Wikipedia Customization through Web Augmentation Techniques

Oscar Díaz
oscar.diaz@ehu.es

Cristóbal Arellano
cristobal.arellano@ehu.es

Gorka Puente
gorka.puente@ehu.es

ONEKIN Research Group
University of the Basque Country (UPV/EHU)
San Sebastián, Spain
www.onekin.org

ABSTRACT

Wikipedia is a successful example of *collaborative* knowledge construction. This can be synergistically complemented with *personal* knowledge construction whereby individuals are supported in their sharing, experimenting and building of information in a more private setting, without the scrutiny of the whole community. Ideally, both approaches should be seamlessly integrated so that wikipedians can easily transit from the public sphere to the private sphere, and vice versa. To this end, we introduce *WikiLayer*, a plugin for *Wikipedia* that permits wikipedians locally supplement *Wikipedia* articles with their own content (i.e. a *layer*). Layering additional content is achieved locally by seamlessly interspersing *Wikipedia* content with custom content. *WikiLayer* is driven by three main wiki principles: affordability (i.e., if you know how to edit articles, you know how to layer), organic growth (i.e., *layers* evolve in synchrony with the underlying articles) and shareability (i.e., *layers* can be shared in confidence through the wikipedian's social network, e.g., *Facebook*). The paper provides motivating scenarios for readers, contributors and editors. *WikiLayer* is available for download at `http://webaugmentation.org/wikilayer.xpi`.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Domain-specific architectures; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Collaborative computing*

General Terms

Human Factors, Design

Keywords

Web Augmentation, Wiki, DSL

WikiSym '12 Aug 27-29, 2012, Linz, Austria.

1. INTRODUCTION

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. Examples of what this technology generically enables include reorganizing page content, supplementing page data, changing fonts and formats, etc. [7, 11]. A popular example is the *Skype* add-on [15], a plugin that turns any phone number found in a Web page into a button that launches *Skype* to call that number. Another example is *LinkScanner* [4], an augmentation utility provided by *AVG* that permits to scan search results from *Google, Yahoo!* or *Bing,* and places a safety rating next to each recovered link, which informs about the trustworthiness of the site. This paper tackles Web augmentation for wikipedians by wikipedians.

Why. Web augmentation brings a kind of externalized customization: users can tune the front-end of a website based on their own needs. Specifically, wikipedians could add local content to augment the raw content of a *Wikipedia* article; or editors could add local annotations about the article's quality. Does this make sense? At first glance, the answer could be negative since, unlike traditional websites, *Wikipedia* permits users to contribute with content right away. No need for an additional *customization* tool. However, three rationales advice a more careful look. First, personal knowledge management. *Wikipedia*'s freedom of editing does not imply all editions becoming publicly available: the edition should be backed by the community. In some case, divergences might not be opinionated but reflect different goals to be fulfilled by the article. In this setting, *Wikipedia* augmentation might offer a backdoor for people to do more personalized exploration (hopefully followed by merging), instead of trying to converge on a consensus. Second, directly editing *Wikipedia* and hence, exposed to public scrutiny, might be too intimidating. Here, *Wikipedia* augmentation might account for a more personal and protected setting which can eventually spur future contributions. Third, augmentation as an annotation-like mechanism lowers the participation barrier. Between reading and publicly contributing, augmentation can provide a middle pier.

How. Web augmentation should be tuned to the "wiki way". Wikis are characterized as being open (i.e., edition is easily conducted without even require to log in), organic (i.e., wikis grow and shrink dynamically along the desires of the community that uses and natures them), and observable

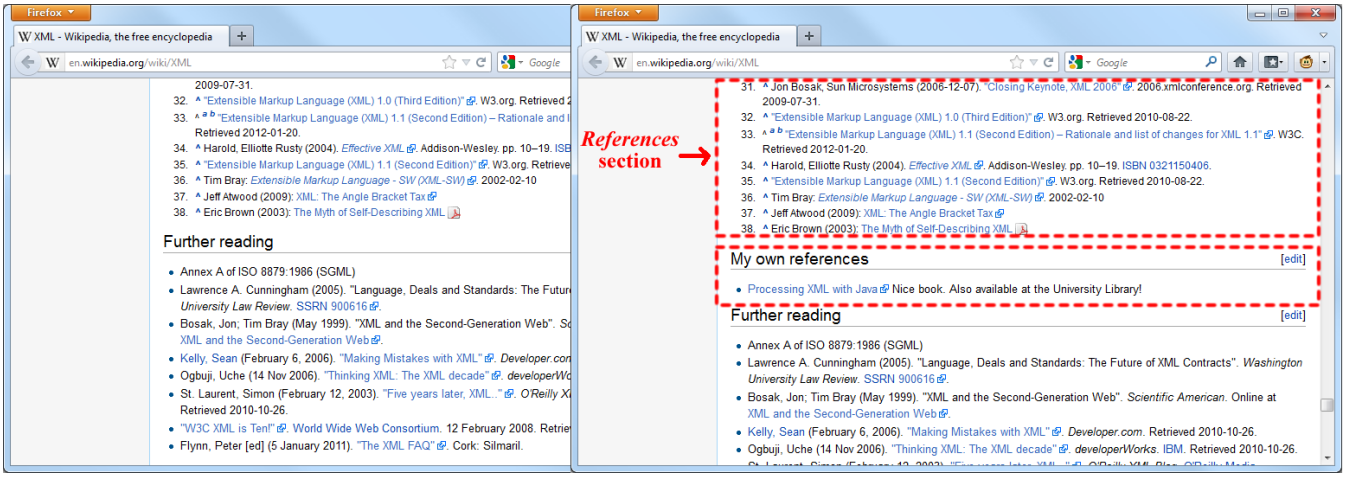

Figure 1: The *XML* article before (left) and after (right): adding a *layer*.

(i.e., changes are tracked and visible to the rest of the community) [1]. Likewise, augmenting *Wikipedia* should then be (1) affordable as the counterpart of open, i.e. the complexity should be similar to that of writing a piece of *wikitext*, (2) modular as the counterpart of organic, i.e. augmentation code should be provided in piecemeal fashion that might be eventually enlarged or reduced at user's wish, and (3), shareable as the counterpart of observable, i.e., your augmentation code should be easy to understand, share and install by other members of the community.

With these requirements in mind, we introduce *WikiLayer*, a plugin for *Firefox* that extends *Wikipedia* rendering with augmentation capabilities. So far, *Wikipedia* supports two modes for article interaction i.e. *article* and *talk*, which are realized through the namesake tabs. *WikiLayer* introduces a third mode: *layer*. *Wikipedia* pages are then extended with an additional tab i.e. *WikiLayer*, which permits users to *locally* supplement the wiki article with their own content, content obtained from other wikis or content obtained from other websites. Back to the *read* mode, the rendering will display both the original content and the custom content seamlessly integrated together. The vision is for end users to *augment Wikipedia* as easily as they currently *edit Wikipedia*. To this end, *WikiLayer* permits to *declaratively* (i.e., *affordable*) state *layers* over articles in a piecemeal fashion (i.e., *modular*) where *layer* sharing is limited to your acquaintance (e.g., *Facebook* followers) who are only a click-away from locally installing their own copy of the *layer* (i.e., *shareable*). We start by introducing some motivating scenarios.

2. MOTIVATING SCENARIOS

Wikipedia augmentation offers a backdoor for users to customize *Wikipedia* articles with their own content. This augmented content (referred to as *'layer'*[1]) very much depends on your goals as reader, contributor or editor. This subsection introduces distinct scenarios for each of these roles. For comprehension purposes, we provide some snippets of the augmentation language as we go along. The full expressiveness of the language is addressed in the next sections.

Readers: **layers as entryways to editing**. Reading has been characterized as "a gateway activity through which newcomers learn about Wikipedia" [3]. Reading *Wikipedia* spurs community engagement, and it is the entry to more involved activities such as editing. However, the gap between reading and editing could be too large for some wikipedians. Layering provides a middle pier in this transition by facilitating 'edition in private'. But layering is not editing for the sake of editing. Layering is editing with a purpose: extending/tailoring the coverage of a topic (i.e., an article). For instance, you can supplement your own local references for the *Wikipedia* article on the *XML* topic, indicating whether the book is in the University library. Figure 1 depicts the *XML* article before and after being augmented with the following *wikinote*:

> *LayerOnArticle("XML").*
>
> *AfterSection("References").*
>
> *EmbedNote("== My own references ==\n* [http://www.amazon.com/dp/0201771861 Processing XML with Java] Nice book. Also available at the University Library!")*

Using *wikitext*, this *wikinote* introduces a new section after section *"References"* of the article *"XML"*. From now on, navigating to *XML* will seamlessly introduce this reference on the fly (see Figure 1). Other *wikinotes* might impact a set of articles characterized by their membership to a *Wikipedia* category. For instance, you might augment articles pertaining to the *XML* category with a link to *delicious* with information about who has bookmarked this article. Thus, you can readily obtain a first feeling about the popularity of the article. The *wikinote* follows:

> *LayerOnArticle("Category:XML").*
>
> *BeforeSection("1").*
>
> *EmbedNote("See who has found this article interesting enough to be bookmarked in Delicious. [http://delicious.com/url/?url=en.wikipedia.org /wiki/$article CLICK HERE].")*

[1]A *layer* is composed by one or more *wikinotes*.

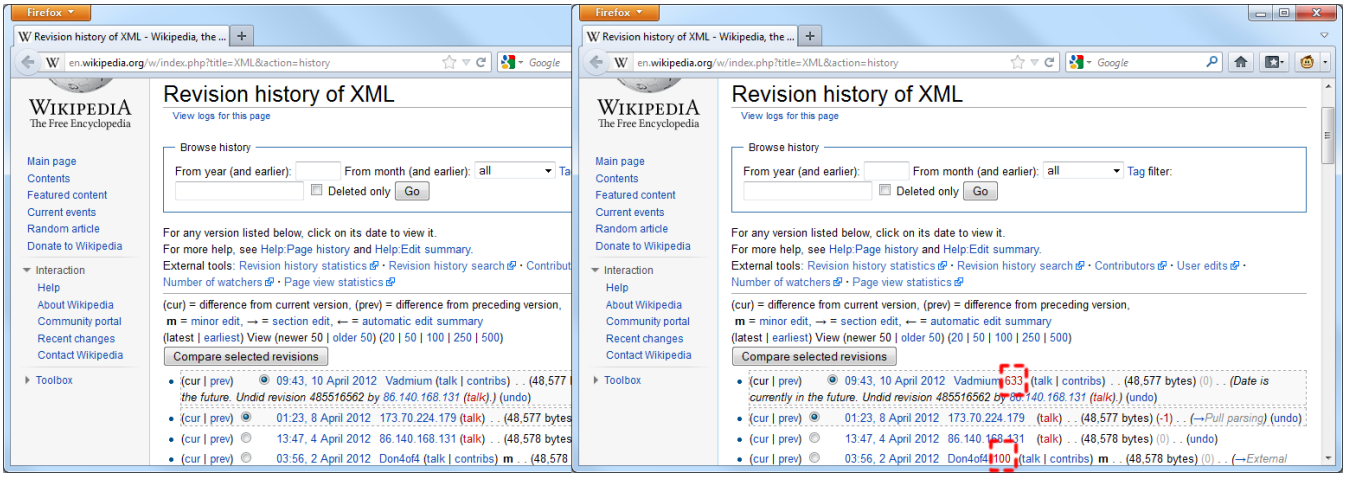

Figure 2: A history page being augmented with *Karma* information.

This *wikinote* applies to any article belonging to the category *XML*. At runtime, the topic of the current article (e.g., *XML Schema*) is kept in the variable *$article*. Similar variables are available to refer to the current section (*$section*), user (*$user*), ip (*$ip*) and other items.

Contributors: layers as enablers of perspective co-existence. Collaborative knowledge building is basically a spiralled process where knowledge first emerges at individual context and then it is socialized [13]. This process involves externalization, publication, internalization and reaction. Most wikis only support the knowledge socialization, but it is fundamental to support personal knowledge building too. Web augmentation offers a way for personal knowledge management to be structured along *Wikipedia* topics. This personal perspective might not be compatible with *Wikipedia*'s "neutral-point-of-view" policy [2]. Such neutrality leads to lean articles that focus on the bare essence of the topic at hand. That is, articles are devoided of any contextualized bias. Contextualization is not bad but addresses the topic from a specific perspective. For instance, the article *XML* restricts itself to introduce the rationales, history, criticism and core notions of this topic. This is sufficient for readers looking for an introduction to *XML*. However, consider the use of *Wikipedia* in a classroom. Besides the bare description of the topic, lecturers might be interested in providing additional teaching material (e.g., figures, commented bibliography, additional resources, hot trends, debates, etc.) along the structure and support offered by the *Wikipedia* article. Directly editing the article might be inconvenient (since adding teaching material is not the aim of *Wikipedia*) or just too intimidating. Augmentation permits a non-intrusive, self-consumption approach to extend *Wikipedia*. In some cases, these different perspectives might already co-exist in the wikisphere. For instance, the topic "Barcelona" is covered in both *Wikipedia* (providing factual information about the population, history, etc. of this city) and *Wikitravel* (facilitating travellers' opinions about where to stay, eat, visit, etc.). As another example, *Wikipedia* is being referred as a source of 'crowd-sourced' perspective that might not match academic standards. This grounds initiatives such as *'citizendium.org'* where contribution might be limited to experts or gently guided by experts. In this setting, you can layer the *XML Wikipedia* article with some sections obtained from other *citizendium*:

LayerOnArticle("XML").

AfterSection("Characters and escaping").

EmbedNote(extractSection("en.citizendium.org /wiki/XML","XML Specification and Origin"))

Editors: layers as productivity tools. Now, we change the focus to *history* pages. A history page contains a list of the page's previous revisions, including the date and time of each edition, the username or IP address of the user who made it, and their edition summary. Broadly, this data accounts for the main dimensions for assessing editing: who, what and when. However, this information might be insufficient for decision taking. Vandalism detection is a case in point. Recent studies [9] suggest the use of metadata (e.g., revision comment length, local time-of-day, etc.) or reputation (e.g., user reputation, country reputation, trust histogram, etc.), as valuable sources to detect vandalism. Some of these data might already be available on the Web: reputation data (a.k.a. *Karma*) can be obtained from `http://wpcvn.com/`; geolocation through IP address is available at `http://maxmind.com/`. Despite being online, navigating back and forth from the history page to these sites can be cumbersome and time consuming, if conducted routinely. Augmentation can help here. Editors can define a *layer* that dynamically augments the history pages at hand with additional information extracted from these places. Back to the *XML* example, we are now interesting in tracking the history of editions but augmented with the *Karma*. Figure 2 compares the "real" history page and the augmentation conducted by the following *wikinote*:

LayerOnHistory("XML").

AfterUser().

EmbedNote(extractFromPage("http://wpcvn.com/s /karma?username=$user"))

This *wikinote* extends the user description in the *XML* history page with his/her *Karma* as obtained from

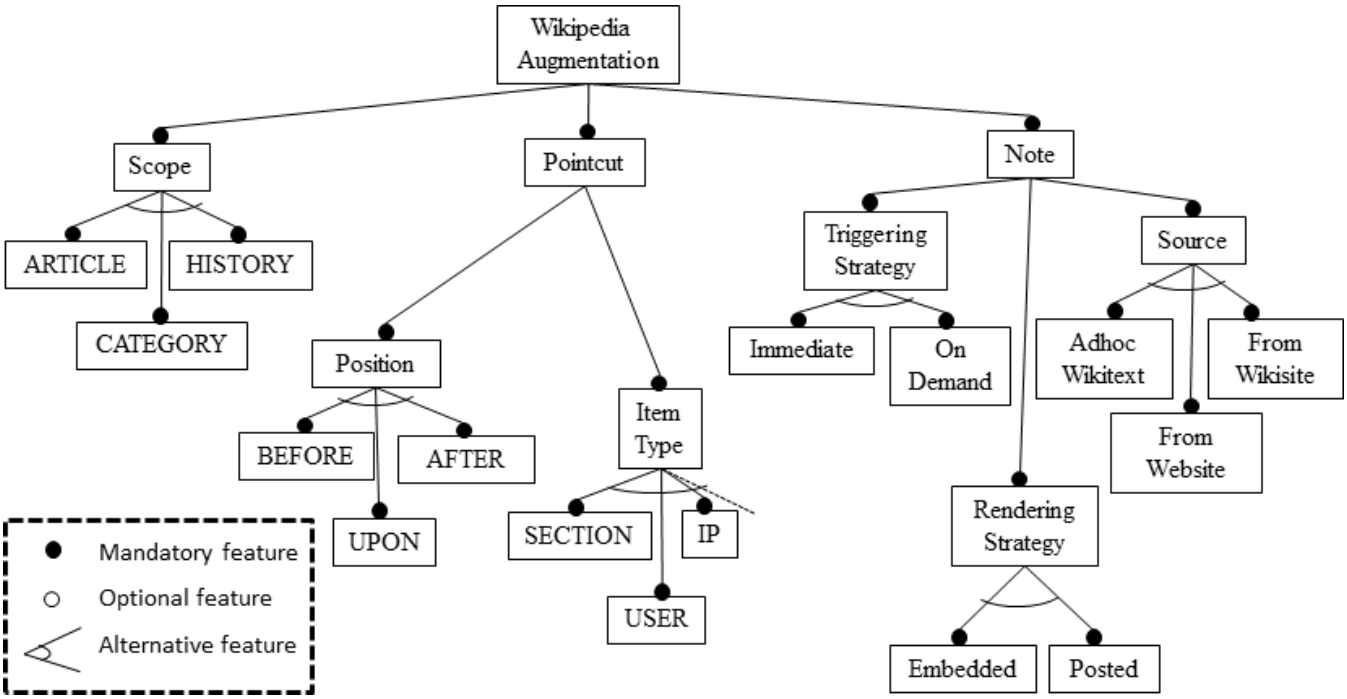

Figure 3: Feature Diagram: characterizing *Wikipedia* augmentation.

wpcvn.com. In this case, the ready availability of the *Karma* improves the productivity of the editor.

Next section introduces *WikiLayer*, a language for *wikinotes* definition integrated within *Wikipedia*. *WikiLayer* for Firefox and the *wikinote* samples are available for download at `http://webaugmentation.org/wikilayer.xpi` and `http://tinyurl.com/wikilayersamples`, respectively. *WikiLayer* has been tested against *Mozilla Firefox* 11.0, and the following wikis: *Wikipedia, Wiktionary, Wikinews, Wikibooks, Wikiquote, Wikisource, Wikiversity, Wikispecies, Wikitravel* and *Citizendium*.

3. A LANGUAGE FOR WIKINOTE DEFINITION

Web Augmentation is the act of superimposing additional directives on top of existing Web pages at run time [5]. The approach is non-intrusive in the sense that the augmented website is unaware of the augmentation. This is achieved through *JavaScript* using special *weavers* that permit a locally provided script to make on-the-fly changes to the currently loaded Web page. Unfortunately, *JavaScript* meets none of our requirements. Scripts are neither affordable (i.e., *JavaScript* is a convoluted programming language, ignored by most of the wikipedians) nor modular (i.e., scripts tend to be a bulk of code, difficult to enlarge and reduce along the *Wikipedia* article evolution) nor shareable (i.e., scripts tend to be poorly documented). The bottom line is that only dedicated programmers are disposed to *produce* scripts, and only courageous *consumers* are willing to install them. We strive to depart from this situation, heading to a vision of *Wikipedia* augmentation as a Web2.0 activity, i.e., end-user oriented.

To this end, we resort to Domain-Specific Languages (DSLs). DSLs are full-fledged languages tailored to specific application domains by using domain-specific terms [8]. Appropriately fixing the scope, the target audience, and the main abstractions of this domain will determine the success of the DSL. Our domain is *Wikipedia* augmentation. Our target audience is *wikipedians*. That is, we expect users to be familiar with the notions of e.g., article, category, edition, history, infobox or *wikitext*. As for the main abstractions, they are obtained after conducting domain analysis, i.e., looking at distinct scenarios of *Wikipedia* augmentation (see Section 2), and ascertaining the main domain concepts and their interdependencies. A main outcome of this analysis is a *feature diagram* [10]. For our purpose, a feature is a prominent and distinctive user visible characteristic to be tackled during *Wikipedia* augmentation. Next section introduces those features.

3.1 WikiLayer Feature Diagram

Insights gained by looking at previous scenarios, should now be made precise in terms of a feature diagram. Figure 3 shows such diagram for the *WikiLayer* DSL. The diagram states that a *WikiLayer* expression (hereafter referred to as a *wikinote*[2]) frames an augmentation within a *scope*. A *scope* holds *pointcuts* that pinpoint where the article content can be locally supplemented with a *note*. More specifically:

- **Scope**. *Wikipedia* comprises a huge bulk of pages. First, we should determine the focus of the augmentation effort. This includes the type of page (i.e., Article, Category or History) and the topic (i.e., a specific page such as the article about *XML*).
- **Pointcut**. The scope delimits the pages subject to augmentation. However, a page might offer different injection points. These points are denoted after

[2]Recall that *wikinotes* are the building blocks of *layers*.

```
<wikinote>           ::=  <scope> <pointcut> <note>

<scope>              ::= "LayerOn" <scopeType> <scopeValues>
<scopeType>          ::= "Article"    | "Category"    | "History"
<scopeValues>        ::= <scopeValue> | <scopeValue> <scopeValues>
<scopeValue>         ::= <wikipediaTopic>  | <wikipediaCategory>

<pointcut>           ::= <position> <itemType> <itemRef>
<position>           ::= "After"      | "Before"      | "Upon"
<itemType>           ::= "Section"    | "IP"          | "User"
<itemRef>            ::= <labelRef>   | <indexRef>

<note>               ::= <triggeringStrategy> <renderingStrategy> <noteSource>
<triggeringStrategy> ::= "" | "OnClickingButton" <buttonLabel>
<renderingStrategy>  ::= "EmbedNote" | "PostNote"
<noteSource>         ::= <wikitext>  | <extractItem> | <extractFromPage>
<extractItem>        ::= "extract"         <itemType> <url> <itemRef>
<extractFromPage>    ::= "extractFromPage" <url>
                     ::= "extractFromPage" <url> <pointInURL>

<wikipediaTopic>     ::=  string
<wikipediaCategory>  ::=  string
<labelRef>           ::=  string
<indexRef>           ::=  string
<buttonLabel>        ::=  string
<wikitext>           ::=  string
<url>                ::=  url
<pointInURL>         ::=  xpath expression
```

Figure 4: *WikiLayer* BNF.

the structural elements found in a page (referred to as "items"). For articles, items include *Sections*, *References*, *ExternalLinks*, etc. For history pages, items include *IP*, *User*, *Contribution*, etc. That is, items depend on the kind of page. In addition, a page is not a set but a sequence of items. An article is a sequence of sections. A history is a sequence of edition traces. Hence, a pointcut is a pair (position, item), for instance *(Before, "Characters and escaping" section)*.

- **Note**. A note is a *wikitext* expression. This note can be static (i.e., directly provided by the user) or dynamic (i.e., the outcome of a function). Functions permit notes to be extracted from Web pages in general, or wiki pages, in particular. Note displaying is governed by both a *rendering strategy* and a *triggering strategy*. The former indicates whether the note is to be *embedded* or *posted* w.r.t the raw page. Embedding implies the reader perceives no difference between the raw content and the augmented content. By contrast, posting makes augmented content visible by visualizing the note as a post on top of the *Wikipedia* page. As for the triggering strategy, it refers to when the note is to be shown up. Matching the scope might directly lead to show up the note (i.e., *immediate strategy*). Alternatively, notes can be shown on demand (i.e., *on-demand strategy*), whereby a user action is required for the note to surface (e.g., clicking a button, passing the mouse over a certain page region, etc.).

A feature diagram serves to state the commonalities and variabilities of the domain at hand, so that commonalities are built-in into the DSL engine whereas variabilities are supported as parameters to be set by the DSL user [12]. These parameters to-be-set-by-the-user are so provided as a DSL expression. This expression follows a concrete syntax. Next subsection provides the details.

3.2 The WikiLayer Language

This subsection introduces the *WikiLayer* language through examples. It also outlines the process of conceiving a *WikiLayer* expression (i.e., a *wikinote*). Figure 4 depicts the Backus-Naur Form (BNF) grammar.

Setting the scope. First, you should indicate *the kind of* pages you are going to act upon: *LayerOnArticle()*, *LayerOnHistory()*, etc. Next, you focus on the specific pages to be subject to augmentation:

1. *LayerOnArticle("XML")*
2. *LayerOnArticle(["XML","XPath"])*
3. *LayerOnArticle("Category:XML")*

This set can be described either extensionally by referring to the *Wikipedia* topic (examples 1 and 2) or intensionally through category membership (e.g., articles that belong to the *XML* category, example 3).

Setting the note. Next, you focus on the new material to be added, i.e., the note. *WikiLayer* supports three options: (1) directly providing the *wikitext*, (2) extracting the note from other wiki pages or (3), extracting the note from other *HTML* pages. Examples follow:

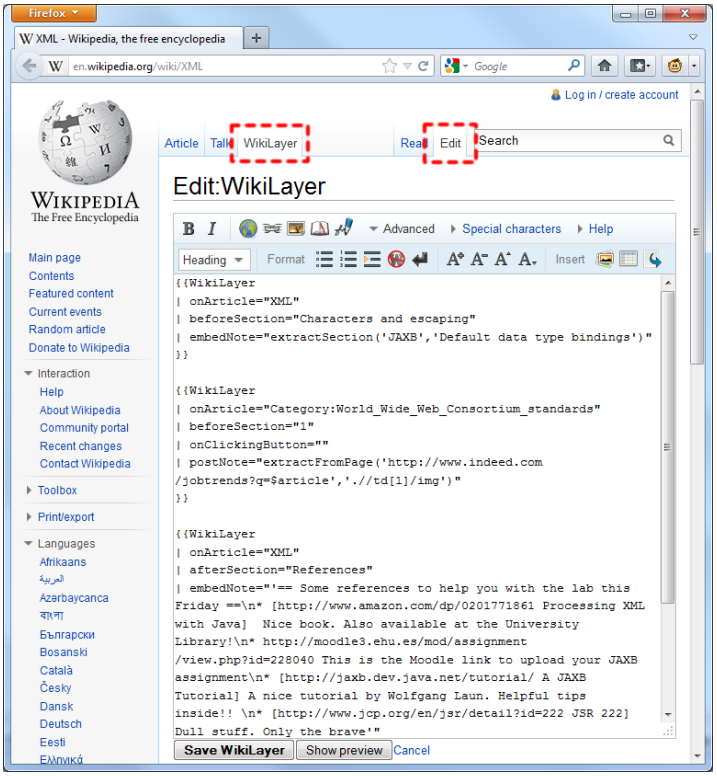

Figure 5: The tab *Edit* when in the *WikiLayer* mode. *Wikinotes* are specified using a template while *wikinote* verification is conducted on showing preview.

1. *"===Java Architecture for XML binding===\n This allows Java developers to map Java classes to XML representations. A nice tutorial can be found [http://jaxb.dev.java.net/tutorial/ here]")*
2. *extractSection("en.citizendium.org/wiki/XML", "XML Specification and Origin")*
3. *extractFromPage("www.vogella.de/articles /JAXB/article.html")*

Example 1 explicitly provides the note as a piece of *wikitext*. If you know how to edit an article, then you know how to write a note. By contrast, examples 2 and 3 take the note from the websphere. Here, we use a transclusion-like approach whereby the inclusion is performed on demand at the time the *Wikipedia* article is loaded. The location is described in terms of the page's URL and a region within this page. For wiki pages, these regions stand for items (e.g., *Section*, *Id*, *Timestamp*, *TotalLength*, etc.). *WikiLayer* provides namesake functions to extract the items from wikis (e.g., *extractSection()*, *extractUser()*, etc.). Example 2 extracts the section "*XML Specification and Origin*" from the *XML* article at *citizendium*.

If the source is not a wiki (better said, a MediaWiki-powered wiki), then the note should be manually pinpointed by the user. Here, it is not clear which kind of "items" should be introduced to play the role of references for note extraction. In the lack of items, you can resort to *XPath* to pinpoint the *HTML* region to extract. But most wikipedians probably ignore *XPath*. As a result, we resort to programming-by-example. Wikipedians provide only the URL parameter to this function: *extractFromPage()*. The first time the *wikinote* containing *extractFromPage()* is enacted, the engine automatically navigates to this URL and intersperse a grid-like structure on top of the current *DOM* tree (i.e., the runtime tree-like structure of *HTML* pages). As the user moves the cursor around the screen, the *DOM* node under the current cursor location is highlighted. By clicking, the user makes up his mind about the fragment to be extracted, and the *wikinote* becomes bound to the so-identified *XPath*. Subsequent enactments of this *wikinote* will directly extract this region.

Setting the injection location. This addresses in-context presentation of *wikinotes*, i.e., how *notes* are integrated in the original wiki content. This is realized by the rendering strategy and the triggering strategy. Some examples follow:

1. *BeforeSection("XML as data type").EmbedNote(...)*
2. *BeforeSection("XML as data type"). OnClickingButton().PostNote(...)*

The rendering strategy has a two-fold implication. First, declaring the pointcut in the form "*PositionItem()*" where "*Position*" can be "*Before*", "*After*" or "*Upon*", and "*Item*" can be any item type (e.g., *BeforeSection()*, *AfterTimestamp()*, *UponTotalLength()*). Second, deciding whether the note is to be showed up on demand. Example 2 illustrates this option: the clause *OnClickingButton()* will cause a button to be rendered whose clicking is necessary for the note to pop up. As for the triggering strategy, it determines whether the note is to be embedded (*EmbedNote()*) or posted (*PostNote()*).

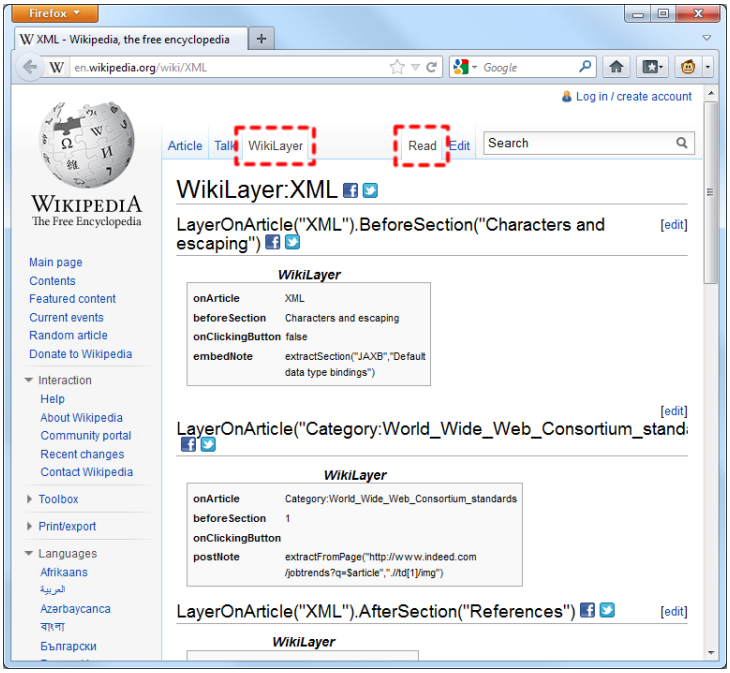

Figure 6: Clicking the *Read* tab when in the *WikiLayer* mode.

Putting the pieces together. This relates to the fluency of the DSL, i.e., making the expression easy to read and understand. At this regard, we prioritize setting the context, i.e., the scope of the note. Hence, *WikiLayer* syntax starts with the scope, next, the pointcuts, and finally, the note. In brief, a *wikinote* looks like follows:

LayerOnArticle("XML").

BeforeSection("XML as data type").
OnClickingButton().

EmbedNote(extractSection("en.citizendium.org/wiki/XML", "XML Specification and Origin")).

4. FRAMING WIKILAYER INTO WIKIPEDIA

Hiding *JavaScript* from wikipedians is not enough. We should strive to frame *WikiLayer* in its context of use, i.e. *Wikipedia*. It should look like *WikiLayer* is part of *Wikipedia*! Our aim is to make *layer* editing an "impulsive" action so that editing can occur at the time and at the place where users consult *Wikipedia*, i.e. the browser. Back to our sample, the user is reading the *XML* article, he comes up with a new reference, and, right at this time, he is impelled to enhance the *layer*. The aim is to drive this impulse at the time it rises. This calls for a seamless integration of *WikiLayer* within the *Wikipedia* front-end, i.e. Web-augmenting *Wikipedia* with *WikiLayer* functionality. Such functionality includes *layer* edition, verification, maintenance and sharing.

Edition. So far, *Wikipedia* supports two modes for article interaction: describing an article, and talking about an article. *WikiLayer* envisages annotations as a third mode: besides describing and talking, articles can also be subject to annotation. *Wikipedia* uses tabs to reflect modes: the *Article* tab and the *Talk* tab. Accordingly, *WikiLayer* introduces a third tab: the *WikiLayer* tab (see Figure 5). By clicking on this tab, the *Read* and the *Edit* tabs become online editors for your *layer*. Click the *Edit* tab. Now, you are ready to provide your *wikinotes*. Akin to wiki editing practices, *wikinotes* are specified using a template. Each of the language clauses are declared as a template parameter. Figure 5 illustrates the case of a lecturer who augments the *XML* article for teaching purposes. This includes: (1) a new *section* about *JAXB* obtained from the *Wikipedia* itself; (2) a new graph about the evolution of *XML* as a keyword found in online job posting as provided by `http://www.indeed.com/`, (3) an extension of the *References* section with commented bibliographical entries provided in terms of *wikitext*; and (4) an extension to the *XML Schema* subsection with links to a tutorial about this topic at `http://www.zvon.org/`. You can download this *layer* from `http://tinyurl.com/wikilayersamples2`.

Verification. Akin to wiki editing practices, before saving a *layer*, it is convenient to obtain a preview. Here, this mechanism is used to verify the syntactic correctness of your *wikinotes*. Any syntactical error will be spotted and reported in the preview. If you directly go to "save" without preview, the verification is still conducted but just a brief error message is issued, should there be any problem. Once *wikinotes* are syntactically correct, the engine stores them in the browser. This entails the article at hand will be automatically augmented the next time it is loaded. If the *wikinote* extracts content from websites other than wikis, the first enactment will require to graphically set the *XPath* that recovers the desired *HTML* fragment (e.g., the job-trend graph at *indeed.com*). From then on, *WikiLayer* will seamlessly intersperse the local content (i.e., the *wikinote*) with the remote content (i.e., the original article).

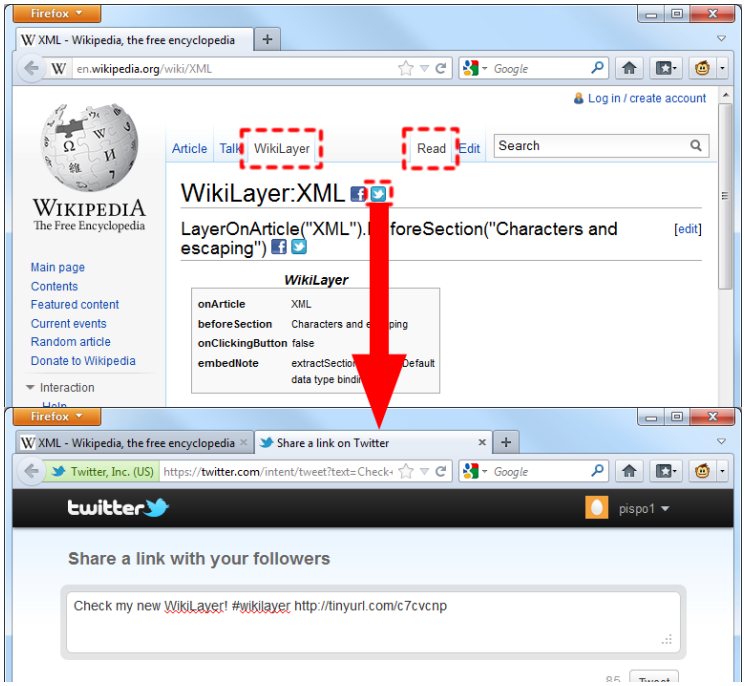

Figure 7: The tab *Read* when in the *WikiLayer* mode. Clicking the *Facebook/Twitter* icon pops up a window to edit a message. The message includes a *tinyURL* generated by *WikiLayer* as a representation of the *layer*. By clicking on this URL, receivers can install the *layer* right away.

Maintenance. Once defined, *layers* are prone to change. Rationales include: (1) *layers* are rooted in a moving target, *Wikipedia* pages (e.g., new sections, paragraphs, etc. can be added or removed, and this will likely percolate to their *layer* counterparts e.g., if you refer to sections based on their numbers then, introducing a new section will no longer properly locate your *layer*); (2) augmentation is a process: new *layers* are envisaged as you come across with interesting stuff in the Web.

WikiLayer is designed with this dynamicity in mind: *layer* modularity helps to easily add/remove *wikinotes*. When on the *WikiLayer* mode, you can see which *wikinotes* operate on the current article by clicking the *Read* tab (see Figure 6). From then on, you can add/remove *wikinotes* at your wish.

Sharing. Annotation alone might not be enough. Sharing might increase not only the quality of *layers* but also the wikipedian's confidence to consolidate the *layer* as part of the original article. Hence, sharing might play a pushing role in this transition from reading to editing. By its very own nature and purpose, *layer* sharing departs from *Wikipedia* article sharing. Rather than a central repository, we regard social networks as more appropriate for *layer* sharing.

Being plain text, *layers* can be easily emailed. But text is no longer the most convenient way of sharing in the Web: URL bookmarks are. *WikiLayer* turns *layers* into URLs so that you can easily share them through *Twitter* or *Facebook*. Figure 7 shows this utility in action. The rendering of *layer* is now decorated with the icons of these social networks. On clicking the *Facebook* icon, the wikipedian is publishing his *layer* URL into his wall. Likewise, pushing the *Twitter* icon creates a tweet that includes the *layer*'s URL. By clicking this URL, followers download the *layer* right away. No need to go to a repository. Of course, this requires followers to install the *WikiLayer* plug-in.

5. RELATED WORK

This work can be framed within the area of Personal Knowledge Management through wikis [14]. Users require means for creating, combining and adapting information in an isolate and guesstimate way before eventually sharing the outcome with their mates for further refinement. At this respect, approaches can be arranged along a continuum from completely detached wikis (a.k.a. personal wikis), passing from P2P approaches (where your personal content can be shared in a pair-like way but not central server exists) to versioning-like architectures where you can start by checking out from a traditional wiki, create your own branch and then merge the content back. We compare these different approaches along a set of dimensions (see Table 1): the *scope* (whether the reach is limited to a single wiki or expands across different wikis), the *contribution type* (what the user contribution is: a whole wiki, an article, an item, a semantic annotation), and finally, the *architectural style* (i.e., standalone, client/server, P2P). The goal is not to provide an exhaustive list of endeavours but outline the main differences with *WikiLayer*. For completeness sake, we also include in the comparison Web annotation tools.

Web Annotation Tools. Being a Web application, *Wikipedia* can benefit from Web annotation tools such as *Diigo* or *A.nnotate*. Nothing wrong with it. However, our premise is that *Wikipedia* (unlike other websites) should look at annotation as a mean to achieve its very own goals: reading and editing articles. While annotating a University website is not directly related with the University goals (i.e.,

Table 1: Personal Knowledge Management: wiki-based approaches.

	Contribution Type	Scope	Architectural Style
WikidPad	wiki	intra-wiki	standalone
P-Swooki	semantic annotatation (tag)	intra-wiki	P2P
Hatta	wiki	intra-wiki	client-server
Federated Wiki	wiki/article	inter-wiki	P2P
WikiLayer	item	inter-wiki	standalone

educating), *Wikipedia* is all about engaging the crowd in article contribution. From this perspective, annotation is no longer an ancillary activity but a main mean to fulfil *Wikipedia*'s ends. By using general-purpose Web annotation tools, we miss the opportunities brought by a *Wikipedia*-specific annotation tool: same rendering experience (i.e., annotation as an *article* mode), in-context presentation (i.e., annotations intermingled with original content), or a 'wiki-ish' annotation description (i.e., use of *wikitext* or transclusion mechanisms). In brief, making annotation a natural gesture for wikipedians. *WikiLayer* is an attempt in this direction.

Personal Wikis. A personal wiki is like a traditional wiki but with a single user. *WikidPad*[3] is a case in point (for a list, refer to `http://c2.com/cgi/wiki?PersonalWiki`). *WikidPad* defines itself as "a Wiki-like notebook for storing your thoughts, ideas, todo lists, contacts, or anything else you can think of to write down. *WikidPad* is like an IDE for your thoughts". The main difference with *WikiLayer* stems from the starting point: *WikiLayer* pivots around an existing wiki. Unlike *WikidPad*, *layers* cannot be created in a vacuum but they are anchored in existing articles. *Layers* look more like annotations.

Semantic Wikis. A semantic wiki allows users to make formal descriptions of resources by annotating the pages that represent those resources. Where a regular wiki enables users to describe resources in natural language, a Semantic wiki enables users to additionally describe resources in a formal language. This facilitates the structuring (hence, querying) and potential reusing of the wiki content. The *P-Swooki* effort complements semantic wikis by introducing the personal perspective [16]: personal semantic annotations are associated to the wiki page and they can only be accessed by the owner user. In this way, personal annotations support the individual understanding in the collaborative knowledge building process while providing personalized knowledge retrieving, structuring and navigation [16]. Users keep their personal annotations local (i.e., the article tags), which can eventually be blended with the publicly visible tags of the semantic wiki. *WikiLayer* shares the same spirit: ability to annotate existing wiki material. The difference stems from the subject of contribution: semantic annotations (tags) in *P-Swooki versus* items (e.g., sections) in *WikiLayer*. In addition, *WikiLayer* favours a mashup approach where material from different wikis can be easily mixed together whereas *P-Swooki* is intra-wiki. And the other side of the coin, *WikiLayer* does not support automatic merging of *layers* into wiki articles whereas *P-Swooki* does.

Wiki Versioning. These tools are inspired by software versioning and revision control utilities like *SVN* or *Git*. Generally, the engine can be installed locally and at the same time, on the server, and periodically (normally on demand) synchronize both installations. Moreover, other users can also have their own local installations which are also centrally synchronized. Normally, the process starts by creating a local clone (a.k.a. check out) (including page history). Examples include *Hatta*[4], or *Firestarter* for *Confluence*[5]. *Firestarter* is described as "a wiki on a USB drive". The envisaged scenarios include working on the wiki while offline. Wiki versioning shares with *WikiLayer* the fact of starting with existing content. However, the ending is not necessarily a blend with the ground article. Rather, *layers* are prone to become "personal views" over existing article by tailoring it to some new context (e.g., teaching). This resembles "forking" as supported by versioning packages whereby a new project is initiated out of the base one. From this perspective, *WikiLayer* can be regarded as a lightweight wiki-oriented approach to article versioning, though no automatic merging is supported. Notice the difference in granularity: *WikiLayer* versions articles while *Hatta*-like applications version the wiki as a whole. In addition, versioning systems like *Git* focus on a code unit which can next be forked if appropriate but the coordination model is that of branching from a single core. Moving away from this centralized approach, we reach federated wikis.

Federated Wikis. At the moment, wiki content is kept within the walls of the wiki engine. Export and import utilities exist, but there is no feature specifically designed to share content across wiki repositories. Pioneer by Ward Cunningham [6], federated wikis strive to open wiki content in a controlled way. Federation has a two-fold implication. First, wikis stay in control of the inflow and outflow streams of wiki content. Second, wikis are engineered for collaboration. For instance, Cunningham's engine rests on the existence of a common *JSON* representation for articles. It does not matter how engines obtain this *JSON* as long as they follow this common format. This *JSON* format becomes the *lingua franca* for exchanging content from different wikis. In addition, articles keep a log, i.e., a trace of the different operations conducted over the article at hand. These operations include "edit", "add", "remove" and the like. It is also possible to "fork" an article. This clones the article to your wiki. From then on, the original article and the clone have different lifecycles, though you can keep an eye on the clone and eventually integrate some of its content. Federated wikis are certainly a step ahead in knowledge sharing. *WikiLayer* complements this view by adding "personal" on top of it. Edition and sharing are conducted within a personal realm: sharing through your friends in *Facebook* or followers in *Twitter*, and edition through a transparent local repository.

[3] `http://wikidpad.sourceforge.net/` (accessed June 2012).

[4] `http://hatta-wiki.org/` (accessed June 2012).

[5] `http://www.appfire.com/enterprise/firestarter/` (accessed June 2012)

6. CONCLUSIONS

We introduced *WikiLayer*, an augmentation facility targeted to wikipedians. *WikiLayer* provides a lightweight, seamless, client-based approach to supplement existing wiki articles with additional content, potentially brought from other websites (wikis or not). The approach has been carefully designed to wikipedians: *layer* design is along wiki concepts (e.g., section, *wikitext*), *layer* syntax resorts to wiki templates, and *layer* management is achieved through wiki-like pages. This endeavour is framed within the efforts to blend social knowledge management and personal knowledge management. From this perspective, *WikiLayer* introduces the personal perspective in wikis.

This work shows the technical feasibility of the augmentation approach. However, the ground premises need yet to be demonstrated. Our next follow-on is to conduct validation experiments to check the following hypothesis: (1) readers using *layers* are more inclined to become editors, (2) wikipedians look at *layer*-enhanced wikis as appropriate hubs to collect web-based material and (3), *layer*-based editing surveillance leads to more frequent monitoring and hence play the advantage of the quality of the monitored articles.

7. REFERENCES

[1] What is a Wiki. Online at www.usemod.com/cgi-bin/mb.pl?WhatIsaWiki.

[2] Wikipedia's Neutral Point of View Guidelines. Online at en.wikipedia.org/wiki/Wikipedia:Neutral_point_of_view.

[3] J. Antin and C. Cheshire. Readers are Not Free-Riders: Reading as a Form of Participation on Wikipedia. In *2010 ACM Conference on Computer Supported Cooperative Work, CSCW '10*, pages 127–130, 2010.

[4] AVG. AVG LinkScanner - How it Works, 2010. Online at linkscanner.avg.com/ww.sals-how-it-works.html.

[5] N. O. Bouvin. Unifying Strategies for Web augmentation. In *Proceedings of the 10th ACM Conference on Hypertext and Hypermedia, HYPERTEXT'99*, pages 91–100, 1999.

[6] W. Cunningham. Smallest Federated Wiki. Online at wardcunningham.github.com.

[7] R. E. Filman. Taking Back the Web. *IEEE Internet Computing*, 10:3–5, 2006.

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

[9] S. Javanmardi, D. W. McDonald, and C. V. Lopes. Vandalism Detection in Wikipedia: A High-Performing, Feature-Rich Model and its Reduction Through Lasso. In *7th International Symposium on Wikis and Open Collaboration, WikiSym '11*, pages 82–90, 2011.

[10] 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 Software Engineering Institute, 1990.

[11] N. McFarlane. Fixing Web Sites with Greasemonkey. *Linux Journal*, 138:1, 2005.

[12] M. Mernik, J. Heering, and A. M. Sloane. When and How to Develop Domain-Specific Languages. *ACM Computing Surveys*, 37:316–344, 2005.

[13] I. Nonaka and H. Takeuchi. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, 1995.

[14] E. Oren, M. Völkel, J. G. Breslin, and S. Decker. Semantic Wikis for Personal Knowledge Management. In *17th International Conference on Database and Expert Systems Applications, DEXA '06*, pages 509–518, 2006.

[15] Skype. Skype button in Internet Explorer or Firefox toolbar, 2005. Online at www.skype.com/intl/en/support/user-guides/toolbar.

[16] D. Torres, H. Skaf-Molli, A. Díaz, and P. Molli. Supporting Personal Semantic Annotations in P2P Semantic Wikis. In *20th International Conference on Database and Expert Systems Applications, DEXA '09*, pages 317–331, 2009.