proceedings*, pages 53–66, 2011.

[WPL15] Robert West, Ashwin Paranjape, and Jure Leskovec. Mining missing hyperlinks from human navigation traces: A case study of wikipedia. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 1242–1252, 2015.

[WW15] Sixuan Wang and Gabriel A. Wainer. A mashup architecture with modeling and simulation as a service. In *Web Information Systems Engineering - WISE 2015 - 16th International Conference, Miami, FL, USA, November 1-3, 2015, Proceedings, Part I*, pages 247–261, 2015.

[WYH09] Guiling Wang, Shaohua Yang, and Yanbo Han. Mashroom: end-user mashup programming using nested tables. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 861–870, 2009.

[Xpaa] Xpath checker :: Add-ons for firefox. `https://addons.mozilla.org/EN-us/firefox/addon/xpath-checker/`. Accessed: 2016-12-12.

[Xpab] Xpath helper - chrome web store. https://chrome.google.com/webstore/detail/xpath-helper/hgimnogjllphhhkhlmebbmlgjoejdpjl. Accessed: 2016-12-12.

[YBCD08] Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. Understanding mashup development. *IEEE Internet Computing*, 12(5):44–52, 2008.

[YCM09] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. Sikuli: using GUI screenshots for search and automation. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology, Victoria, BC, Canada, October 4-7, 2009*, pages 183–192, 2009.

[ZCW+16] Zhongyi Zhai, Bo Cheng, Zhaoning Wang, Xuan Liu, Meng Liu, and Junliang Chen. Design and implementation: the end user development ecosystem for cross-platform mobile applications. In *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11-15, 2016, Companion Volume*, pages 143–144, 2016.

Acknowledgements

Five years and eight months is a long time; there were many ups and downs along the way, and I was only able to cross the finish line thanks to the assistance and support of many people. I would like to extend my appreciation especially to the following.

First of all, I wish to express my most sincere gratitude to my supervisors, Prof. Dr. Oscar Díaz. My sincere appreciation is because I have learned from him and from his continuous help and support in all stages of this dissertation. I would also like to thank him for being an open person to ideas, and for encouraging and helping me. From the personal viewpoint, he treated me superb.

Thanks to the *Onekin Research Group* I could see a group where research is conducted. Thanks to all the colleagues that I have known during these years: Itziar Otaduy, Maider Azanza, Arantza Irastorza, Leticia Montalvillo, Jeremias Perez, Haritz Medina, Juanan Pereira, Iker Azpeitia, Cristóbal Arellano, Jokin García, Gorka Puente, Josune de Sosa, Jon Iturrioz and Felipe Ibañez for the innumerable assistance during the development of this dissertation.

This thesis was economically supported by the Spanish Ministry of Education and Science under the FPI Program. It has allowed me to be independent from the economical point of view and has supported the research stage at Toulouse (France).

To my colleagues from the ICS group at Paul Sabatier Toulouse III University Philippe Palanque, the head of the ICS team, Célia Martinie, David Navarre, Regina Bernhaupt, Eric Barboni, Camille Fayollas, Martin

Cronel, Racim Fahssi, Thiago Silva, Jean-Luc Hak, Guillaume Pottier, Arnaud Hamon, Raphaël Guénon and François Manciet that made the research stage easier. Finally I would like to express my sincere gratitude to Marco Winckler because he welcomed and integrated me in his research group.

I am deeply and forever indebted to my parents that let me choose my way and put their resources that were essentials to reach to this point and to my sister for her love, support and encouragement throughout my entire life. I thank all of my family due to their support above all these years. To my friends that unconditionally have been there and have understood my occasional absences and to my basketball teams (DKS and "Rojillas") because they put up with my stress and they helped me to disconnect from my work.

Last, but not least, I would like to express my gratitude to Oihana, her parents and sister. I would like to thank Oihana for his understanding and her love. We have gone hand in hand and she has helped me to grow not only at the professional dimension but the personal one. She was always there cheering me up and she stood by me through the good and bad times.

Glossary

- **Browser extension** is a plug-in that extends the functionality of a web browser.
- **Cascading Style Sheets (CSS)** is a style sheet language used for describing the presentation of a document written in a markup language.
- **Design Science Research** is the scientific study and creation of artefacts as they are developed and used by people with the goal of solving practical problems of general interest [JP14].
- **Do it yourself (DIY)** is the method of building, modifying, or repairing things without the direct aid of experts or professionals.
- **Document Object Model (DOM)** is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document.
- **Domain-Specific Language (DSL)** is a computer language specialized to a particular application domain.
- **End-User Development** can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact [LPW06].

- **Hypertext Markup Language (HTML)** is the standard markup language for creating web pages and web applications.
- **Javascript (JS)** is a high-level, dynamic, weakly typed, object-based, multi-paradigm, and interpreted programming language.
- **Locator** is a mechanism for uniquely identifying an element on the Web Content i.e. in the Document Object Model (DOM) [RLS+13].
- **Mashup** is a web page, or web application, that uses content from more than one source to create a single new service displayed in a single graphical interface.
- **Modding** is a slang expression that is derived from the verb "modify". Modding refers to the act of modifying hardware, software, or virtually anything else, to perform a function not originally conceived or intended by the designer [Wik].
- **Rich Internet Application (RIA)** is a Web application that has many of the characteristics of desktop application software, typically delivered by way of a site-specific browser, a browser plug-in, an independent sandbox, extensive use of JavaScript, or a virtual machine.
- **Robustness** is the ability of a computer system to cope with errors during execution.
- **Uniform Resource Locator (URL)** is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.
- **Web Augmentation** is to the web what *Augmented Reality* is to the physical world: layering relevant content/layout/navigation over the existing web to customize the user experience *[DAA13]*.
- **Web Personalization** refers to making a web site more responsive to the unique and individual needs of each user [CDA00a].

- **Widget** is an element of a graphical user interface (GUI) that displays information or provides a specific way for a user to interact with the operating system or an application.