Wiki Refactoring as Mind Map Reshaping

Gorka Puente and Oscar Díaz

Onekin Research Group, University of the Basque Country (UPV/EHU),
San Sebastián, Spain
{gorka.puente,oscar.diaz}@ehu.es

Abstract. Wikis' organic growth inevitably leads to wiki degradation and the need for regular wiki refactoring. So far, wiki refactoring is a manual, time-consuming and error-prone activity. We strive to ease wiki refactoring by using mind maps as a graphical representation of the wiki structure, and mind map manipulations as a way to express refactoring. This paper (*i*) defines the semantics of common refactoring operations based on *Wikipedia* best practices, (*ii*) advocates for the use of mind maps as a visualization of wikis for refactoring, and (*iii*) introduces a DSL for wiki refactoring built on top of *FreeMind*, a mind mapping tool. Thus, wikis are depicted as *FreeMind* maps, and map manipulations are interpreted as refactoring operations over the wiki. The rationales for the use of a DSL are based not only on reliability grounds but also on facilitating end-user participation.

Keywords: Wiki, refactoring, DSL, mind maps, end-user.

1 Introduction

Wikis are becoming mainstream for knowledge formation and sharing [14]. Consubstantial to knowledge formation is exploration, tentative guessing and trial-and-error practices. That is, knowledge formation goes together with regular knowledge revision. In a wiki setting, this knowledge (i.e., content) and its structure evolve with its supporting community (a.k.a. the wiki's Organic Principle [4]). In practice, this ends up in large structures of articles and categories which constantly need manual refactoring. Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behaviour [8]. Our premise is that for wikis, this "external behaviour" (i.e., the invariant to be kept during refactoring) is the wiki content and authorship. Wiki refactoring can change the wiki's internal structure for the sake of navigability, accessibility or comprehension, but the content (and its authorship) should be kept immutable.

Unfortunately, wiki engines (e.g., *MediaWiki*[1]) are thought for spurring contributions (certainly the cornerstone of this approach) but overlook refactoring. As a result, wiki refactoring is far from trivial. For instance, merging/splitting

[1] `www.mediawiki.org` (accessed 19-Mar-12).

J. Ralyté et al. (Eds.): CAiSE 2012, LNCS 7328, pp. 646–661, 2012.

two wiki articles requires of at least five interactions in *MediaWiki*. In other words, the semantics of refactoring is not natively supported by the wiki engine. The implications are twofold. First, refactoring is left to the user interpretation. Different users can face the same refactoring problem with different strategies. Although best practices are textually documented[2] [12], the wiki engine does not ensure coherence among the refactoring strategies used throughout the wiki lifespan. Second, the engine does not ensure refactoring reliability. Refactoring operations behave like database transactions in the sense that they comprise a sequence of wiki interactions that (*i*) should be performed in an all-or-nothing manner, and (*ii*) should move the wiki to a consistent state (i.e., wiki content must be preserved). This operational semantics is certainly not supported in current wiki engines but on the minds of the wiki users. Consequently, users are left unassisted with the cumbersome task of refactoring.

We advocate for wiki refactoring to follow the main wiki hallmarks [4], namely:

- *Open*, which implies lowering the barriers for layman participation. This tenet entails refactoring to be conducted with minimal disturbance (i.e., reducing "accidental complexity") and in terms closer to the user. This calls for the introduction of Domain-Specific Languages (DSLs) [13] that help users to conduct refactoring in high-level terms.
- *Observable*, which requires wikis to track changes as well as providing pervasive peer-review mechanisms. To counteract potential misbehaviour, the community can detect and reverse malicious editions. Refactoring should also be observable. Although refactoring keeps wiki content changeless, it can rearrange the very same content along a different set of articles and categories. Such rearrangement should be traceable while preserving authorship attribution.

This work contributes to the area of wiki refactoring by (*i*) identifying main wiki refactoring constructs, (*ii*) providing the operational semantics for these constructs, and (*iii*) introducing a graphical DSL for these constructs, i.e., *WikiWhirl*. *WikiWhirl* does not achieve anything that cannot be obtained by directly interacting through the *MediaWiki* front-end. The difference stems from productivity (how long does it take?), accessibility (who can do the refactoring?) and reliability (are authorship preserved?). *WikiWhirl* is available to download at `www.onekin.org/wikiwhirl`.

The paper starts by highlighting the differences between wiki and database refactoring (Section 2). Next, we provide a refactoring session as a motivating scenario (Section 3). We introduce the abstract syntax for *WikiWhirl* (i.e., the definition of the concepts of the language and their relationships) and its concrete syntax in Section 4 and Section 5, respectively. Related work and some conclusions end the paper.

[2] `http://en.wikipedia.org/wiki/Wikipedia_guidelines` (accessed 19-Mar-12).

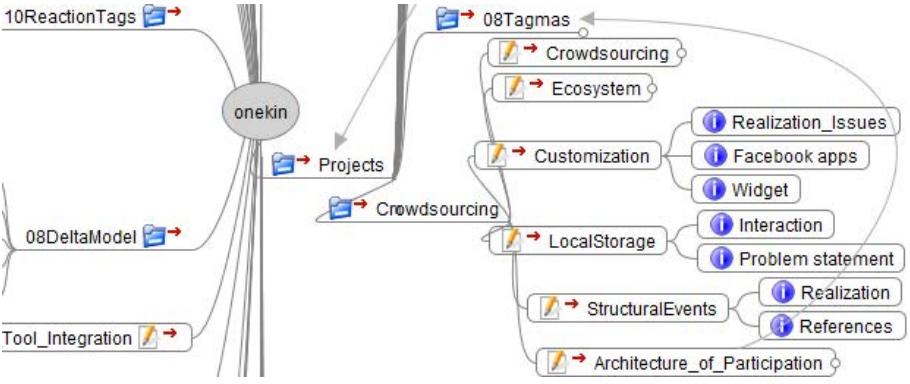

Fig. 1. The *Onekin* wiki depicted as a mind map

2 Wiki Refactoring *versus* Database Refactoring

At first sight, wiki refactoring does not look a big deal. After all, Ward Cunningham defines wikis as "the simplest online database that could possibly work"[3]. And database refactoring is a long lasting research topic [1]. However, the fact that wikis are databases does not mean that wiki refactoring equates with database refactoring. Differences stem from the "what", the "how" and the "who" conducts the refactoring.

What. Database refactoring mainly focuses on the database schema. At most, tuples might be subject to co-evolution, i.e., a migration process from the old schema to the new schema. By contrast, wiki refactoring is not about the schema of the database that keeps the wiki content. These tables are set by the wiki engine (e.g., *MediaWiki*): all *MediaWiki* wikis use the very same database schema. The wiki is not the schema but the data, i.e., the tuples. Therefore, wiki refactoring entails changing the data, not the schema.

How. Logical independence is a solid principle of database operation whereby changes in the database schema should minimally disturb client applications. A similar issue arises when facing API evolution, leading to the so-called backward compatibility. In general, this notion of "logical independence-ness" rises from any *share* resource that evolves: let it be a database schema, a component or a software library. Wikis are shared resources. The question is then what can be affected by wikis' evolution. While applications are impacted by changes in the database schema, wikis impact end users in their double role of readers and authors. Changing the wiki structure (as result of a refactoring) should cause minimal interference on these activities (i.e., reading and authoring). Hence, we introduce two dimensions of independence for wikis:

[3] `http://wiki.org/wiki.cgi?WhatIsWiki` (accessed 19-Mar-12).

– *Readership independence.* Refactoring does not alter the content but how this content is distributed among articles or categories. Wiki readers should be informed of where content has been moved to. In addition, wiki articles are Web resources users can bookmark. Hence, readership independence also includes the ability for a wiki to preserve URL addresses upon evolving articles/categories.
– *Authorship independence.* Acknowledging the authorship has been reported as a main motivator of contributions [2], and it is one of the directives of *Wikipedia*. Wiki refactoring must preserve authorship.

Who. The complexity and criticality of DBMSs require of dedicated users: the database administrators. By contrast, wikis promote openness and accessibility. This is realized as minimizing the skills to operate upon wikis. Unfortunately, wiki refactoring is not technically obvious. Not only is it time consuming but also error prone. Defective refactoring can compromise wiki coherence (e.g., no common understanding of how to conduct article merge) as well as authorship and readership independence. As a result, those that know *what to refactor* (i.e., the domain experts that know which articles are better merged) might ignore *how to refactor* (i.e., the operative that goes to properly merge two articles). Means are needed for end users to refactor their own body of knowledge by themselves. Next, we provide an example of such a refactoring process.

3 A Refactoring Session

A wiki is basically a graph of articles and categories (i.e., keywords for article characterization). For the time being, let us depict this graph in a radial way, where the center is the wiki's title, the leaves are the wiki's articles, and categories are in-between nodes. Fig. 1 provides an example for the wiki of the *Onekin* research group. This wiki documents ongoing research projects. Projects are reflected as wiki categories (e.g., *Crowdsourcing)* whereas project issues become wiki articles (e.g., *Ecosystem*).

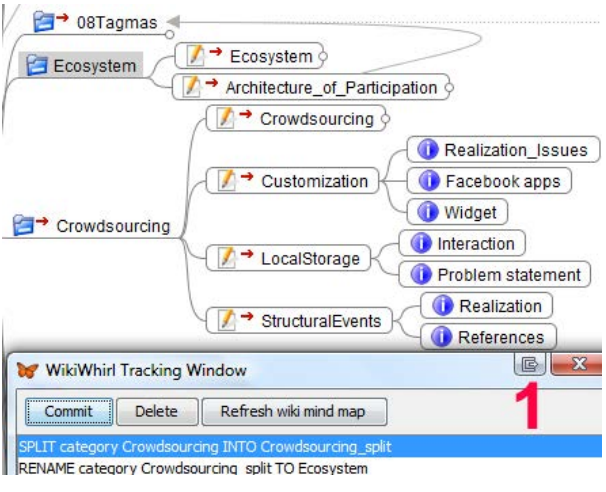

Fig. 2. Split operation during a refactoring session

For its very nature, research projects are conceived and developed on the go. Wikis perfectly fit this way of working. For instance, gains in understanding what *Crowdsourcing* is about, can impact the wiki as follows (Figs. 2, 3 and 4):

1. **Splitting** *Crowdsourcing*. Rationale: the issue of *Ecosystem* has evolved into a new matter on its own,
2. **Merging** articles *LocalStorage* & *StructureEvent* into a *Realization* article. Rationale: wide overlap between these issues,
3. **Moving** the section *Realization_Issues* from *Customization* to the *Realization* article. Rationale: *Realization* should frame all realization concerns no matter where they arise.

These scenarios should be conducted through the wiki's front-end. However, wikis' front-end favours easy contribution rather than refactoring. Moving or merging implies moving back and forth between the affected wiki articles. In addition, the user should care for preserving authorship: if you move a section, authorship should be moved as well by leaving appropriate references. This entails to handle redirects, talk pages and recent changes. In short, the previous splitting-merging-moving process may take a long time in *MediaWiki*.

Notice however, that it is not only a question of efficiency but also of reliability and coherence. Some refactoring operations might involve more than one interaction on *MediaWiki*. The *user* should ensure such set is handled as "a transaction" in the sense of atomicity and coherent meaning. In addition, the process should care for authorship and readership independence.

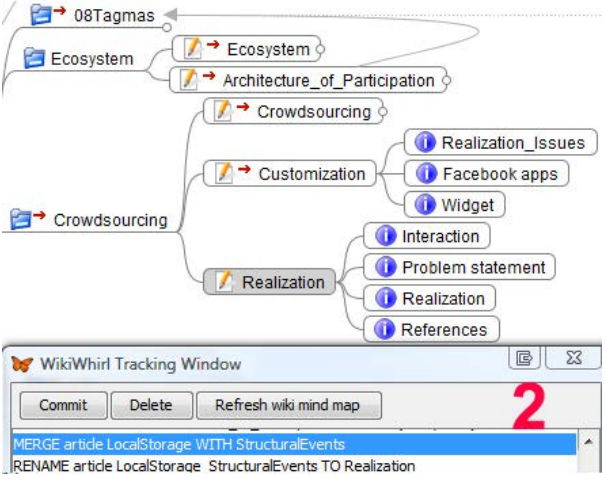

Fig. 3. Merge operation during a refactoring session

We advocate for abstracting the terms in which refactoring is conducted. That is, moving from low-level UI interactions to high-level domain-specific operations. To this end, we investigate the use of Domain-Specific Languages (DSLs). This entails systematizing a set of refactoring operations and their semantics (the DSL's abstract syntax), and nailing down a way to specify them (the DSL's concrete syntax). The output is the *WikiWhirl* DSL.

4 WikiWhirl's Abstract Syntax

The abstract syntax describes the concepts of the language, the relationships among them, and the structuring rules that constrain the model elements and their combinations in order to respect the domain rules [19]. This is expressed as the DSL metamodel. Fig. 5 depicts the *WikiWhirl* metamodel. A *WikiWhirl* process (*WW_Process*) includes: (*i*) a *Wiki* model, and (*ii*) a set of *WikiWhirl* refactoring operations *(WW_Operations)* on the *Wiki* model.

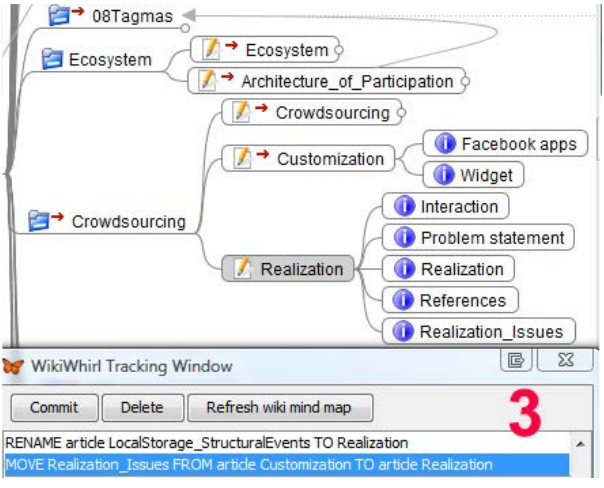

Fig. 4. Move operation during a refactoring session

The Wiki Model. It conceives the wiki as an aggregate of resources (i.e., *categories*, *articles* and *sections*). Wiki content is supported through *sections*[4]. Notice that sections are abstractions over existing wikis. Wiki engines do not support sections as independent artefacts but embedded as part of the wikitext of pages. However, we promote sections as full-fledged class elements due to its importance during refactoring. Categories play a double role: as pages, they describe what the category is about; as tags, they are used to organize and locate articles along the wiki. Adding a category to an article creates a link that permits easy navigation from this page to pages in that category.

Operations. Table 1 summarizes the *WikiWhirl* refactoring operations. There are two main reasons to include an operation. First, the operation is natively supported by most wiki engines (e.g., *create*, *categorize*). Second, the operation as such is not supported but documented as part of *Wikipedia*'s good practices (e.g., *merge*, *split*)[5]. The table shows the expected frequency and productivity gains. The latter is measure as the number of clicks required to conduct the operation through *MediaWiki*. For example, *drop* an article involves one click on the delete button and one click on the confirmation button.

However, metamodels only capture the structure but not the operational semantics of these operations, i.e., the impact of the operation on the *Wiki model*. This is important not only for the user to know what to expect about the enactment of these operations but also to assess the abstraction effort made by these operations. Such semantics is scattered and textually described in different

[4] Only first level sections are considered ('== *sectionName* ==' in *MediaWiki*).
[5] `http://en.wikipedia.org/wiki/Wikipedia_guidelines` (accessed 19-Mar-12).

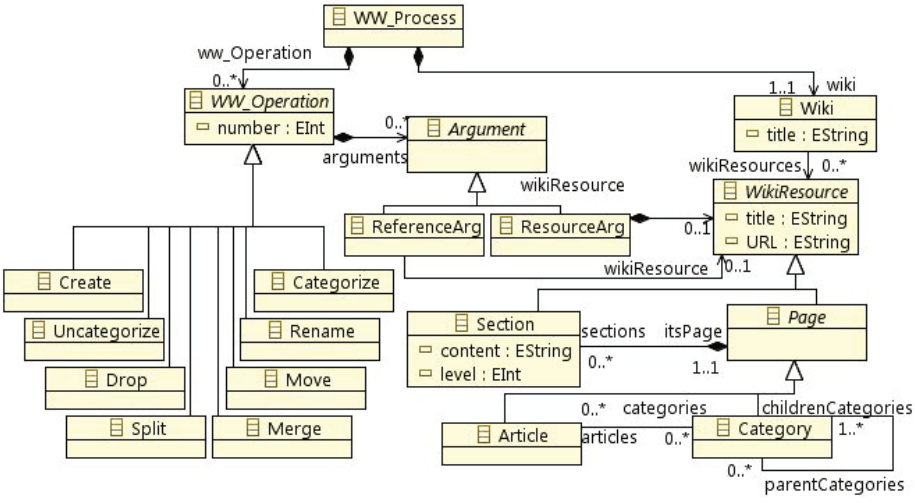

Fig. 5. *WikiWhirl* metamodel (abstract syntax)

Wikipedia recommendations. In addition, we should indicate also the impact on authorship and readership independence. This requires outlining other ancillary wiki artefacts that, although removed from the refactoring abstraction, they do play an important role in ensuring the independence principles that refactoring operations should obey. That is, these artefacts are transparent for the user during refactoring, but they need to be considered during the implementation of the refactoring operations. These artefacts include:

- *Talk pages* (a.k.a. discussion pages) hold discussions about the content of the associated page without interfering with content editing. Talk pages might be used to publicize refactoring changes on the associated articles.
- *Redirect pages* send the reader to another page. These can handle alternative syntactic representations of the same concept whereby no matter the reference used they are all redirected to the same page. During refactoring, articles' content can be moved to different places so the original article vanishes. Redirection avoids dangling references to the removed article so that existing references are dynamically redirected to the new location.
- *Recent changes* page keeps a trace of the most recent edits made to the wiki. Using this page, users can monitor and review the work of other users, allowing mistakes correction and vandalism elimination. This page can also be used to trace refactoring changes so that the rest of the community is informed about who, when and how conducts the refactoring. A brief explanation, called edit summary, can be added to each edit.

Next subsections focus on *splitting* and *merging*.

Table 1. *WikiWhirl* operations: expected frequency & #clicks in *MediaWiki* & impact on authorship and readership independence

WikiWhirl **refactoring operation**	**Frequency**[6]	**# clicks**	**Authorship independence**		**Readership independence**	
			Recent changes	Talk page	Summary section	Redirect page
Create	High	3	✔			
Categorize	High	2	✔			
Uncategorize	Medium	2	✔			
Rename[7]	Medium	2	✔	✔		✔
Drop	Medium	2	✔			
Split	Low	6	✔	✔	✔	
Merge	Low	9	✔	✔	✔	✔
Move	Medium	5	✔	✔	✔	✔

4.1 Article Split (sourceArticle, newArticle)

Article split is a refactoring process documented by *Wikipedia*[8]. The two main reasons for article split are size and content relevance. If either the article becomes too large or its content seems to diverge between different purposes, then a split may be considered. The *newArticle* is categorized along the lines of the *sourceArticle*.

Authorship Independence. It is a requirement of *Wikipedia*'s licensing that attribution is given to the original author(s), and deletion of that content should be avoided. We follow this recommendation by (*i*) preventing content deletion and (*ii*), introducing in the associated talk pages a note such as *"Section sectionName from sourceArticle was copied into newArticle at timestamp"*. In addition, the *recentChanges* page of the *newArticle* is to include a summary, noting the origin of this article (e.g., *"split content from [[article name]]"*). Likewise, the *recentChanges* page of the *sourceArticle* should also include another note indicating that it has been subject to split (e.g., *"split content to [[article name]]"*). This permits users to follow the content trail and to protect against the article subsequently being deleted and the history of the new page eradicated.

Readership Independence. If a section is split from the *sourceArticle*, a summary section should be left in this article. At the top of the section, it should contain a link to the newly created page. Category split is similar to article split. The difference stems from children (articles or categories) of the *sourceCategory*

[6] `http://en.wikipedia.org/wiki/Wikipedia:Statistics`

[7] Category rename is not possible in MediaWiki. (i) Create a new category, (ii) copy the content, (iii) change the category tag manually on every page that link to that category, and (iv) redirect the old to the new category. This is automatic in WikiWhirl.

[8] `http://en.wikipedia.org/wiki/Wikipedia:Splitting` (accessed 19-Mar-12).

might need to be re-allocated into the *newCategory*. That is, some articles might undertake a *categorize* operation.

4.2 Article Merge (articleToMerge1, articleToMerge2)

Article split is a refactoring process documented by *Wikipedia*[9].Reasons to merge an article include the unnecessary duplication of content, significant overlap with the topic of another page, and minimal content that could be covered in or requires the context of a page on a broader topic. This operation creates a new article whose content is that of the merged articles, and its title results from concatenating *"articleToMerge1_ articleToMerge2"*.

Authorship Independence. A comment in the edit summary must be made in the *articlesToMerge* as to where they have being merged to, and must be noted in the *targetArticle*'s edit summary where the content from other pages are being merged from. For our sample scenario, Fig. 6 shows the outcome of merging *LocalStorage* and *StructuralEvents* into *Realization*.

Readership Independence. Merging should always leave a *redirect* from the *mergedArticles* to the *targetArticle*.

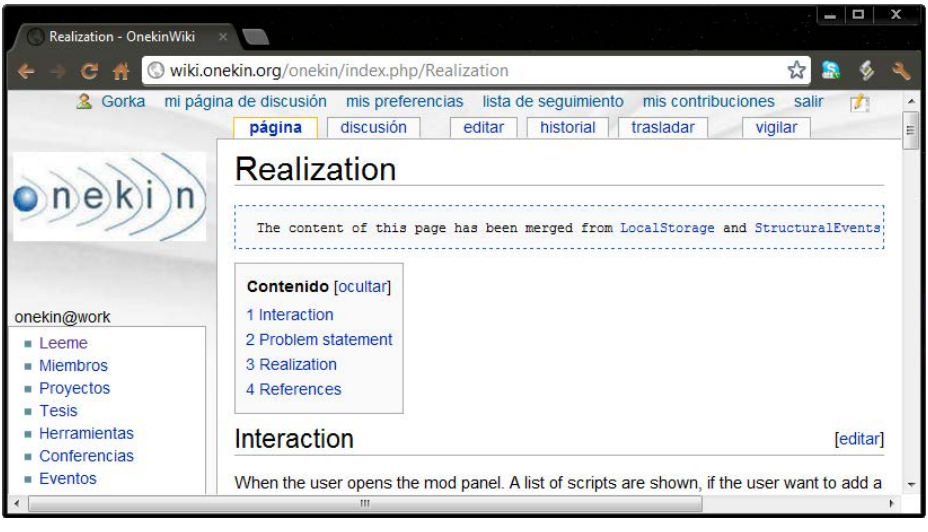

Fig. 6. *"Realization"* article after merging. The warning at the top is automatically generated as part of the merging semantics.

5 WikiWhirl's Concrete Syntax

Broadly, the concrete syntax is a mapping between the metamodel concepts (i.e., abstract syntax) and their textual or visual representation. While the abstract

[9] `http://en.wikipedia.org/wiki/Wikipedia:Merging` (accessed 19-Mar-12).

syntax addresses expressiveness, the concrete syntax cares for the DSL usability and learnability. Should these goals be better served by a textual or a graphical representation? If graphical, which kind of notation? As highlighted by Lengler and Eppler [10], there might not be only one appropriate visualization for a body of knowledge. The selection of the visualization depends on the requirements to meet: visualize author collaboration [20], editions number, network density, page connections[10], single page relationships (e.g., *Annoki* [17]), wiki structure (e.g. *WikiNavMap* [18]), etc. In addition, the expressiveness of these representations for the matter at hand should be balanced against learnability.

Therefore, we need to look into not only the characteristics of the object to be displayed (e.g., size) but also how this object is to be manipulated. The object to be displayed is a graph, where nodes stand for pages while edges denote relationships between these pages. Specifically, we focus on corporate wikis which have an estimate of up to 1500 nodes [16]. The second criterion is manipulation, i.e., the process of refactoring a wiki. Two approaches co-exist. In the bottom-up approach, the user knows the subject of refactoring (i.e., you know which article/category needs to be refactored), and next, a larger view might be required to set this subject into a larger context. By contrast, the top-down approach starts with a global view of the wiki, and next, the user looks for "bad smells" (e.g., too deep category hierarchies with few articles may indicate too much structure [12]). This way of working calls for agile visualizations that permit to define "views" over existing wiki graphs as well as collapse or extend these views as we gain understanding about the refactoring needed.

A main premise of this work is that refactoring is a process of gradual understanding of the wiki structure. Even if you detect "bad smells", the strategy to follow will in most cases, require some "refactoring reasoning" before taking the decision. Indeed, it not clear whether "refactoring recipes" (i.e., patterns) like those available for software would make sense for wikis. Wiki articles are not code but knowledge, better said, "knowledge drafts". It is this *embryonic-like* nature of wiki content together with its *organic growth* what suggests the parallelism with mind maps. According to its proponents, mind maps mimic the way the brain works: "Your brain does not think linearly or sequentially like a computer; it thinks multilaterally, radiantly. When you create a mind map the branches grow outwards from the central image to form another level or sub-branches, encouraging you to create more ideas out of each thought you add" [3].

Drawing on these insights, we select mind maps as the visual representation for wiki refactoring. Notice that this requires moving from graphs to a tree-like representation. Rather than developing our own visual representation, we capitalize on an existing one: *FreeMind*[11]. With over 6,000 daily downloads, it is one of the most popular mind mapping tools. Fig. 1 shows a *FreeMind* map that stands for a wiki. The main advantage rests on the easiness to play around as you gain understanding about the wiki (e.g., nodes are easily moved

[10] `http://sonivis.org` (accessed 19-Mar-12).
[11] `http://freemind.sourceforge.net` (accessed 19-Mar-12).

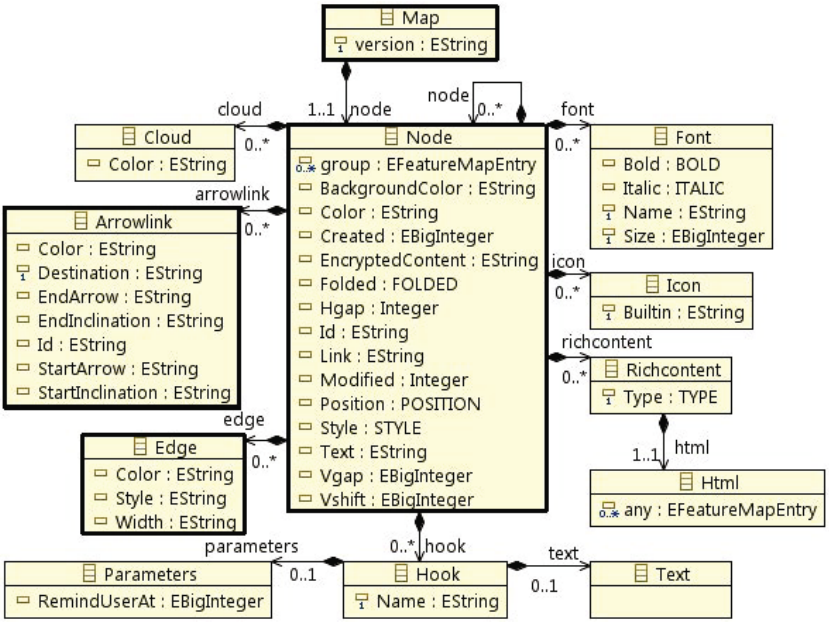

Fig. 7. *FreeMind* metamodel

around; branches collapsed, etc.). Betting for an existing tool not only speeds up development but learnability as well on the hope that the target audience (i.e., employees in corporate wikis) may be familiar with *FreeMind*. The rest of this section introduces *WikiWhirl* as a visual language on top of *FreeMind,* i.e., how the wiki model and its operations are realized in *FreeMind*. This still leaves open two questions: (*i*) how to *import* the wiki into *FreeMind*, and (*ii*) how to *export* the map changes back to the wiki. This is the topic of Section 5.3.

5.1 Mapping Wiki Models to Mind Maps

A *WikiWhirl* model is to be realized as a compliant *FreeMind* map. A mapping is then set between the *WikiWhirl*'s metamodel (Fig. 5) and their visual counterparts, i.e., the *FreeMind* metamodel. The latter is depicted in Fig. 7. A **Map** is a compound of **Nodes**. Nodes have *Text* as the title and might hold a *Link* to an external resource (local or remote) as well as a set of properties mainly for rendering concerns. For instance, the *Style* property can be *fork* or *bubble* and determines the look of the node as a tagged line or a bubble, respectively. Next, nodes are basically arranged in a tree-like way. A central node serves as the tree root. Tree structures are constructed using **Edges**. An Edge is a connector that relates a node with its parent. Additionally, **Arrowlinks** are also connectors

Table 2. *WikiWhirl-to-FreeMind* mapping

WikiWhirl primitives	**FreeMind** primitives
"Wiki" class	Root node
"WikiResource" class (see subclasses)	Node
"title" property	node title: Text
"URL" property	Link
"Category" class first *"parentCategories"* reference rest "parentCategories" references	Node with *"folder"* icon edge arrow link
"Article" class first *"categories"* reference rest *"categories"* reference	Node with *"edit"* icon edge arrow link
"Section" class	Node with *"info"* icon
"itsPage" reference	edge

but in this case, the connection is between two arbitrary nodes. Finally, **Icons**[12] and **Fonts** can be associated with nodes in an attempt to reflect the underlying semantics of the node. Of course, this semantics resides in the users' head.

Next, we define the *WikiWhirl-to-FreeMind* correspondence (see Table 2). *WikiResources* are mapped as *nodes*[13]. *WikiResources* are classified as categories, articles and sections. This categorization is represented through *FreeMind* icons: the *"folder"* icon, the *"edit"* icon and the *"info"* icon denotes category, article and section nodes, respectively. Next, *WikiWhirl* relationships. Since some relationships are M:N, the first relationship instance is represented through an *edge* (the links that support the tree-like structure) while the rest of the relationship instances are denoted through *arrowLinks*. Fig. 1 illustrates this situation. Article "*Architecture of Participation*" belongs to two categories; the relationship with "*Crowdsourcing*" is denoted by an *edge*, while that of "*08Tagmas*" is depicted as an *arrowLink*.

5.2 Mapping Refactoring Operations to Manipulations on Mind Maps

FreeMind is now turned into a refactoring tool for wikis. This basically entails two aspects. First, traditional actions in *FreeMind* are re-interpreted in terms of the refactoring operations (e.g., "node removal" becomes "article drop"). Second, we extend *FreeMind* to cater for refactoring specifics: aspects that do not rise

[12] *FreeMind* provides a fixed set of icons. In the last version, users can introduce their own icons, although it is not recommended for interoperability reasons.

[13] Interesting enough, this representation provides a "site map" of the wiki, i.e., the map accounts for a global view of the wiki where users may navigate to any page just by clicking on a node. *FreeMind* mind maps can be inlayed in *MediaWiki* using the *FreeMind* extension `www.mediawiki.org/wiki/Extension:FreeMind`

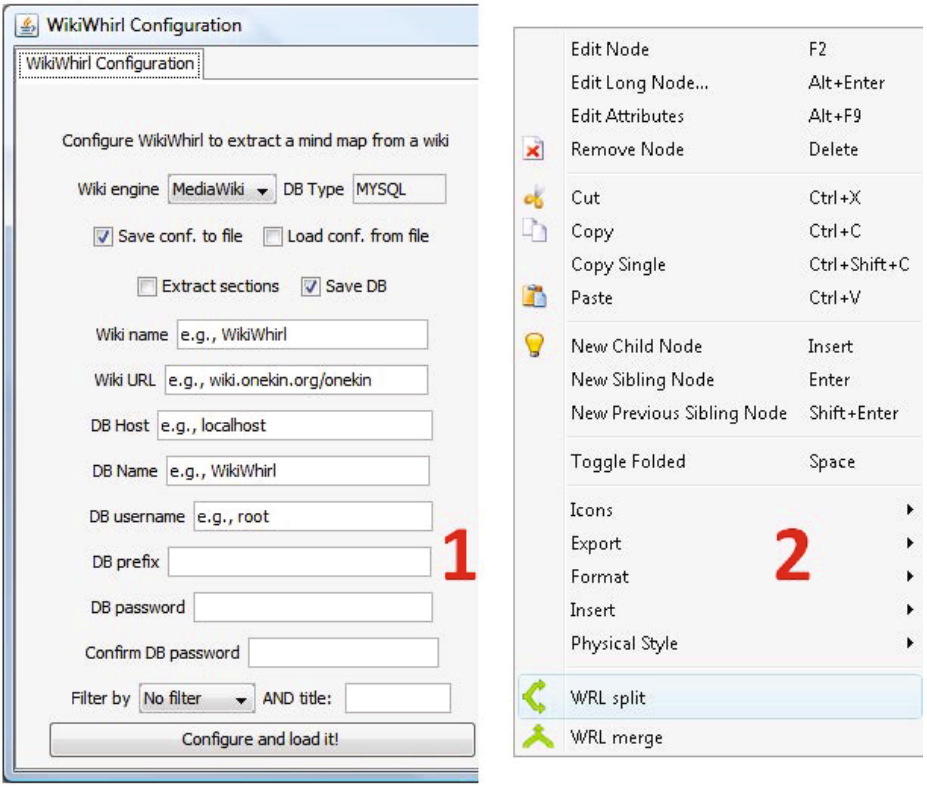

Fig. 8. Turning *FreeMind* into a wiki refactoring tool. (*i*) The configuration menu, and (*ii*) the popup menu (right mouse click).

during the handling of traditional mind maps. This implies (see Fig. 8): (*i*) a way to *import mind maps* from wikis. This is achieved through the configuration menu. This menu sets the configuration service and the load mode. The former indicates the wiki database configuration parameters. The load mode permits to filter sections out of the refactoring session for the sake of efficiency; (*ii*) *inhibiting the removal of nodes* that stand for sections. Sections can only be moved, but never removed since they are content containers (the wiki invariant). Likewise, article/categories can only be deleted if they contain no section**;** (*iii*) introducing "*merge*" and "*split*" as operations on nodes. For this purpose, the right-mouse-click menu is extended with these two options; and (*iv*)**,** introducing a *tracking window* to trace what the user is graphically doing (Fig. 1). The trace is described in terms of *WikiWhirl* operations. The window offers three buttons: the "*Delete*" button removes the highlighted operation from the trace; the "*Commit*" button enacts the refactoring process, enduring the changes in the wiki database, and the "*Refresh wiki mind map*" button restores an initial database dump and regenerates the mind map.

5.3 Importing and Exporting *WikiWhirl* Expressions

A *WikiWhirl* expression is a wiki model (i.e., a mind map) plus a sequence of operations over that wiki. A trivial expression is that with no operations (i.e., an empty sequence) but just a wiki model. A refactoring session:

- Starts with a trivial expression that is gradually enriched as the user operates over the wiki model. It is not expected users to directly provide this initial expression, i.e., the wiki model. Rather, wiki maps should be harvested from existing wiki databases. That is, we import a trivial *WikiWhirl* expression from the wiki database. For this purpose, we use *Schemol* [7], a transformation language that declaratively specify how class models can be obtained from tables.
- Ends by exporting the "sequence of operations" as obtained from *FreeMind* into an SQL script. This process is threefold. First, the refactoring trace is mapped to a *WikiWhirl* model using the EMF persistence framework[14]. Second, the *WikiWhirl* model is transformed to a SQL script. This model-to-text transformation is realized through *MOFScript*[15]. Finally, the *SQL* script is executed; which updates the *MediaWiki* database, i.e., the wiki project.

6 Related Work

This work is related with wiki visualization and management. As for the former, a recent survey [11] highlights that wikis usually exhibit poor structure. On account of this, distinct tools try to improve the user experience by providing improved textual or graphical representations, or even combinations of both. Hirsch et al. [9] define a *Visual Wiki* as a combination of a visual and a textual representation of a wiki. Two of the proposed prototypes, *Thinkbase* and *Thinkpedia*, represent two wikis: *FreeBase* (a semantic wiki) and *Wikipedia*. The main differences with our work are: (*i*) they represent the semantic relationships and not the wiki structure (i.e., hierarchy of categories and articles), and (*ii*) their aim is the visual navigation and exploration whereas our is wiki refactoring.

As for wiki management, in a previous work [5], we also resorted to *FreeMind* as a convenient means to engage users for wiki scaffolding. The aim was to depict a blueprint of the wiki as a mind map, and next, generate the wiki installation. Now the problem is the other way around. *WikiWhirl* first extracts the wiki structure from the wiki database; next depicts the mind map counterpart, which can then be modified by the users; finally, the resulting mind map is transformed into a set of refactoring *MediaWiki* directives. Only this last step keeps some resemblance with our previous work on wiki scaffolding.

As far as automating repetitive tasks, bots[16] are the most common mechanism. Wikis have many routinely tasks to be done and many of them are

[14] `http://eclipse.org/modeling/emf/` (accessed 19-Mar-12).
[15] `http://eclipse.org/gmt/mofscript/` (accessed 19-Mar-12).
[16] `http://en.wikipedia.org/wiki/Wikipedia:Bots` (accessed 19-Mar-12).

too cumbersome to be performed by users (e.g., mass edits or check copyright violations). Bots are tools that take care of articles maintenance. However, other cases are more dubious, and automatic correction is inappropriate. An assisted aproach seems to be more appropriate when it needs the supervision of a human eye. This is the rationale behind [6], where ballots are used to detect inconsistencies and inform the users.

Rosenfeld et al. [15] propose a strategy for semantic wiki evolution based on software refactoring. They identify "bad smells" and the refactoring pattern counterparts. They introduce six *semantic* refactoring operations (e.g., move annotation: change the subject of an annotation to another) and four bad smells (e.g., concept too categorized: it belongs to many categories). The differences with our work stem from (*i*) the focus (semantic resources *vs.* wiki structure), and (*ii*) the approach (template-based description *vs.* graphical DSL).

7 Conclusions

We aim to pave the road for an assisted refactoring for wikis. So far, wiki refactoring is a cumbersome, lengthy activity whose *modus operandi* does not match the accessible and friendly way of editing wiki articles. Such difficulty puts the layman off. But, it is the layman (no *tech-savvy* people) who writes the article, knows the wiki content, and detects refactoring opportunities. This paper strives to abstract from low-level wiki interactions to domain-specific constructs that permit easily and reliably express refactoring processes to the layman. This vision is realized into *WikiWhirl*, a DSL built on top of *FreeMind*: (*i*) wikis are imported as mind maps, (*ii*) users perform refactoring operations as re-arrangements of mind map nodes, and (*iii*) this re-arrangement denote refactoring operations that can be saved into the wiki database while preserving authorship and readership. Future work includes to collect empirical evidence about the usefulness of *WikiWhirl*, and to extend *WikiWhirl* to other wiki engines.

Acknowledgments. We wish to thank Javier Luis Cánovas Izquierdo of AtlanMod Research group, for his support with *Schemol*. This work is co-supported by the Spanish Ministry of Education, and the European Social Fund under contract TIN2011-23839 *(Scriptongue)*. Puente has a doctoral grant from the Spanish Ministry of Science & Education.

References

1. Ambler, S.W., Sadalage, P.J.: Refactoring Databases: Evolutionary Database Design. Addison Wesley Professional (2006)
2. Arazy, O., Stroulia, E., Ruecker, S., Arias, C., Fiorentino, C., Ganev, V., Yau, T.: Recognizing Contributions in Wikis: Authorship Categories, Algorithms, and Visualizations. Journal of the American Society for Information Science and Technology (JASIST), 1166–1179 (2010)

3. Buzan, T., Griffiths, C.: Mind Maps for Business. BBC active (2010)
4. Cunningham, W.: Design Principles of Wiki: How can so Little do so Much? In: Int. Sym. Wikis (WikiSym), pp. 13–14 (2006)
5. Díaz, O., Puente, G.: A DSL for Corporate Wiki Initialization. In: Mouratidis, H., Rolland, C. (eds.) CAiSE 2011. LNCS, vol. 6741, pp. 237–251. Springer, Heidelberg (2011)
6. Díaz, O., Puente, G., Arellano, C.: Wiki Refactoring: an Assisted Approach based on Ballots. In: Proceedings of the 7th International Symposium on Wikis and Open Collaboration, WikiSym 2011, pp. 195–196. ACM (2011)
7. Díaz, O., Puente, G., Izquierdo, J.L.C., Molina, J.G.: Harvesting Models from Web 2.0 Databases. Software and Systems Modeling, 1–20 (2011), doi:10.1007/s10270-011-0194-z
8. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston (1999)
9. Hirsch, C., Hosking, J.G., Grundy, J.C., Chaffe, T.: ThinkFree: Using a Visual Wiki for IT Knowledge Management in a Tertiary Institution. In: Int. Sym. Wikis, WikiSym (2010)
10. Lengler, R., Eppler, M.J.: Towards a Periodic Table of Visualization Methods for Management. In: Conference on Graphics and Visualization in Engineering, pp. 1–6 (2007)
11. Lykourentzou, I., Dagka, F., Papadaki, K., Lepouras, G., Vassilakis, C.: Wikis in Enterprise Settings: a Survey. In: Enterprise Information Systems, pp. 1–53 (2011)
12. Mader, S.: Wikipatterns: A Practical Guide to Improving Productivity and Collaboration in your Organization. John Wiley & Sons Inc., Wiley (2008)
13. Mernik, M., Heering, J., Sloane, A.M.: When and How to Develop Domain-Specific Languages. ACM Computing Surveys 37(4), 316–344 (2005)
14. Raman, M.: Wiki Technology as A "Free" Collaborative Tool within an Organizational Setting. IS Management 23, 59–66 (2006)
15. Rosenfeld, M., Fernández, A., Díaz, A.: Semantic Wiki Refactoring. A Strategy to Assist Semantic Wiki Evolution. In: 5th Workshop on Semantic Wikis Linking Data and People, SemWiki 2010 (2010)
16. Stein, K., Blaschke, S.: Corporate Wikis: A Comparative Analysis of Structures and Dynamics. In: Wissensmanagement, pp. 77–86 (2009)
17. Tansey, B., Stroulia, E.: Annoki: A MediaWiki-based Collaboration Platform. In: Int. Conf. on Software Engineering, ICSE (2010)
18. Ullman, A.J., Kay, J.: WikiNavMap: a Visualisation to Supplement Team-based Wikis. In: Human Factors in Computing Systems (CHI), pp. 2711–2716 (2007)
19. Vallecillo, A.: On the Combination of Domain Specific Modeling Languages. In: Kühne, T., Selic, B., Gervais, M.-P., Terrier, F. (eds.) ECMFA 2010. LNCS, vol. 6138, pp. 305–320. Springer, Heidelberg (2010)
20. Viégas, F.B., Wattenberg, M., Dave, K.: Studying Cooperation and Conflict between Authors with History Flow Visualizations. In: Conference on Human Factors in Computing Systems (CHI), pp. 575–582 (2004)