Improving Creation, Maintenance and Contribution in Wikis with Domain Specific Languages

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)

Gorka Puente García

Supervisor: *Prof. Dr. Oscar Díaz García*
San Sebastián, Spain, 2012

This work was hosted by the *University of the Basque Country* (Faculty of Computer Sciences). The author enjoyed a doctoral grant under the **FPI** (***F**ormación de **P**ersonal **I**nvestigador*) doctoral program from the *Spanish Ministry of Science and Education* (Ministerio de Educación y Ciencia) during the years 2009 to 2012. The work was co-supported by the *Spanish Ministry of Science and Education*, and the *European Social Fund* under Contract TIN2011-23839 (Scriptongue) and TIN2008-06507-C02-01 (MODELINE).

To my parents, Atxen and Javi,
for their wholehearted support and patience,
to my brother, Jon,
Semper Fidelis,
to my fiancée, Esme,
for being you

"If you think you are worth what you know, you are very wrong. Your knowledge today does not have much value beyond a couple of years. Your value is what you can learn and how easily you can adapt to the changes this profession brings so often."

– Jose M. Aguilar

Summary

The radical simplicity and versatility of wikis have encouraged individuals and groups to rapidly embrace this technology. Wikis produce impressive results with minimal resources for knowledge formation and sharing. However, wikis' main characteristics have to be reconsidered when wikis are integrated in an existing organization as opposed to an open wiki (e.g., *Wikipedia*), due to the challenges this entails: (*i*) during wiki inception, wikis have to be tuned to the existing information ecosystem (e.g., documentation, organigram, milestones, etc.); (*ii*) wikis' organic growth results in huge structures of pages which constantly need manual restructuring and maintenance; and (*iii*) corporate users may demand a personal and protected setting before the exposition of their personal knowledge to the public scrutiny.

This Thesis presents an approach, uses cases and a prototype to the aforementioned challenges: (*i*) it advocates the use of a "Wiki Scaffolding", i.e., a wiki installation that is provided at the onset to mimic the organizational setting; (*ii*) it facilitates "Wiki Refactoring" by mind maps manipulation; and (*iii*) it proposes a mechanism for wiki augmentation as a way for users to locally supplement their own content. The proposed approaches rely on Domain Specific Languages (DSLs) to overcome the challenges, specifically *WSL*, *WikiWhirl*, *WikiLayer* and *ScheMol*.

Contents

List of Figures

Chapter 1

Introduction

"Happiness is not something ready made. It comes from your own actions."

– *Dalai Lama.*

1.1 Overview

The radical simplicity and versatility of wikis have encouraged individuals and groups to rapidly embrace this technology, producing impressive results with minimal resources [Lam04]. Along these lines, not just individuals but also companies are increasingly realizing the benefits of wikis [Car07]. However, the very qualities that make wikis so successful as a grassroots-oriented pilot can also create corresponding challenges as the project grows [Lam04].

Due to the nature of the hosting organization, let this be an open community (e.g., *Wikipedia*), a learning organization [TMC08] or a company [LB10], there are differences when it comes to integrating a wiki in the organization. These differences directly impact on wiki success, and mechanisms are needed to facilitate the alignment of the wiki with organizational practices, promote management engagement, enhance the visibility of the wiki's practices, or encourage employee participation. This thesis addresses some of the challenges encountered during the integration of wikis in organizations.

1.2 Context

21st century organizations need mechanisms to cope with the fast business environment, where information and knowledge are the key factors in their never-ending competitive race. In this setting, wikis are becoming commonplace for knowledge formation and sharing [Ram06]. Indeed, wikis are being globally adopted within companies' intranets. The Social Intranet Study [War11] surveys the use of Web2.0 tools (i.e., forums, blogs, wikis, instant messaging, RSS, tagging, etc.) in companies. This study shows that 61% of the respondent companies (1,401 participants) were using wikis. In fact, in a Forrester report, researchers predict that wikis "will have the greatest impact on workplace collaboration" [YMY^{+}08].

This comes as no surprise since most executives realize that intranet 2.0 tools are delivering value in terms of reducing costs, improving productivity and enhancing the user experience [Fin09]. Even though the benefits go beyond money, and the less tangible benefits are the most valuable (e.g., knowledge retention and management, stakeholder collaboration, etc.), the *return on investment* (ROI) measurement in a corporate setting is a must. A study [Fin09] revealed that the average annual ROI of the respondent companies was $1 million, although answers varied from $0 to $20 million. Specifically, companies such as British Telecom (BT), Intrawest Placemaking or T. Rowe Price report important savings (from hundreds of thousands of dollars to millions) thanks to the use of wikis. Curiously enough, the Social Intranet Study [War11] indicates that around 30% of the companies have only a single person dedicated to managing the intranet, what may explain that most companies, 80%, do not measure the ROI of their intranets.

Nevertheless, the original intention of wikis was to be open wikis (as opposed to corporate wikis), and wikis' main principles conflict with the organizational setting peculiarities. Corporate wikis differ from open wikis in: (*i*) *who* use them. Communities in open wikis are built around the wiki itself, whereas organizations already have employees that will become wiki users. In addition, employees adopt distinct roles within projects, departments, and so forth, making user definition and management harder; (*ii*) *what* they are used for. Open wikis are created for an end (e.g., an encyclopaedia), while corporate wikis represent a means for an end (e.g., knowledge management, support projects or tasks, decision-making, etc.), so that the effort to create and maintain the wiki should be minimal; (*iii*) *how* they are used. Corporate wikis demand complex access control, editing entails a responsibility, deadlines are frequent, and so on, what requires advanced management mechanisms; and, (*iv*) *what* is used in them. When a corporate wiki is to be created, there are existing documents (e.g., spread sheets or word documents, policies, reports, templates, etc.) that must be ready from the very beginning, demanding import mechanisms.

Therefore, the wiki success might not be directly translated to a corporate setting. A tension rises between wikis' affordances (i.e., action possibilities) and the nature knowledge is managed in organizations (e.g., restrictions in access) [YA12]. Specifically, this work delves into three activities: wiki initialization, wiki refactoring and wiki customization.

1.3 General Problem

An Information System (IS) is defined as "a designed system that collects, stores, processes, and distributes information about the state of a domain" with three main functions: (*i*) *memory* to maintain a representation of the state of the domain; (*ii*) *informative* to provide information about the state of the domain; and, (*iii*) *active* to perform actions that change the state of that domain [Oli07]. As any other IS, the interplay of technology, work practice, and organization is paramount to achieve successful wiki deployments. Therefore, wikis' main characteristics have to be reconsidered when wikis are integrated in an existing organization. This dissertation tackles three scenarios, namely:

1. *Wiki initialization*. First of all, the wiki is created and during this stage the peculiarities of each organization will certainly percolate the wiki. Documentation, organigrams, project milestones should all be there by the time the wiki is created. This contrasts with open wikis (e.g., Wikipedia) where the community did not exist prior to the wiki. As a result, corporate wikis need to be tuned at the onset to the already existing information ecosystem.

2. *Wiki refactoring*. Wiki content and structure evolve hand in hand with its supporting community (a.k.a. the wiki's *Organic Principle* [Cun06]). Since people have different mental maps (i.e., perception of the world), they express them in different structures although not always properly. In practice, this ends up in large structures of articles and categories which constantly need manual maintenance. Wiki refactoring has to do with the hassles of keeping the wiki in "good shape".

3. *Wiki customization*. The last scenario comes with the tuning of the wiki to the user specifics. So far, wiki editions are exposed directly to the public scrutiny. Even though this might not be a big burden for anonymous contributors in open wikis, it might cause distress in identified contributions in corporate wikis. Now, the observers are your own colleagues. This advises to introduce a private sphere where users can control the disclosure of their contributions as they gain confidence.

Corporate environments are demanding and if these challenges are overcome, wikis are able to provide a low upfront investment, a reduced maintenance and a successful collaboration framework.

1.4 This Dissertation

This Thesis proposes an approach, use cases and a prototype for each of the aforementioned challenges, for doing so:

1. During wiki inception, this dissertation advocates for the use of a “Wiki Scaffolding” to align wikis with the corporate strategy. A “Wiki Scaffolding” is a wiki installation that is created at the onset to mimic the organizational setting. This approach saves resources (both economical and in manpower) and makes wikis accessible to domain experts.

2. While the wiki grows, the presented approach allows wiki users to conduct wiki refactoring themselves. In this regard, the expression of wiki refactoring is facilitated by the manipulation of mind maps. This approach accounts for an improved global understandability (i.e., correct interpretation of wiki content), productivity (i.e., less time and errors) and refactoring affordance (i.e., a perceived opportunity for action) while authorship and readership are preserved. Moreover, mind maps provide a general perspective that makes the spot of refactoring opportunities easier, accessible to end-users and more intuitive.

3. Wikis may become repositories of personal knowledge without losing their main principles. Users can augment wikis based on their own needs. To this end, web augmentation might offer a backdoor for people to do more personalized exploration; instead of trying to converge on a consensus, augmentation might account for a more personal and protected setting, which can eventually spur contributions.

Following sections describe the issues that need to be addressed.

1.5 Problem Statement for Wiki Initialization

- *What*. “Wiki Scaffolding” stands for a wiki installation (a.k.a. a wiki project) that is available from the wiki’s onset, before any contribution is made. Such installation mirrors the practices of the hosting organization. Some examples follow: (*i*) company schedulings might impact the pace at which wiki articles are provided (e.g., deadlines, project milestones); (*ii*) products, services, customers or established terminology within an organization might become categories to classify wiki articles; (*iii*) employees eligible to contribute, and their access control permissions, might be based on the company’s organigram. A “Wiki Scaffolding” captures this setting as a wiki installation where the basic wiki configuration might be extended (through plugins) based on the selected scaffolding features (e.g., a plugin for events and calendars).

- *Why*. The fact that wikis facilitate knowledge creation does not imply that such knowledge comes out of the blue. Both, the paralysis of facing an empty article and the lack of a holistic view of the wiki content, might prevent grassroot initiatives from “getting off” the ground. At this respect, scaffolding brings three main benefits:

  1. Scaffolding facilitates wikis to be better aligned with the organization strategy. Wikis are frequently a bottom-up phenomenon whereby the wiki is introduced by an individual employee or a small group within the organization and without the support of management. This approach may be useful to uncover hidden knowledge or hidden ways-of-working in a dynamic and unplanned way. However, it might fail in having a strategic intent. A lack of strategy might result in no clear guidelines about what, how and who should contribute. If so, “Wiki Scaffolding” forces to think about these concerns right from the beginning.

2. Scaffolding promotes user engagement. In a corporate setting, a wiki article might require some permissions, be subject to a deadline, belong to some wiki categories, or follow a given template. All these aspects might not be directly related with the article's content as such, yet they frame the contribution. Setting this frame is cumbersome and delays users in putting their wheels in motion (e.g., start to edit the article). "Wiki Scaffolding" permits this frame to be available by the time contributors start their articles.

3. Scaffolding as a wiki map. The "rules of practice" that govern a site (i.e., roles, access rights, templates, etc.) should be easily accessible to newcomers. So far, this information is scattered around the wiki, and frequently hidden in administrative pages. At best, a *README* page can provide some textual description of these practices. "Wiki Scaffolding" can play the role of an initial "practice sitemap". Newcomers can consult the scaffolding to have an eye-bird view of the rules that govern the wiki's operation.

- *How*. "Wiki scaffolding" faces two main obstacles. First, it implies an upfront investment before any content is provided. Second, it requires knowledge about the wiki engine (e.g., *MediaWiki*[1]) and third-party extensions, both outside the competences of the layman. This will make "Wiki Scaffolding" yet another burden for the organization's IT department since most users will lack the required skills. Akin to the wiki spirit, the scaffolding should be managed by the users themselves. Therefore, both cost-effectiveness and end-user affordability are main prerequisites for scaffolding to be adopted. This advocates for the use of Domain-Specific Languages (DSLs) [MHS05]. Furthermore, collaboration and easy sharing can be promoted by using graphical DSLs (as opposed to textual

[1] `www.mediawiki.org` (accessed December 2012).

DSLs). Mind maps are popular diagrams that capture ideas around a central topic [BG10]. This thesis capitalizes from this popularity, and introduces a DSL described as a mind map to both capture and enact "Wiki Scaffoldings". Specifically, it is introduced the *Wiki Scaffolding Language* (*WSL*) (pronounced "whistle"). WSL is built as a plugin of *FreeMind* [Fre], a popular, open source tool to create mind maps. Users create their scaffolding by drawing mind maps. Next, users can "export" their mind map as a "Wiki Scaffolding": a new wiki is created along the lines of the directives of the scaffolding (see a video of WSL at work at `http://vimeo.com/31548363`). The source code, examples and installation instructions can be found at `http://www.onekin.org/wsl`. Alternatively, WSL source code is also available in the official FreeMind code repository `http://bit.ly/xsA040`.

1.6 Problem Statement for Wiki Refactoring

- *What*. Wikis are becoming a mainstream for knowledge formation and sharing [Ram06]. 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., the wiki content) and its structure evolve with the supporting community. Michel Buffa [BG06] quotes the discussion of wiki creator Ward Cunningham along with *Wikipedia* founder Jimmy Wales, at Wikisym'05 conference, who explained the "wiki way" philosophy:

  "*A wiki is like a garden; users... must take care of it. Start with some seeds and watch it grow, and the wiki will become moderated by its users' community... Do not impose a rigid structure, users will refactor and structure the wiki as it grows...*"

In practice, this ends up in large structures of articles and categories, which constantly need manual refactoring. Despite the early identification of refactoring as part of the wiki's *modus operandi*, most efforts have been directed to facilitate content editing while content refactoring has been largely overlooked. In this regard, "Wiki Refactoring" is the process of changing the wiki's internal structure for the sake of navigability, accessibility or comprehension, but the content (and its authorship) is kept immutable.

- *Why*. The aim of wikis is an *affordable* approach to collaborative knowledge formation and sharing. Wiki refactoring is certainly part of the knowledge formation effort. For "Wiki Refactoring" to be effective, it has to be *affordable*. Hence, this needs first to systematize "Wiki Refactoring" (the "*what*"), and second, to assess "Wiki Refactoring" affordances (i.e., a perceived opportunity for action) for current wiki engines (the "*how*"). In software, refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behaviour [Fow99]. As for wikis, this definition raises two questions: (*i*) how the wiki structure can be altered i.e., the refactoring operations, and (*ii*) what this "external behaviour" is i.e., the invariant to be kept during "Wiki Refactoring". Operations include *categorize* (i.e., characterizing the page content with a category), *split* (i.e., dividing the page content into two pages), *merge* (i.e., joining two pages into a single one), etc. As for the invariants, good practices advise to preserve wiki content and authorship[2]. Wiki refactoring can change the wiki internal structure for the sake of navigability, accessibility or comprehension, but the content and its authorship should be kept immutable.

[2] `http://en.wikipedia.org/wiki/Wikipedia_guidelines` (accessed December 2012).

Unfortunately, these operations and invariants are frequently only in the mind of the user with little support from the wiki engines (e.g., MediaWiki). Wiki engines are primarily thought for spurring editions but overlook refactoring. As a result, "Wiki Refactoring" might become convoluted. For instance, merging/splitting two wiki articles requires of at least five interactions in MediaWiki (the most common wiki engine). 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: 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 can be thought as database transactions in the sense that they might 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 lives on the minds of the wiki users who need to delve into the intricacies of the wiki engine.

- *How*. Those that know *what to refactor* (e.g., the knowledge workers that know which articles to merge) might ignore *how to refactor* (e.g., the operative that goes to properly merge two articles). As pointed out in [HF12] when talking about the wiki openness

  > “*this decentralized mode of governance is what has made it possible for the all-volunteer wikipedian community to collectively build and maintain the project. However, this mode is often impenetrable for new editors who lack the organizational literacies required to interpret and author texts and traces*.”

  While cosmetic editing might require minimal familiarization with the domain at hand, structural refactoring expects deep knowledge about the wiki corpus and refactoring principles. This work proposes to improve refactoring affordances of knowledge workers (i.e., wiki contributors) by lowering the participation barrier through the introduction of a DSL. However, refactoring changes in the wiki structure impact in readers and authors. Hence, we introduce two dimensions of independence for wikis: (*i*) *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. And (*ii*) *authorship independence*. Acknowledging the authorship has been reported as a main motivator of contributions [ASR$^+$10]. Wiki refactoring must preserve authorship. The output is the *WikiWhirl* DSL, which supports wiki refactoring by the rearranging of mind maps nodes. WikiWhirl preserves the two dimensions of wiki independence, readership and authorship. Results from a controlled experiment suggest that WikiWhirl outperforms traditional wiki front-ends in three main affordance enablers: global understandability, productivity and automatic compliance of refactoring good practices. WikiWhirl is available to download from Onekin[3], and its source code from Sourceforge[4].

[3] `www.onekin.org/wikiwhirl` (accessed December 2012).

[4] `http://sourceforge.net/p/wikiwhirl/code` (accessed December 2012).

1.7 Problem Statement for Wiki Customization

- *What*. 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. [McF05, Fil06]. A popular example is the *Skype* add-on [Sky05], 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* [AVG10], an augmentation utility provided by *AVG* (a popular anti-virus) that permits to scan search results from Google, Yahoo! or Bing, and places a security rating next to each recovered link, which informs about the trustworthiness of the site.

  For a given website, Web Augmentation brings a kind of externalized customization: users can tune the front-end of a website based on their own needs. Therefore, wiki customization is the process whereby users *locally* supplement wiki pages with their own content, content obtained from other wikis or content obtained from other websites. This process should be as easy as it is currently editing wikis.

- *Why*. Wiki users could add local content to augment the raw content of an 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, wikis 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. Wikis' freedom of editing does not imply all editions becoming publicly available: some editions 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, wiki customization 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 a wiki and hence, exposed to public scrutiny, might be too intimidating. Here, wiki customization 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*. Wiki Customization should be tuned to the "wiki way". Wikis are characterized as being open (i.e., edition is easily conducted without even to 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 (i.e., changes are tracked and visible to the rest of the community) [Cun02]. Likewise, augmenting wikis should then be (*i*) *affordable* as the counterpart of open, i.e., the complexity should be similar to that of writing a piece of wikitext, (*ii*) *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 (*iii*), *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 wikis rendering with augmentation capabilities.

  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., *Twitter* followers) who are only a click-away from locally installing their own copy of the layer (i.e., *shareable*). WikiLayer has been contextualized for

Wikipedia and can be found at `http://webaugmentation.org/wikilayer.xpi` with samples[5].

Next section summarizes the main contributions of this Thesis.

1.8 Contributions

Wiki Initialization

Firstly, we enumerate the specificities of moving to a corporate setting. Then, this work introduces the notion of "Wiki Scaffolding" as a way to create an initial blueprint for wikis. Finally, we tap into mind maps to come up with a DSL to capture "Wiki Scaffoldings".

Wiki Refactoring

This work collects and identifies the main operators in wiki refactoring to give shape to a new DSL. In so doing, this DSL for "Wiki Refactoring" preserves authorship and readership, while allows layman users perform the refactoring operations by themselves in an easy and reliable manner.

Wiki Customization

In this approach, web augmentation emerges as the main enabler to wiki customization. To this end, a tool blends the social knowledge with the personal knowledge, introducing a personal perspective in wikis. The approach is designed along wiki concepts to mimic the wiki way.

1.9 Design-Science as Research Approach

The design-science is a paradigm where knowledge and understanding of a problem domain and its solution are achieved in the building and

[5] `http://tinyurl.com/wikilayersamples` (accessed December 2012).

Table 1.1: Design-Science Research guidelines as indicated in [HMPR04].

GUIDELINE	DESCRIPTION
Guideline 1: Design as an artefact	Design-science research must produce a viable artefact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design evaluation	The utility, quality, and efficacy of a design artefact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artefact, design foundations, and/or design methodologies.
Guideline 5: Research rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artefact.
Guideline 6: Design as a search process	The search for an effective artefact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

application of a designed artefact, whereby the boundaries of human and organizational capabilities are extended [HMPR04]. Thus, the design-science research cycle creates and evaluates artefacts intended to solve identified organizational problems. Hevner et al. [HMPR04] propose seven guidelines to conduct and evaluate design-science research in Information Systems (see summary in Table 1.1). This dissertation follows the design-science research approach along those seven guidelines to ensure the relevance and effectiveness of the research. In so doing, the contributions are put in the design-science context:

- *Problem relevance*. The Section 1.3 (Chapter 1) indicates *what* is the general problem and *why* it is important, whereas the Sections 1.5, 1.6 and 1.7, describe the problem statement for Wiki Initialization, Wiki Refactoring and Wiki Customization, respectively.

- *Research contributions.* This introductory Chapter 1, gives a quick overview of the contributions in Section 1.8, whereas in the Conclusions (Chapter 6, Section 6.2), the results can be seen in more detail. For a complete description of the contributions, refer to the main Chapters: Chapter 3 for Wiki Initialization, Chapter 4 for Wiki Refactoring and Chapter 5 for Wiki Customization.

- *Design as an artefact.* The artefacts developed during this research are WSL (Chapter 3), WikiWhirl (Chapter 4), WikiLayer (Chapter 5) and ScheMol (Appendix A).

- *Design evaluation.* WSL expressivity was evaluated by conducting a literature review, and creating different *case studies* extracted from the literature and real examples. In addition, the company HalloWelt![6] is currently trying it to prove its viability in a real setting and will provide feedback. WikiWhirl has been evaluated in a controlled experiment by measuring fulfillment of refactoring good practices, global understandability and productivity (Chapter 4). Finally, Cristobal Arellano is at the moment performing a user evaluation of WikiLayer, to be included in his PhD. Regarding ScheMol, it is being used in a real project with the company ISG[7] , which provides real and valuable feedback for its improvement.

- *Research rigor.* The *artefacts* of this research follow the development phases of a DSL as described by Mernik et al. [MHS05]. Furthermore, the contributions of this thesis are based on the state of the art related to wikis and open collaboration systems.

[6] `http://www.hallowelt.biz` (accessed December 2012).

[7] `http://www.isg4.com` (accessed December 2012).

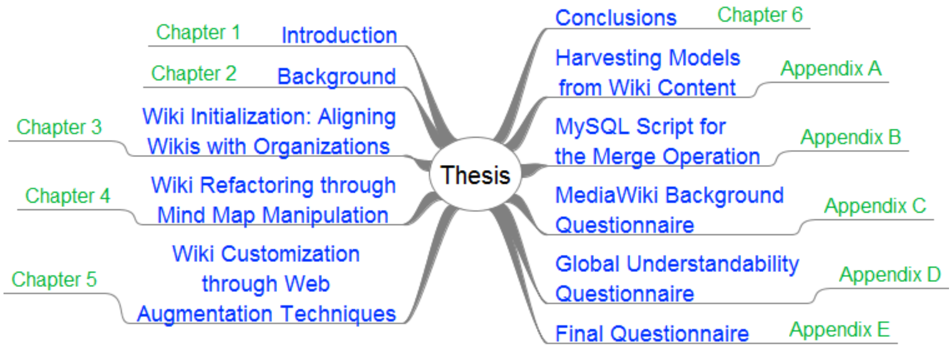

Thesis
Chapter 1
Introduction
Chapter 2
Background
Chapter 3
Wiki Initialization: Aligning Wikis with Organizations
Chapter 4
Wiki Refactoring through Mind Map Manipulation
Chapter 5
Wiki Customization through Web Augmentation Techniques
Conclusions
Chapter 6
Harvesting Models from Wiki Content
Appendix A
MySQL Script for the Merge Operation
Appendix B
MediaWiki Background Questionnaire
Appendix C
Global Understandability Questionnaire
Appendix D
Final Questionnaire
Appendix E

Figure 1.1: Chapter map.

- *Design as a search process*. The premises taken and the suppositions made were based on the state of the art in wikis and open collaboration settings. However, some contributions go further "common wisdom" in wiki design, which would suggest (*i*) an organic design approach begins with little planning but then refactors the wiki when needed to address emerging needs. And, (*ii*) normal wiki method would suggest a common access method for all users to modify the wiki, thus avoiding "walled gardens"[8].

- *Communication of research*. The significance of the findings is demonstrated through publications in refereed conference proceedings and journal articles (Chapter 6).

1.10 Outline

This Thesis consists of six chapters and five appendix. The Figure 1.1 shows the chapter map and the rest of this section gives an overview of each of those chapters and appendices.

[8] Thanks to Christian Wagner for this appreciation.

Chapter 2

This chapter describes the background on wikis and Model Driven Engineering (MDE). These are the main topics on which this dissertation is based on.

Chapter 3

This chapter addresses the following research question: *how can corporate strategies permeate wiki construction while preserving wiki openness and accessibility?*. To this end, the chapter introduces the notion of “Wiki Scaffolding”, and advocates for the use of DSLs as the engineer means. Specifically, we introduce the *Wiki Scaffolding Language* (WSL).

Chapter 4

This chapter poses the following research question: *how to improve the refactoring affordances of current wiki engines*. In so doing, there are contributions to the area of “Wiki Refactoring” by (*i*) formalizing a set of refactoring operations, (*ii*) evidencing the limitation of current approaches from an *affordance* perspective, (*iii*) proposing the use of mind maps as a suitable concrete syntax to express refactoring, and (*iv*) providing WikiWhirl as a proof-of-concept.

Chapter 5

This chapter raises the following research question: *how could wiki users enhance wikis with their personal knowledge or perspective?*. In this regard, this chapter presents WikiLayer as an augmentation tool. WikiLayer provides a lightweight, seamless, client-based approach to supplement existing wiki articles with additional content, potentially brought from other websites (wikis or not).

Chapter 6

This chapter presents the main results, publications, future work and limitations, concluding the Thesis.

Appendix A

This appendix introduces an approach for extracting models out of database schemas. Concretely, by describing *Schemol*, a DSL that considers Web2.0 specifics. Some examples are provided to show the usefulness of Schemol when it comes to extracting models out of wikis.

Appendix B

This appendix shows a SQL script autogenerated by WikiWhirl for the merge operation.

Appendix C

This appendix shows the questionnaire used in the user evaluation of WikiWhirl to assess the MediaWiki knowledge of the participants prior to the experiment.

Appendix D

This appendix presents the questionnaire used to evaluate the global understandability of the participants in the WikiWhirl experiment.

Appendix E

This appendix exposes the final questionnaire used in the WikiWhirl experiment to gather the final results.

1.11 Conclusions

In this chapter we have introduced the main issues, which will be discussed during the presented thesis. Next chapter reviews the basic background that is needed to situate the reader for the following chapters.

Chapter 2

Background

"If I had eight hours to chop down a tree, I'd spend six hours sharpening my axe."

– *Abraham Lincoln*

2.1 Overview

This background chapter explains the basic topics on which this dissertation is based, which lies the grounds for this dissertation, namely *wikis* and *Model Driven Engineering* (*MDE*). The intention is to describe the main ideas behind them, core definitions and brief introductions to prepare the reader for the rest of the Thesis.

Wikis have already demonstrated their value as collaborative tools and knowledge enablers. Model Driven Engineering is an established software engineering approach. Both topics find their way together in this dissertation, a background on both is, therefore, a must.

2.2 Wikis

The first *Wiki*, *WikiWikiWeb*[1], was developed by Ward Cunningham in 1995. It was conceived to share ideas about software development and patterns. Ward Cunningham describes a wiki as "*the simplest online database that could possibly work*" [Cun02]. Its name comes from the Hawaiian word "wiki", which means quick or fast[2]. It refers to the wiki content that is available in a quick and easy way.

Wikis are very popular primarily due to the success of the open encyclopaedia *Wikipedia*. Some people may only be exposed to Wikipedia and this may create a misconception about what a wiki is for. Wikipedia is an *open community* wiki but wikis are also of increasing importance in other domains, namely:

- *E-government*: Wiki government is a young area. Effective governance in the twenty-first century requires collaboration and the public sector is approaching innovation to design a more democratic and collaborative government [Nov09].

- *Education*: Wikis in education are being used to support courses, collaborative learning or to encourage student participation from primary schools to Universities [TPP09, KMC11].

- *Organizations*: Wikis hosted by existing organizations are the so called corporate wikis [MWY06]. Corporate wikis boost collaboration and knowledge sharing among organization members. In addition, they differ from other domain wikis in power relationships, participant goals and content ownership [DS08].

Wikis are defined and motivated in the remainder of this section. Wiki engines, successful case studies and current research issues finish the background for wikis.

[1] `http://c2.com/cgi/wiki?WikiWikiWeb` (accessed December 2012).

[2] `www.mauimapp.com/moolelo/hwnwdshw.htm` (accessed December 2012).

2.2.1 Definition

A wiki is a web-based software that allows all viewers of a page to change the content by editing the page online in a browser [EGHW08]. Wikis are supported by wiki engines and in general, store the wiki content in databases (some wikis store the content in files e.g., *DokuWiki*). Wiki design principles can be summarized as [Cun06]:

- *Simple*: It is easier to use than abuse.
- *Open*: If an incomplete or poorly organized page is found, readers can edit it as they see fit.
- *Incremental*: It is possible and useful to cite pages that have not been written yet.
- *Organic*: It means that wikis grow and shrink dynamically along the desires of the community that uses and natures them.
- *Mundane*: Only a small number of text conventions is needed to provide access to the most useful page markup for formatting.
- *Universal*: A writer is also an editor since the mechanisms of organizing a wiki are the same as those of writing.
- *Overt*: The formatted output will suggest the required input to reproduce it.
- *Precise*: Wiki page titles will be descriptive enough to avoid name clashes.
- *Tolerant*: Inputs will produce outputs even when the output is not the expected. This is preferred to errors.
- *Observable*: Wikis demand change tracking and peer-review mechanisms. Thus, activity within the site can be watched and reviewed by any other visitor (e.g., through a “recent changes” wiki page).

- *Convergent*: The duplication and ambiguity of content should be avoided, and that content related to similar within the wiki.

Wiki main components are the *users* and the *content*. Users are normally divided into groups. In this way, user rights can be assigned to those user groups. Wiki content is stored in wiki pages. A wiki page basically stands for a canvas that permits easy editing. The bases of the wiki content are:

- *Wiki text* is text that uses wiki markup. Wiki markup denotes the special syntax and keywords used by wiki engines to format text.
- *Wiki article* is the core concept of a wiki. It is an entry on the wiki with information written on it. A wiki article usually contains links to other wiki articles.
- *Wiki category* is a keyword or tag 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 other pages in that category and, in so doing, facilitates browsing related articles.
- *Wiki template* is a special page whose content is intended to be included in multiple pages. Its purpose is to offer a structuring mechanism for homogenizing easily updatable content.
- *Wiki talk* pages or discussion pages are a special kind of pages used to hold discussions about the content of the corresponding page. With this mechanism, the content is kept separated from discussion threads.

2.2.2 Motivation

Wikis have been outshone by its most known example: Wikipedia. In 2001 a website called Wikipedia asked people for collaboration to create a free encyclopaedia and now, over ten years later, it has become the world biggest encyclopaedia ever done. Not only that, Wikipedia statistics speak for themselves: sixth most visited website according to Alexa's top sites [3], over 400 million visitors every month, over 17 million articles in 270 languages[4]. Wikipedia represents an act of mass collaboration with over 100,000 people as global volunteers. But above all, it promotes the cause of free knowledge for anyone on the planet. Nevertheless, wikis are more than just Wikipedia as explained next.

Corporate Wikis

As previously mentioned, corporate wikis differ from wikis in other settings (e.g., educational, public-access, etc.), and present challenges when it comes to integrate them in an existing organization [GP10]:

- Companies are hierarchically organized and the common botton-up approach does not naturally fit.
- These wikis usually work behind a firewall, so access may be restricted outside the company.
- Companies require lasting information infrastructures but the introduction of a new technology may be disruptive.
- There many internal factors affecting the original pace at which open wikis work such as stringent constraints, document repositories, complex communication channels or authority figures.

Even so, companies have realized the potential wikis have, and these are being used for different organizational processes, reporting several advantages [LDP$^+$12]:

- *Knowledge codification*: Organizations utilize wikis as a means to collect, codify and maintain corporate knowledge, and at the same time, as a means to distribute this knowledge among the organization members.

[3] `http://www.alexa.com/topsites` (accessed December 2012).

[4] `http://wikimediafoundation.org/wiki/Wikipedia_Celebrates_10_Years_of_Free_Knowledge` (accessed December 2012).

- *Communities of practice*: Wikis present many capabilities for customization and integration with existing technologies, allowing to cater for learning activities within in-house communities of practice. These communities benefit from larger networks of informal knowledge sharing, which has a positive effect on the knowledge and communication platforms of the organization.

- *Interaction with third parties*: Wikis may easily connect organization members, customers, stakeholders, and so on, in different ways: (*i*) help desk wikis provide external support in corporate product or services, being an easy and up-to-date solution; (*ii*) marketing and advertising through wikis reduces e-mail overload and provide a direct engagement; and, (*iii*) enabling the public to participate in news and publications actively involves the customers with the organization.

- *Information Systems (ISs) development and maintenance*: Wikis represent, not only an affordable alternative but also an effective one for different tasks in ISs development and maintenance: IS documentation (e.g., link code to documentation), software reuse (e.g., same artefacts in different projects), requirements engineering and elicitation (e.g., frequent changes need a dynamic mechanism), or end-user programming (e.g., collaborative perspective).

- *Management activities*: In decision making processes, wikis provide managers with a new source of information and an opportunity to assess project risks beforehand. Regarding project management, scheduling, planning and real-time tracking of activities (e.g., *Trac* a wiki-based project management system[5]) are successfully integrated within wikis. Wikis are also valuable tools for capturing and sharing managerial experience.

[5] `http://trac.edgewall.org/` (accessed December 2012).

- *Organizational response in crisis situations*: Wikis can help to overcome time lag among the different geographically spread groups and assign roles among unit members. Wikis may also support cross-unit collaboration and improve knowledge communication among units.

2.2.3 Wiki Engines: MediaWiki

A wiki engine is the core software that runs a wiki. Most implementations are written in *PHP* and require the presence of a database, commonly *MySQL*. However, there are wiki engines written in *Java*, *Python*, *Perl*, *Ruby*, etc., and based on *PostgreSQL*, *Oracle*, *SQLite* or even files instead of a database. There are currently over 130 different wiki engines[6] and the most popular one is *MediaWiki* (based on the number of sites using it and the number of downloads). This wiki engine has been selected in this work for several reasons:

- It is maintained by the *Wikimedia* foundation, which supports about 800 wikis (e.g., *Wikiquote*, *Wikinews*, etc.) and Wikipedia. This makes MediaWiki performance and scalability a major concern, thus it is highly optimized.

- It has a devoted developer community from all around the world, who organizes conference and meetings focused on development. This ensures quick bug fixing and robust releases.

- Users can easily find assistance on the support desk, an official mailing list, books, forums, and so forth.

- Hundreds of extensions are available to enhance the basic features provided by MediaWiki.

[6] `www.wikimatrix.org` (accessed December 2012).

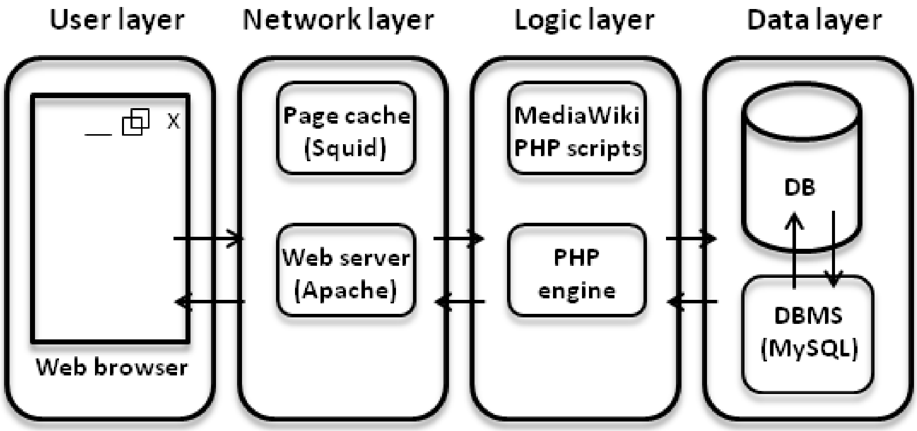


Figure 2.1: MediaWiki four layers architecture.

Figure 2.1 presents the four layered MediaWiki architecture[7]. Users interact with the web browser (i.e., user layer), that fetches wiki pages from the web server (i.e., network layer). The PHP engine (i.e., logic layer) renders the pages dynamically, by first extracting the information from the relational database back-end.

The current database schema (MediaWiki 1.19) contains 53 tables, but, for the purpose of this Thesis, only 12 are described, mostly related with page content and associated information. Figure 2.2 depicts (a partial view of) the MediaWiki database schema: *redirect* table contains a record for each page that is a redirect to other page; *categorylinks* and *pagelinks* tables keep the links among categories and pages, respectively; *category* table tracks all existing categories. In MediaWiki, something is a category if there is a link to it of the type [[Category:NameOfTheCategory]], even if it does not really exist (i.e., a link to a page that does not exist yet is useful to indicate that a page will be created soon or should be created); *recentchanges* table logs data about modifications on the wiki; *page* table holds the description of pages no matter they realize articles, categories, templates, discussions, etc; *logging* table keeps every log action (e.g.,

[7] `www.mediawiki.org/wiki/Manual:MediaWiki_architecture` (accessed December 2012).

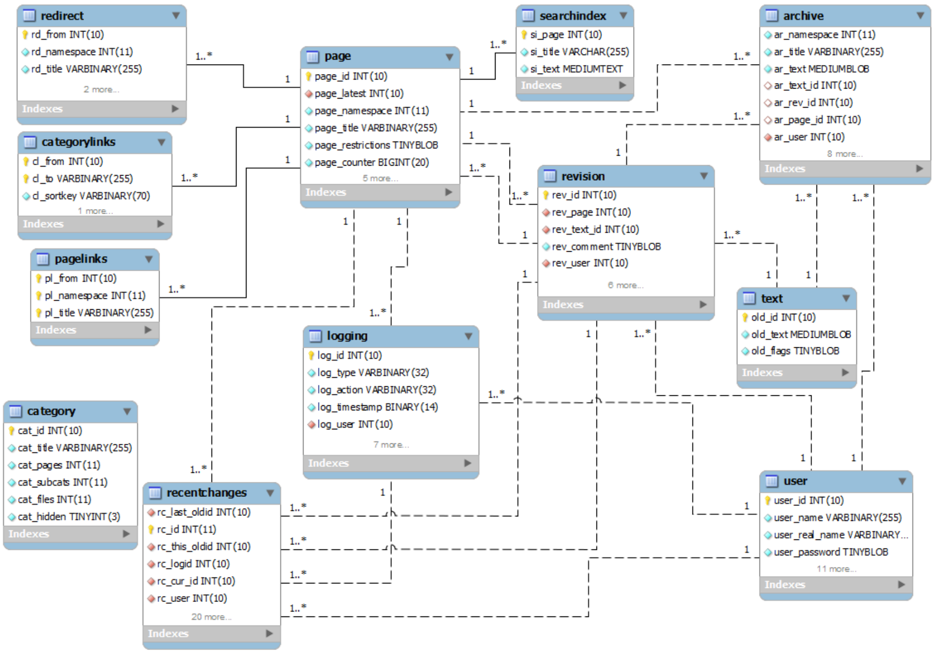


Figure 2.2: MediaWiki database schema (partial view).

delete, restore, protect, etc.); *searchindex* table is used to provide text searches; *revision* table stores metadata for every edition from wiki users; *archive* table keeps information about deleted pages; *text* table holds the wikitext of individual page revisions; and finally, *user* table stores all the information about the wiki users.

2.2.4 Successful Case Studies

Wikis have succeeded in all the areas described above (Section 2.2). This section shows successful cases for each of them.

Wikipedia on its own already represents a success of wikis. Even though, there are many more *open communities* wikis, Wikipedia is on the top ten world websites (sixth in Alexa's ranking), with over 100,000 hits per second. It is worth noting that it is supported by a non-profit organization, the Wikimedia foundation. Moreover, the funding comes from donations and there is no advertising. This makes its success even more remarkable in a market dominated by big multinationals.

Melbourne is one of the pioneers in *e-government* with its Future Melbourne Wiki[8]. The aim of this wiki is drafting a ten year city plan for the City of Melbourne. It has public consultation periods, where the general public as well as council workers can collaborate together. After a review [Gro09], several advantages were identified by the use of a wiki as the enabling technology: (*i*) public participants performed hundreds of contributions; (*ii*) successful management of collaboration between general public and internal staff; (*iii*) participation was diverse and of a high quality; (*iv*) easier to use than corporate intranets; (*v*) helped to communicate and to share ideas and information, and so on.

Several *organizations* have successfully introduced wikis into their intranets. *Microsoft*[9] [GP10], *IBM*[10] [Due08] or the *FBI*[11] found in wikis a way to capture and share the knowledge of their members.

Wikis are well suited to be used in *education* or learning environments. The success in this domain can be indicated by the large number of universities that have already implemented this technology[12]. The use of wikis in education varies from encyclopaedias, support in courses, a way to assess subjects or as shared spaces where students can collaborate together [GKSK11, TLEC11].

Another successful case is *Wikia*[13], which is a wiki hosting company (i.e., a wiki farm). It has around 275,000 wikis and 45 million visitors each month. Wikia revenues come mainly from advertising.

[8] `www.futuremelbourne.com.au/wiki/view/FMPlan` (accessed December 2012).

[9] Micropedia `http://news.cnet.com/8301-13860_3-9886332-56.html` (accessed December 2012)..

[10] Bluepedia `http://www-03.ibm.com/press/us/en/pressrelease/23218.wss` (accessed December 2012).

[11] Bureaupedia `http://fcw.com/articles/2008/09/26/fbi-creates-knowledge-wiki.aspx` (accessed December 2012).

[12] List of some university wikis `http://universitywikinodewiki.wikia.com/wiki/University-wikis` (accessed December 2012).

[13] `www.wikia.com` (accessed December 2012).

2.2.5 Current Research Issues

This dissertation reflects on two main wiki principles, open and simple, since they are not always so. Wikis are easy to edit but hard to (*i*) initially set up (technical skills are required), (*ii*) restructure (tedious and cumbersome process), and (*iv*) customize (all users see the same content). This Thesis addresses these three challenges in Chapters 3 (Wiki Initialization: Aligning Wikis with Organizations), 4 (Wiki Refactoring through Mind Map Manipulation), and 5 (Wiki Customization through Web Augmentation Techniques).

In addition, Lykourentzou et al. [LDP$^+$12] identify several open research issues, where the non-technical oriented ones are the most challenging for researchers and can be summarized as follows:

- *Assessment of the return on investment (ROI)*: This challenge is interesting from a business point of view, and critical when it comes to launch a new corporate wiki project.

- *Identification of the success factors of a corporate wiki*: There are already some works [GP10, LDP$^+$11] on this regard, but the proper identification of these factors may affect the future of corporate wikis. This issue can also be generalized to other domains (e.g., education or e-government).

Finally, Ward Cunningham has opened a new trend in wiki research: *Smallest Federated Wiki* [Cun11]. His new vision of wikis innovates in three ways:

> "It shares through federation, composes by refactoring and wraps data with visualization."

In federated wikis, instead of editing a common wiki, each user has her wiki, which shows and brings information from different sources (i.e., servers, websites, sensors, other (federated) wikis). Thus, in Cunningham words "your pages and my pages don't get mixed up at least we want to".

The idea behind is that each user can tell her own *story* about the data. This new trend will be subject of discussion and debate but certainly, also of research.

2.3 Model Driven Engineering

MDE is a broad and consolidated approach used in all branches of software engineering and it comprises other approaches. *Model Driven Development* (*MDD*) is a software development approach in which the focus and primary artefacts of development are *models* (*vs* programs) and, is based on two time-proven methods: *abstraction* (realm of modeling languages) and *automation* (realm of tools) [Sel07]. The *Object Management Group*'s (*OMG*) initiative for MDD is the *Model Driven Architecture* (*MDA*) [OMG] and proposes a set of standards. Model Driven Engineering (MDE) is a broader trend that encompasses the previous terms plus other tasks based on models such as *Model Driven Reverse Engineering* (*MDRE)*, *Model Driven Interoperability* (*MDI)*, *Model Driven Web Engineering* (*MDWE*), etc. This is graphically represented by Ameller in his Master Thesis [Ame09] as seen in Figure 2.3, MDE is a superset of MDA, which is in turn a superset of MDD.

Thus, the superset of MDE represents a shift in the field of software engineering. Bezivin [Béz04] suggests that similar to the object-oriented principle "*Everything is an object*", in model engineering, the basic principle "*Everything is a model*" is key to identify the essentials of this trend. MDE treats models as first class citizens as opposed to object oriented programming, where objects are first class entities.

In short, the intention of MDE is to raise the level of *abstraction* in program specification and increase *automation* in program development [Bat06].

Next, MDE and its main components are defined and motivated. After that, the main concepts related with MDE that appear throughout the Thesis are described.

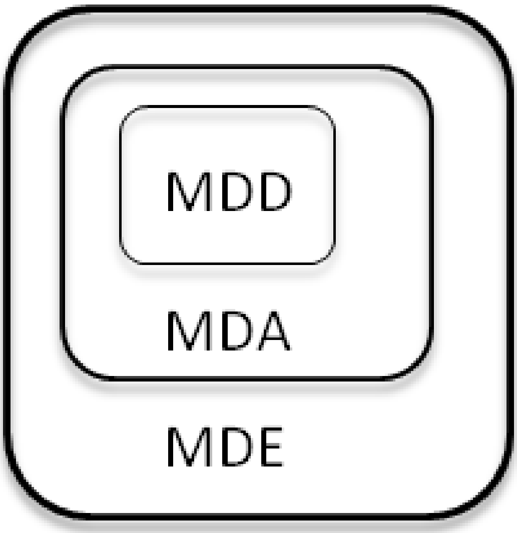


Figure 2.3: MDE is a broader term that encompasses other model based initiatives.

2.3.1 Definition

Model Driven Engineering is a software engineering approach that addresses platform complexity and the inability of third-generation languages to alleviate this complexity and express domain concepts effectively. For so doing, it combines (*i*) *domain-specific modeling languages* (*DSML*). DSML are described using metamodels, which define concepts and their relationships within a domain; and (*ii*) *transformation engines* that use models to synthesize software artefacts [Sch06].

2.3.2 Motivation

The use of models as the primary development artefacts has reported advantages in terms of productivity, quality, automation, standardization, communication, portability, maintainability and interoperability [MD08a]. However, the lack of knowledge of how and when to use MDE may lead to failure cases. Hutchinson et al. [HWRK11] illustrate MDE influences, *positive* and *negative*, for the three main impact factors, namely:

- **Productivity**:
  - *Time to develop code.* It is reduced by automatically generated code but it may be increased when it comes to implement model transformations and executable models.
  - *Time to test code.* There are less common mistakes and the testing can be performed to models directly. However, there is an extra effort in model validation and testing of model transformations.
  - *ROI on modeling effort.* Modeling harnesses creativity solutions and helps to understand the system as a whole. Over-modeling or even increased complexity may be an added problem.
- **Portability:**
  - *Time to migrate to a new platform.* Models are independent of the platform, needing only new transformation for a migration. The implementation of new transformations also requires effort.
- **Maintenance:**
  - *Time for stakeholders to understand each other.* Models are easier to understand existing systems but the generated code may be hard to understand.
  - *Time needed to maintain software.* The maintenance is done at model level, reducing the time needed, and there is a trace automatically created (from models to code). However, this time may be increased to synchronize the models with the code.

MDE has to be applied in those contexts where it influences positively, although it is not always easy to absolutely measure its benefits. Next, the main elements of MDE are explained.

2.3.3 Models

Since MDE revolves around models, we need to address the central notions about models. When "someone models" it means that she is representing something (i.e., a *system*) using something else (i.e., a *model*) [MFBC10]. Therefore, the first step is to understand what a model is. OMG defines it as [mem03]:

> "A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language."

The main elements of this definition are: (*i*) a model is an abstraction or simplified view of a system, (*ii*) with an initial purpose, and (*iii*) presented in a (modeling) language. In this definition, it should be note that for the language to be automated it must be defined in a modeling language. Besides that, it is worth to highlight that the information contained by models has an initial intention or purpose, although it can be later reused for some other intention [MFBC10].

Seidewitz [Sei03] differentiates between *descriptive* models, when the model is used to describe a *system under study* (*SUS*), or *specification* models, when it is used to specify an SUS or a class of SUS. Thus, models are categorized whether they are earlier or later than the system.

The key characteristics a model must fulfil to be useful and effective, are described by Selic [Sel03] as:

- *Abstraction*: The essence can be easily understood by removing or hiding details, obtaining a model as a reduced rendering of the system.

- *Understandability*: A model must be easier to understand than the modeled system. What remains after abstracting the model away, must be presented in a notation that appeals to intuition.

- *Accuracy*: A model must be a realistic representation of the modeled system's features.
- *Predictiveness*: A model should allow to correctly predict the modeled system's interesting but nonobvious properties, either through experimentation or formal analysis.
- *Inexpensive*: A model should be significantly cheaper to construct and analyze than the modeled system.

As previously stated, a model is presented in a modeling language; this language is in turn defined by a model, called *metamodel.* A model conforms to a metamodel in the same way that a computer program conforms to its grammar, or a XML conforms to its XML Schema. Next section defines metamodels.

2.3.4 Metamodels

OMG defines a metamodel [mem03] as:

> "A model of models."

Even though this is a scarce definition, one can sense the "meta" nuance of "modeling a model". OMG also defines a metamodel [OMG02] as:

> "A metamodel is an "abstract language" for describing different kinds of data; that is, a language without a concrete syntax or notation."

The main elements of this definition are: (*i*) a metamodel defines a language to represent data of a concrete domain, and (*ii*) it has no specific syntax (i.e., it is abstract). Thus, a metamodel is a language for models adapted for a certain domain. On the other hand, if the intention is to create an executable language for that domain, other elements complement metamodels to obtain that language. This language is called a *Domain Specific Language* (*DSL*). Next section delves into further details on DSLs.

2.3.5 Domain Specific Languages

This section describes what a Domain Specific Language (DSL) is and its components, development phases and several advantages reported in the bibliography.

A DSL is defined by van Deursen et al. [vDKV00] as:

> "A domain specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain."

Vallecillo [Val10] recalls that the definition of a DSL (or *Domain Specific modeling Language DSML*) has three main parts: (*i*) the domain concepts and the rules that constraint them i.e., the abstract syntax, (*ii*) the notation used to represent these concepts i.e., the concrete syntax and, (*iii*) the semantics of the language:

- *Abstract syntax*: It is usually defined using a metamodel, which 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.

- *Concrete syntax*: It comprises a mapping between the abstract syntax and its visual (more intuitive) or textual (more expressive) representation.

- *Semantics*: It defines the meaning and it may be represented as a metamodel. In practice, the semantics are not always explicitly defined.

The development phases of a DSL are described by Mernik et al. [MHS05] as:

- *Decision*: An initial decision has to be made whether to use a *General Purpose Language* (*GPL*), develop a new DSL or reuse an existing one (if possible, this would be the best option since there is no need to invest resources, both economical and human expertise, in its development). The investment of develop a DSL has to pay for itself by subsequent less expensive systems development and maintenance.

- *Analysis*: In this phase, the domain knowledge is identified (e.g., concepts, descriptions, terminology, scope, etc.). A main outcome of this analysis is a *feature diagram,* which 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.

- *Design*: Two orthogonal dimensions are taken into account: relationships between the DSL and the existing languages, and the formal nature of the design description. The former distinguishes if an existing language is used (*piggyback*), is restricted (*specialization*) or is extended with new features (*extension*). The latter refers to an *informal* design in natural language or a *formal* design with a grammar, regular expressions, etc. A corollary here is “design only what is necessary”. In this phase, the domain concerns have to be formalized in an abstract (i.e., DSL metamodel) and one or more concrete syntaxes (textual or visual).

- *Implementation*: Authors identify seven patterns for executables DSLs: interpreter, compiler/application generator, preprocessor, embedding, extensible compiler/interpreter, commercial off-the-self (COTS) and hybrid (a combination of the previous approaches). More details about each one of them can be found in [MHS05].

- *Deployment*: During this phase, end-users (e.g., domain experts) utilize the DSL for the specification of domain models.

In addition, Visser [Vis07] adds a sixth phase to the cycle:

- *Maintenance*: When new requirements (or any other kind of software evolution) need to be reflected, the DSL has to be updated.

As for the motivation of adopting DSLs as opposed to GPLs, DSLs may present several advantages [vDKV00, Cza05] :

- DSLs provide *abstractions* of the problem domain. Thus, domain experts can understand, validate, modify, and even develop the programs by themselves.

- DSLs give a natural notation (i.e., *concrete syntax*) for the domain at hand; this avoids syntactic clutter of GPLs.

- DSL programs are *concise*, *self-documenting* and can be *reused* for different purposes.

- DSLs enhance *productivity*, *reliability*, *maintainability* and *portability*.

- DSLs have domain *knowledge* embedded, enabling the *conservation* and *reuse* of this knowledge.

- More errors (*error checking*) can be found with static analyzers and report the errors in a language more familiar for the domain expert.

- The code can be *optimized* based on domain-specific knowledge.

- The domain-specific knowledge explicitly captured by the DSL can be used to *improve* developer *tools*.

Transformations are very important components of MDE. Next section provides more details on this regard.

2.3.6 Transformations

OMG defines a model transformation [mem03] as:

> "Model transformation is the process of converting one model
> to another model of the same system."

MDE is not only applicable for (*i*) creating new software systems, but it can also be used for (*ii*) modernizing or reengineering existing systems [ADM, UN10]. The former implies developers to create abstract models describing the system. These models are then transformed to generate new software artefacts (e.g., source code, documentation, models). The latter first requires model harvesting (a.k.a. model extraction), i.e., the process whereby models are obtained from other software artefacts (e.g., code, databases, spreadsheets, etc.) [IM10, RGvD06]. In both approaches, model transformations are key in the process, allowing to (semi) automatically create software artefacts or to reengineer existing ones.

Figure 2.4 shows the basic concepts involved in model transformations. A transformation is defined with respect to a source (or input) metamodel and a target (or output) metamodel. The transformation engine is in charge of executing tranformations. The transformation uses concrete models, which conform to their metamodels. It is also possible to have several source (or input) and target (or output) metamodels in a transformation.

It is worth distinguishing the top level categories of transformations [CH06]:

- *Model-to-text (m2t)*: A model to text transformation has strings as output, which are used to generate system artefacts (e.g., java code, XML descriptors, etc.).

- *Model-to-model (m2m)*: This kind of transformation generates models conformant to their target metamodel.

- *Text-to-model (t2m)*: This sort of transformation parses source code to extract models from existing system artefacts.

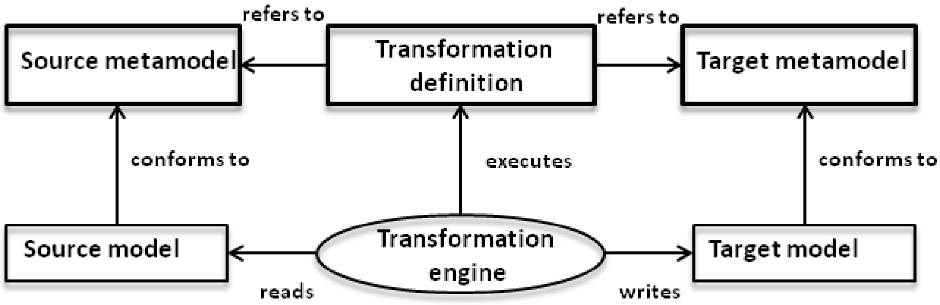


Figure 2.4: The basic concepts of a model transformation are the source and target model and metamodels, the transformation engine and the transformation definition.

Following section describes OMG's layered framework for metamodeling, which presents models at different levels of abstraction and their relationships.

2.3.7 The Four Layer Architecture

The classical framework for metamodeling of the OMG [OMG02] is based on a four layer architecture: (*i*) the described data is found in the information layer. (*ii*) The model layer describes data in the information layer. (*iii*) The metamodel layer is comprised of the descriptions that define the structure and semantics of the models. (*iv*) The *meta-metamodel* layer describes the structure and semantics of the metamodels.

Bezivin [Béz04] renames this architecture as 3+1 layer and he illustrates it as in Figure 2.5. The *M0* layer is the modeled system in the real world. The next layer, *M1*, is where a model represents the system. A model conforms to its metamodel, which is defined in the *M2* layer. A metamodel in turn, conforms to its meta-metamodel in layer *M3*. The meta-metamodel conforms to itself. A layered architecture is commonly used to define the relationships among languages and models involved in a particular *Technical Space* (*TS*) [KBA02], therefore, technical spaces are explained next.

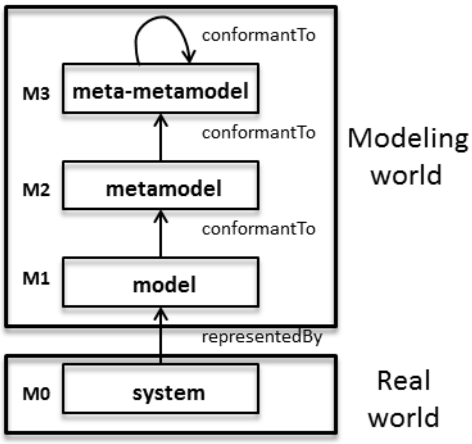


Figure 2.5: The four layer architecture revisited by Bezivin.

2.3.8 Technical Spaces

A TS is defined [KBA02] as:

> "A working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities."

Examples include the XML TS, the database TS, the modelware TS, the *Eclipse modeling Framework* (*EMF*) [EMF] TS, OMG/MDA TS and so on. As stated in [BK05], "*bridging technical spaces is especially useful when it brings new capabilities not available in a given space*".

The Figure 2.6 shows OMG's layered architecture realized for the modelware TS and the relational database TS. Following a bottom-up description, both TSs start with real-world entities (a.k.a. Universe of Discourse in database parlance). This "real world" is captured in terms of data that becomes tuples or models (layer M1). This data is collected along some established descriptions as captured by the database schemata or the metamodel (layer M2). Finally, languages exist to describe M2 artefacts (layer M3), e.g., SQL grammar for describing database schemas, or either *Ecore* or *EMOF* to specify metamodels within EMF.

OMG's standard framework MDA [OMG] uses *Unified modeling Language* (*UML*) as its language (layer M2). UML conforms to its

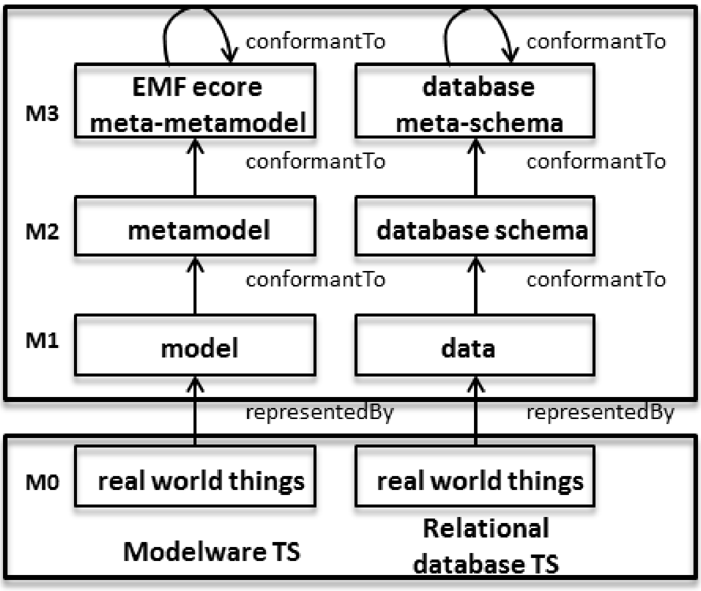


Figure 2.6: The four layer architecture for the relational database and modelware Technical Spaces.

meta-metamodel known as *Meta Object Facility* (*MOF*) [OMG02]. MOF defines an abstract language and a framework to specify, construct and manage technology neutral metamodels.

Bezivin and Kurtev [BK05] motivate the study of technical spaces (TS) with two observations:

- Even though technical spaces present differences purposes, they have *common characteristics*. Capturing and studying these commonalities help to understand and compare new technical spaces.

- Technical spaces show *complimentary features*. Being aware of the strengths and weaknesses of each technology allow to collaborate between different techniques to solve problems at hand.

As in Vallecillo's "village metaphor" [Val08], crossing the (semantic) bridges of the different technical spaces can help to get the best of all worlds. Thus, tools, knowledge or any possibility can be *interoperated* through model driven techniques. Next section describes *Model Driven Interoperability* (*MDI*) in more detail.

2.3.9 Model Driven Interoperability

According to ISO9126 [ISO], interoperability refers to the ability to exchange and use information from different systems to enable them to operate effectively together.

MDI uses models as key artefacts for enabling and achieving interoperability between systems and artefacts, namely: data sets, services, event systems, languages, tools, technological platforms, and so on [BSV10].

A task force created within the network of excellence *INTEROP-NoE* (*Interoperability Research for Networked Enterprise Applications and Software*[14]), published an MDI proposal [JPBB07]. In this proposal is described how a model driven approach can solve interoperability problems by introducing different levels of abstraction among enterprise models.

The *ATHENA* MDI Framework [ath] provides guidelines about how MDE approaches can be applied in achieving interoperable enterprise software systems. As Sun et al. [SDJ$^+$08] state "*each specialized tool contributes to a crucial step in the development process*". Authors also highlight benefits learned from applying model transformation to the tool interoperability problem, such as separation of concerns across the integration process and adaptability and extensibility in defining new tools, among others.

Next section provides examples of successful cases studies on the use of MDE.

[14] `http://interop-vlab.eu/` (accessed December 2012).

2.3.10 Successful Case Studies

Many success stories on the use of MDE are not public because they involve a competitive advantage over their competitors. However, some other successful case studies have been reported on a variety of organizations. A review of industry experiences can be found in [MD08b].

In a case study [Men], it is shown an improvement from 670 days (using a traditional iterative/incremental approach) to 171 (using Mendix[15] modeling platform) in a project programming in Java.

Motorola has been automating code generation from models for over 15 years [BLW05], in addition to using test models for automatic test generation. Even though issues were found, benefits are outstanding:

- 33% reduction in the efforts required to develop test cases.
- Network elements in some projects achieved 65%-85% code-generation.
- 30X-70X reduction in time to correctly fix a defect found during integration testing.
- 1.2X-4X overall reduction in defects and 3X improvement in phase containment of defects.
- 2X-8X productivity improvement when measured in terms of equivalent source lines of code (LOC).

Finally, OMG [OMG] also reports success stories on the use of MDA of organizations like *ABB*, *Siemens*, *Daimler Chrysler*, *Deutsche Bank*, *Lockheed Martin* among others.

Deutsche Bank highlights the use of MDA with benefits like: (*i*) promote the coexistence of diverse technologies, which allows for evolutionary development, (*ii*) decouple the business logic from the

[15] Mendix `www.mendix.com` is a software company with over 75 employees. (accessed December 2012).

technological details of the implementation, allow to flexibly respond to technology changes, or (*iii*) savings amounted to 40% compared to normal development.

Lockheed Martin explains other benefits like: (*i*) application models are platform independent so they can be reused, (*ii*) modellers are isolated from software and hardware details thus, they can concentrate on the problem space, or (*iii*) generation of code eliminates manual coding, which eliminates defects traditionally introduced in the coding phase.

2.3.11 Current Research Issues

During a plenary session in the *MoDELS* workshop "*Challenges in Model-Driven Software Engineering*" [SMB08] the participants identified the following major challenges in MDE:

- *Model quality*. The necessity to deal with quality aspects in modeling and MDE.

- *Run-time models*. Several challenges arise such as representing dynamic behaviour, relation with static models, maintainability of such models, and so forth.

- *Requirements modeling*. How to model requirements, integrate requirements specifications into modeling, or traceability between requirements and models are challenges still to tackle.

- *Standards and benchmarks*. There is a need for standards and benchmarks in order to compare tools and approaches.

- *Modeling languages*. Although this is one of the main research topics, some questions remain open, like how to support better modularity in MDE, or how to deal with multi-models.

- *Domain-specific modeling*. Here arise challenges such as how to develop and integrate models using different domain-specific modeling languages (DSMLs), how to increase reuse across different DSMLs, and so on.

- *Empirical analysis*. How to deal with the trade-offs between realism and control or how to obtain adequate estimations for MDE processes.

- *Model verification and validation*. Challenges include how to verify, validate, debug and test models and generated code; automatically generate test cases from models; or, support formal verification of models.

- *Process support*. Questions here include: how existing process embrace MDE; how to teach MDE; or, how to incorporate MDE environments to MDE processes.

- *Fuzzy modeling*. Models are not always complete and some inconsistencies need to be tolerated, which leads to deal with imperfect, incomplete, imprecise, ambiguous or inconsistent models.

- *Industrial adoption*. Research of negative and positive industrial cases is basic for the engineering aspect of MDE. Technological threshold, learning curve, legacy code and return of investment are vital topics when it comes to adopt MDE in the industry.

- *Formal foundations*. Verification and validation needs to be done in a way that the user of the modeling environment does not need expertise in the different formalisms and techniques for verification.

- *Scalability issues*. MDE community must focus on (*i*) intrinsic scalability issues for MDE, (*ii*) what elements of MDE do not scale well and why, and (*iii*) scalability problems for MDE that cannot be addressed with solutions of other software engineering domains.

- *Models consistency and co-evolution.* Systems evolve, so do models and metamodels. A major challenge is to assess the impact of changes in models, metamodels and generated artefacts. Support for inconsistencies, model-code synchronisation and round-trip engineering is another open research question.

2.4 Conclusions

During this chapter a background on wikis and Model Driven Engineering has been given, as the basis for understanding the content of this thesis.

The aim was to introduce the reader to the notions presented in the following chapters. We refer to the cited articles for further details on any of the mentioned topics.

Chapter 3

Wiki Initialization: Aligning Wikis with Organizations[1]

> "Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution."
>
> – *Albert Einstein.*

3.1 Overview

Wikis are main exponents of collaborative development by user communities. These communities may already exist (e.g., company employees in corporate wikis) or may be created around the wiki itself (e.g., community of contributors in *Wikipedia*). In the former case, the wiki is not created in a vacuum but as part of the information ecosystem of the hosting organization. As any other Information System resource, wiki success highly depends on the interplay of technology, work practice and the organization. Thus, wiki contributions should be framed along the

[1] Parts of this chapter have been previously presented [DP11a, DP11b, DP12].

concerns already in use in the hosting organization in terms of glossaries, schedules, policies, organigrams and the like.

This chapter addresses the following research question: *How can existing corporate strategies permeate wiki construction while preserving wiki openness and accessibility?* To this end, we introduce the notion of "Wiki Scaffolding", i.e., a wiki installation that is provided at the onset to mimic these corporate concerns: categories, users, templates, articles that are all initialized with boilerplate text and are introduced in the wiki before any contribution is made. We propose the use of DSLs as the engineereering means. To retain wikis' friendliness and the capability to engage layman participation, we propose *scaffoldings* to be specified as mind maps. Mind maps are next *exported* as wikis. We show the feasibility of the approach introducing the *Wiki Scaffolding Language* (*WSL*, pronounced "whistle"). WSL has been implemented as a plugin for *FreeMind* [Fre], a popular, open source tool for mind mapping. Hence, WSL expressions are mind maps. Mind maps are diagrams that capture ideas around a central topic [BG10]. Our bet is that users may have already been exposed to mind maps, and even to FreeMind, hence reducing the learning curve for WSL. Through a single click, the WSL plugin for FreeMind generates the wiki along the lines of the scaffolding specification.

The WSL source code is available in the official FreeMind's GIT repository `http://bit.ly/xsA040`, and it is open to contributors. Additional information, installation instructions and a video showing WSL at work are available at `http://www.onekin.org/wsl` and `http://vimeo.com/31548363`.

The structure of this chapter follows the development phases of a DSL (as described in Chapter 2): decision (Section 3.2), analysis (Section 3.3), design (Section 3.4), implementation (Section 3.5) and deployment (Section 3.6). Finally, a discussion through the related work (Section 3.7) and conclusions (Section 3.8) end the chapter.

3.2 WSL Decision

The decision in favour of developing a DSL is based on some *decision patterns* as identified by Mernik [MHS05]. A decision pattern is a common situation that developers may find themselves for which successful DSLs have been developed in the past. Behind them there are two general interrelated concerns: (*i*) improved software economics, and (*ii*) enabling of software development by end-users. In this regard, “Wiki Scaffolding” faces two main obstacles. First, it implies an upfront investment before any content is provided. Second, it requires knowledge about the wiki engine (e.g., MediaWiki) and third-party extensions, both outside the competences of the layman. This will make “Wiki Scaffolding” yet another burden for the organization’s IT department since most users will lack the required skills. Akin to the wiki spirit, the scaffolding should be managed by the users themselves. Therefore, both cost-effectiveness and end-user affordability are key enablers of this approach. Based on these two concerns, it advocates for the use of a DSL. Furthermore, given the non-technical nature of the users, collaboration and easy sharing can be promoted by using a visual DSL (as opposed to a textual DSL), which are regarded as more intuitive. Therefore, we advocate for introducing a visual DSL based on mind maps to both capture and enact “Wiki Scaffoldings”.

In addition, the following decision patterns seem to fit well for “Wiki Scaffolding” as a DSL:

- *Task automation*: In tedious tasks (e.g., implementation) that follow the same pattern, a DSL can generate the required code. The tasks required for initializing a wiki (e.g., create several categories) are repetitive and can be executed through SQL statements against the wiki database.

- *Data structure representation*: Data structures whose complexity may make them difficult to write and maintain are easily expressed with a DSL. A wiki may become a very complex data structure and representing it as a mind map simplifies its understanding by end-users. Besides that, capitalizing on an existing mind map tool (e.g., FreeMind) avoids the implementation of a graphical interface and consequently its development cost.

- *System front-end*: A DSL front-end may be used to configure and adapt a system. A DSL may embed all the elements needed for the initial wiki configuration, including extensions, restrictions and user configurations.

The above decision patterns show the appropriateness of a DSL for the development of "Wiki Scaffoldings".

3.3 WSL Analysis

It is important to note that a scaffolding is "piece of code", i.e., a wiki installation. "Pieces of code" that support scaffoldings for different companies would be different, yet they share a family likeness. That is, they belong to the same domain: "Wiki Scaffolding". This section identifies the scope and main abstractions behind this domain. The aim is to capture both the company's work practice and settings as long as they impact on wiki operations. A main outcome of this analysis is a *feature diagram* that describes the domain concepts and their interdependencies. A feature is a prominent and distinctive user visible characteristic of a system [KCH$^+$90]. In classical conceptual modelling, concepts are described by listing their features (i.e., attributes), which differentiate instances of a concept. In software engineering, software features differentiate software systems. Software system features are not only related to user-visible functional requirements of the system, but also related to non-functional requirements (i.e., quality attributes), design decisions, and implementation details [WLS$^+$07].

In a DSL context, 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 [MHS05]. To the best of our knowledge, the notion of "Wiki Scaffolding" has not yet been the subject of a systematic study. Therefore, we need first to quantify the problem statement, and next, to precisely synthesize the main concerns. This entails to assess the extent to which the wiki community suffers the traditional approach, and determine which would be the corporate aspects that, if available at the wiki onset, would have made a change. This is the topic of the next subsection.

3.3.1 The Need for Wiki Scaffolding

To the best of our knowledge, the notion of "Wiki Scaffolding" for wikis is rather new. We firstly need to collect evidences that suggest the necessity of scaffolding, even if they do not term it that way. To this end, we conducted a literature review on wiki usage in organizations. Next, we provide those seven cases that more clearly seems to suggest the need for scaffolding in wikis. The aim is to provide vivid examples outside our own experience, which sustain this work. We have also included wikis in educational settings because they sit in-between organizational and open wikis, offering some pre-existing context but with less demanding constraints than companies.

"*Using Wiki Technology to Support Student Engagement: Lessons from the Trenches*" [Col09]. This paper reports on a failed experiment to use wiki technology to support student engagement. 37% of the students cited difficulties with the use of the technology. Authors conclude that "*had greater instructional scaffolding be provided, in the form of lab-based exercises and the creation of an accompanying instruction handout, then maybe some of those students that experienced technical difficulties, or self-confidence issues, would have posted to the class Wiki*". "Wiki Scaffolding" can help to readily provide (*i*) wiki templates that guide and advice student contributions, or (*ii*) wiki categories along the terminology set at the classroom.

"*Designing Knowledge Management Systems for Teaching and Learning with Wiki Technology*" [RRO05]. This case study reports on the use of wikis to support collaborative activities in a knowledge management class at a graduate-level information systems course. The authors indicate that "*Wiki technology can be used as a collaborative learning technology, but a lot of design needs to be done before bringing it into the classroom*". The paper indicates that "*the initial findings suggest that effective... use of a wiki... is contingent upon familiarity of both students and instructors with the technology, level of planning involved prior to system implementation and use in class*". This ending is particularly insightful for our purpose: the need of planning prior to system implementation is regarded as a success criterion. This is what "Wiki Scaffolding" is for.

"*Using Wiki to Support Constructivist Learning: A Case Study in University Education Settings*" [TPP09]. Here, the aim is threefold: the assessment of learning, the monitoring of student participation, and the need for communication support in the learning process using wikis. For the purpose of the work presented in this dissertation, we notice the importance given to communication and how basic wiki mechanisms seems to fall short: "*communication problems seemed to be a hinder to the writing of the wiki*" while "*groups which communicated more actively achieved better results, both in terms of quantity and quality*". The authors finally resort to create "*an external discussion forum and encouraged students to use it to discuss and coordinate the development of the wiki*". This seems to suggest that communication design should be included as part of the "Wiki Scaffolding". In addition, it is reported that at times it was difficult to know who was supposed to do what. Some anxiety about the end result was also a concern for many students. Both remarks hint to the need of an existing context where to frame the contributions.

"Did You Put It on the Wiki? Information Sharing through Wikis in Interdisciplinary Design Collaboration" [Phu09]. This paper explores the use of wikis in software development projects. The author states that "*the project wiki was created by the project manager a few weeks after*

the project started. At the beginning of the project, the project manager created a project definition page, which contained important information about the project such as goal, project team members, stakeholders, project description, success criteria, high-level schedule, deliverables, and communication plan. The document was reviewed and accepted by all team members". This suggests the collaborative production of a blueprint for the wiki but, in this case, this blueprint has to be manually turned into a wiki installation.

"*A Wiki Instance in the Enterprise: Opportunities, Concerns and Reality"* [DS08]. This work reports on *ResearchWiki*, a wiki that supports yearly planning work by members of a globally distributed, research organization. The authors point out that users "*preferred to use their project-specific repositories for recording progress in their projects rather than using the ResearchWiki. In many cases these repositories pre-dated the ResearchWiki and had evolved to support the operational needs of particular projects. This included access control as in many cases their project partners were from outside the research division and had not been given access to the ResearchWiki*". This highlights the role of the wiki as part of the information ecosystem, and the fact that companies tend to have stringent access control policies.

"*Enterprise Wikis - Types of Use, Benefits and Obstacles: A Multiple-Case Study*" [ST11]. The study highlights a main factor for wiki success: "*a sufficient number of wiki-articles must exist right from start. Only then will employees perceive and accept the wiki as a useful knowledge base*". This suggests the role of scaffolding as a way to engage users. In addition, "*first wiki properties and wiki structures had been eagerly discussed within internal group meetings, but no strict definitions arose*". This hints the notion of blueprint. Finally, "*the 'built-in' simplicity of the wiki-software is rather a minimum requirement than a success factor*". Besides content editing, simplicity should also be sought in setting up an environment that helps in matters other than editing (e.g., category setting or permission restrictions).

"*Planning for a Successful Corporate Wiki*" [LDP+11]. Based on thirty case studies, this work aims to identify the key factors that affect the success of a corporate wiki. This analysis considers both technological and cultural aspects of wiki adoption. As for the purpose of the work presented in this dissertation, the following success factors are identified: (*i*) *Bottom-up knowledge sharing culture*, scaffolding might help to involve different stakeholders at the very beginning on the search for a balance between free, bottom-up participation and the alignment with the corporate strategies; (*ii*) *content structure to avoid difficulties during navigation and information retrieval*, create a basic initial structure may help users to avoid the empty-wiki syndrome, and provide an early, global view of the wiki goals; (*iii*) *mechanisms to inform users of changes*, e.g., *RSS* feeds or email might complement wiki offerings, and (*iv*) *pre-populating with existing content*. The latter highlights the importance of providing direction settings [ZKK12]. When different people participate, it might not be clear who has to do what. Highlighting specific tasks at the onset in terms of articles to be written (e.g., related to hallmarks already scheduled) might help to spur people to start contributing.

3.3.2 Setting the Features

The previous subsection provides empirical evidence for scaffolding. These insights are now made precise in terms of features. Figure 3.1 depicts the feature diagram for "Wiki Scaffolding". The diagram states that a "Wiki Scaffolding" captures the company settings in terms of existing documentation practices, communication means, restrictions, the existing organigram and finally, presentation concerns. The rationales for these features should be sought in the previous quotations as well as our own experience. Next paragraphs introduce each feature:

- *Documentation Setting.* A common problem for open communities is that of fixing a common terminology and understanding. This is easier in the case of corporate wikis where glossaries,

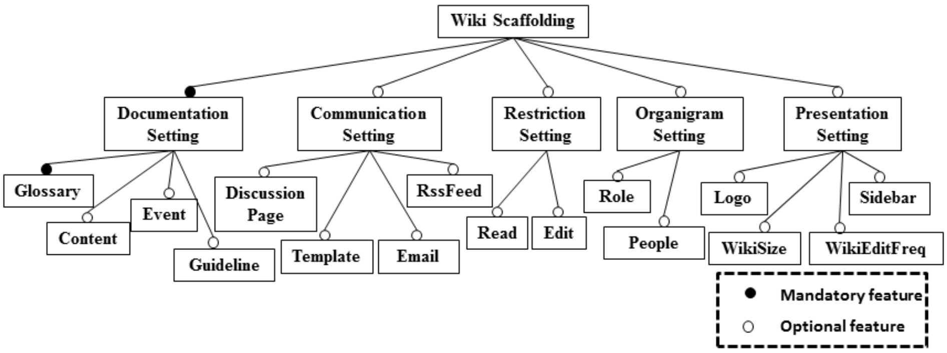


Figure 3.1: Feature Diagram. Different corporate settings are identified as impacting wiki operation.

documentation, guidelines or even some content might already exist. This setting needs to be captured in wiki terms. A basic classification of wiki pages is that of articles, categories and templates. Articles stand for the *content* that is incrementally and collaboratively edited. Next, categories are commonly used as tags to easily locate, organize and navigate among articles. Corporate *glossaries* can help to identify initial wiki categories. Finally, templates provide content to be embedded in other pages. Through parameterization, they permit to reuse and ensure a formatted content along distinct pages. Corporate *guidelines* can then be re-interpreted as wiki templates that guide article editing.

Figure 3.1 depicts *glossary*, *content* and *guideline* as three features of the company's documentation practices that can impact the wiki. Moreover, wikis frequently support ongoing projects where project milestones might need to be accounted for by the wiki. This does not apply to other settings where content is the result of free-willing participation and hence, contribution is not tight to set schedules. Wiki wise, this implies that *event* is a semantically meaningful piece of data, and so should it be markuped and rendered (e.g., through a calendar).

- *Communication Setting*. Wikis are an effective mechanism to support knowledge formation. This implies the existence of coordination and conflict resolution strategies. When wikis are deployed in an existing organization, wikis become an additional means that should be integrated with existing communication channels. This poses a range of questions: Who is going to be notified of what? Does the existing organizational structure need to be mirrored in the wiki? How is such communication currently achieved? Can email/phone/chatting be effectively used for this purpose?.

  Wiki wise, communication can be internal or external. Internal communication is achieved within the wiki. At this respect, two mechanisms are considered: *discussion pages* and *templates*. Discussion pages (a.k.a. “talk” pages in MediaWiki) can be used for discussion and communication with other users. In this way, discussions are kept aside from the content of the associated page. Templates have also been identified as effective means to deliver fixed messages (e.g., warnings, to-do reminders, etc.). On the other hand, external communication refers to the ability to notify wiki changes outside the wiki itself (e.g., through *RSSFeeds* or *email*).

- *Restriction Setting*. Unlike open wikis, corporate wikis normally limit access to its own staff. Permissions are counterintuitive in a wiki setting where openness is a hallmark. Indeed, MediaWiki natively supports a basic mechanism where the scope is the whole wiki: you can either edit the whole set of wiki pages or not. By default, wiki pages can be freely operated. However, permissions are more stringent in a company setting, and finer-grained scopes need to be introduced. Indeed, a study on the use of wikis in the enterprise reports that power relationships and competition between stakeholders created a need to read-only access [DS08]. For the time being, two permissions are considered: *read* and *edit*. Additional permissions will be added in future releases if feedback so advises[2].

- *Organigram Setting*. The notion of *role* captures the distinct functions *people* (e.g., employees) can play within the organization, and which can also impact wiki edition and management.
- *Presentation Setting*. Companies care for their image on the Web (both in the intranet or the extranet). Wikis resort to *skins*[3] for rendering. These skins are platform specific. However, we do not expect our target audience to know about skins. We should strive to capture presentation concerns in abstract terms, better said, through domain criteria that could later be used by the DSL engine to determine the most appropriate skin. Specifically, we consider *wikiSize* and *wikiEditFreq*. Based on the expected size and edit frequency of the wiki, heuristics can make an educated guess about the wiki skin. In this way, the DSL engine frees stakeholders from being knowledgeable about presentation issues, offering good-enough outputs. Notice that the wiki administrator can later change this automatically-generated skin. Additionally, the *logo* and *sidebar* features are introduced for customizing both the headers and the index panes of the wiki.

Based on the analysis made to the "Wiki Scaffolding" domain, we introduce the *Wiki Scaffolding Language* (*WSL*) as the new DSL.

3.4 WSL Design

In a DSL context, 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

[2] MediaWiki permissions include *read*, *edit*, *createpage*, *createtalk*, *upload*, *delete*, *protect* (i.e., allows locking a page to prevent edits and moves), etc

[3] A skin is "*a preset package containing graphical appearance details*", used to customise the look and feel of wiki pages.

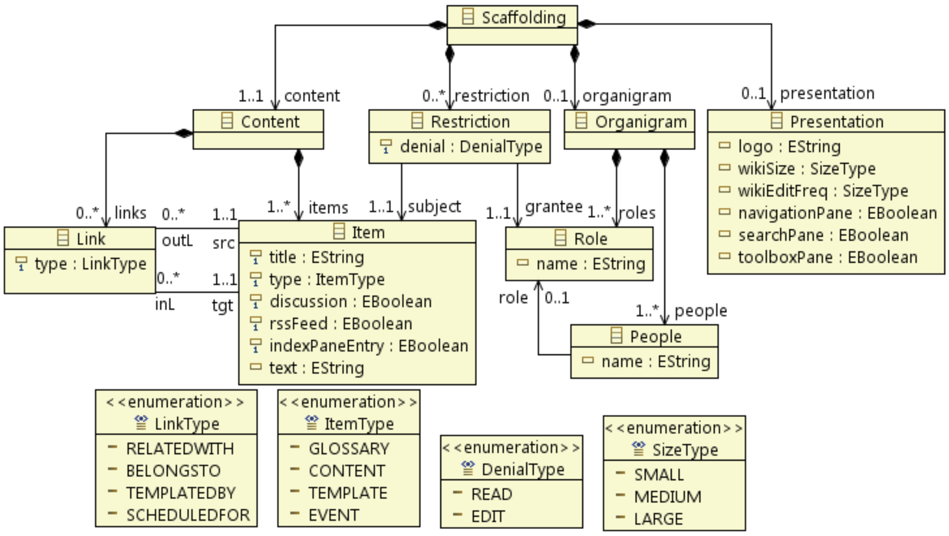


Figure 3.2: WSL metamodel (abstract syntax): a *Scaffolding* provides basic infrastructure as for the *content*, *organigram*, access *restrictions* or *presentation* of the wiki.

by the DSL user [MHS05]. These parameters to-be-set-by-the-user are so provided as a DSL expression. This expression follows a concrete syntax which in turn, is a realization of the DSL's abstract syntax. The latter takes the form of a metamodel where the features captured during the DSL analysis are enriched to be fully operative.

3.4.1 WSL 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. This is expressed as the DSL metamodel (see Figure 3.2). The constructs of the DSL are obtained from the feature diagram. A *Scaffolding* (meta)model includes four main model (meta)classes, namely:

- The *Content* class, which represents a graph described along *Items* and *Links*. *Items* capture the different kinds of data existing in the organization that need to be also available at wiki inception. As identified in Section 3.3, this content includes glossary terms to be included as categories (*GLOSSARY itemType*), content to be readily available as a wiki article (*CONTENT itemType*), guides for content structure (*TEMPLATE itemType*), or events to capture scheduling milestones (*EVENT itemType*). Next, *Links* relate these *Items* together. *Links* are also typed based on the type of the related *items*: a general *item*-to-*item* association (*RELATEDWITH linkType*), to associate a *category* to an *item* (*BELONGSTO linkType*), to associate a *template* to an *item* (*TEMPLATEDBY linkType*), or to associate an *event* to an *item* (*SCHEDULEDFOR linkType*). *Items* also hold three boolean attributes: *discussion* (to indicate whether this *item* is subject to discussion), *rssFeed* (to specify the availability of a feed subscription for this *item*) and *indexPaneEntry* (to capture that the *item* is to be indexed in the sidebar). Finallly, *text* keeps the textual content of its *item*.

- The *Organigram* class, which captures a basic arrangement of *People* (i.e., employees) in terms of *Roles*.

- The *Restriction* class, which binds together three elements: a permission *subject* (i.e., an *Item*), a permission *grantee* (i.e., a *Role*) and a *denial* (i.e., *READ* and *EDIT denialType*).

- The *Presentation* class, which holds properties to guide the rendering of the wiki (i.e., *wikiSize*, *wikiEditFreq*). In addition, three common index schemas are preset, which might appear as wiki panes if so decided: *toolboxPane* (common index entries in the wiki: "*what links here*", "*Upload file*", "*printable version*", etc.), *navigationPane* (common index entries in the wiki: "*recent changes*", "*help*", "*main page*", etc.) and *searchPane* (a search box to locate articles based on content). It is also possible for the designer to define an *ad-hoc* index pane, i.e., its entries are defined through the *indexPaneEntry* attribute: *true* causes an entry in the wiki index pane for that *item*.

3.4.2 WSL Concrete Syntax

The concrete syntax comprises a mapping between the metamodel concepts (i.e., the abstract syntax) and their textual or visual representation. While the abstract syntax addresses expressiveness, the concrete syntax cares for usability as for the target audience. Our target audience is ordinary users. On these grounds, we select mind maps as the concrete syntax for WSL. That is, a scaffolding is to be captured as a mind map. The reasons are twofold. First, mind maps offer a way to display different concerns radiantly (see Figure 3.3). The limited coupling between the different scaffolding features suits this radial distribution. Second, mind maps are catching on for decision making within organizations. Indeed, mind maps are reckoned to be a valuable, visual approach for people to collaborate and share ideas [BG10]. Therefore, we expect organizations to be used to mind mapping, hence, reducing the learning cost of WSL.

However, there exists a plethora of graphical representation and tools for mind mapping. Rather than developing our own visual representation, we decide to capitalize on an existing editor: FreeMind [Fre]. We stick to FreeMind on the following grounds: (*i*) popularity (over 6,000 daily downloads); (*ii*) soundness (over 8 years in the market); (*iii*) interactiveness (e.g., easiness to play around with the map, where nodes and their descendants can be easily moved and edited, branches can be collapsed, etc.); (*iv*) open source (access to the source code); (*v*) extensibility (through plugins); (*vi*) export facilities (maps can be turned into applets, html code, flash code or image formats, which can next be embedded as part of the wiki content); and finally, (*vi*) scripting (*Groovy* scripts can be attached to mind map nodes[4]). Before delving into how WSL constructs are mapped into FreeMind elements, the next subsection

[4] `http://freemind.sourceforge.net/wiki/index.php/FreeMind_0.9.0:_The_New_Features` (accessed December 2012).

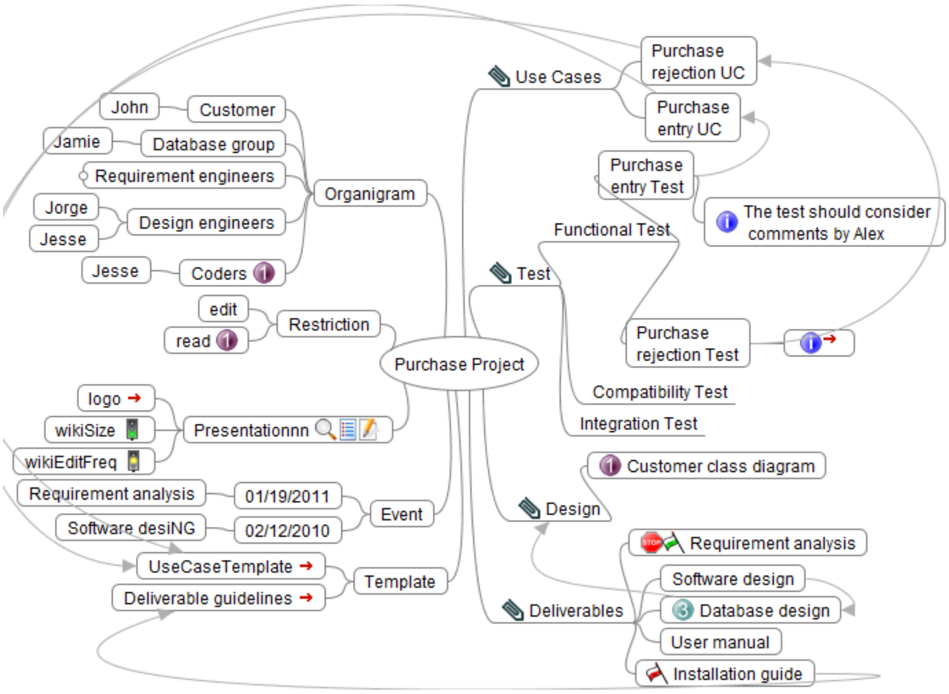


Figure 3.3: A sample Scaffolding for wiki-based software project management (some errors are made on purpose for future debugging).

introduces an example.

WSL to Support Software Projects

Wikis have been proposed for software documentation and planning. The distribution of stakeholders, the need for collaboration and tracking, and the iterative manners that characterize software projects make wikis an attractive platform [Lou06]. Figure 3.3 provides an example for the "Purchase" project.

FreeMind depicts ideas and their relationships as nodes and edges that follow a radial distribution. In our example, the *Organigram* branch captures the existing roles (e.g., *Customer*, *Database group*) as well as the employees (e.g., *Jorge*, *Jesse*) assigned to these roles. The *Restriction* branch lists limitations in terms of wiki operations. The *Event* branch captures two milestones attached to *Requirement analysis* and *Software* *desiNG*. Next, the company already has some guidelines to capture use cases and document deliverables. Such practices should also be adhered to when in the wiki. The *Template* branch refers to two such guidelines through the *UseCaseTemplate* node and the *Deliverable guidelines* node. The *Presentation* branch will impact on the rendering of the wiki based on the expected *wikiSize* and *wikiEditingFreq*. A "*traffic light*" icon is used to indicate the three possible values of these properties: large (🚦 red light), medium (🚦 yellow light) and small (🚦 green light). As for the sidebar, this node includes a navigation pane (denoted by the "*list*" icon ☰) and a search pane (denoted by the "*magnifier*" icon 🔍). The sidebar is finally completed with an index pane (denoted by the "*look here*" icon ✎ on categories *Use Cases*, *Test*, etc.). Regarding to restrictions, "*priority*" ❶ icon sets a restriction whereby *Coders* (i.e., the role) are restricted from *read* (i.e., the denial type) the article *Customer class diagram* (i.e., the item).

As for the corporate glossary, common terms already in use include *Use Cases*, *Functional Test*, *Compatibility Test*, etc. These terms find their way as wiki categories (i.e., *fork* nodes). Hierarchical relationships among categories are captured by describing a category as a child of the parent category (e.g., *Test* parent of *Functional Test*). Wiki articles are denoted as *bubble* nodes (e.g., *Requirements analysis* stands for an article which is categorized as *Deliverables*).

It can look odd to introduce articles at wiki inception since wiki's *raison d'etre* is precisely collaborative editing. Indeed, we do not expect too many articles to be introduced at scaffolding time. However, the need to come up with some articles might be known from the very beginning. The scaffolding permits so by introducing a node whose title becomes the title of the wiki article. For instance, the node *Software design* yields a wiki article with the namesake title. Even more, some relationships might be known at the outset. For instance, trace requirements made advisable to keep a hyperlink between the *Purchase entry test* and the *Purchase entry UC*. This is depicted as an arrow link between the node counterparts.

Based on preliminary user feedback, we also consider article content to be known at scaffolding time. This is realized as a child of the given article (together with the "*info*" icon ⓘ). Figure 3.3 illustrates both options. The content of *Purchase entry test* is explicitly provided as the text of its child node. By contrast, the content of *Purchase rejection test* is already available at the company as a *MS Word* document. FreeMind permits to introduce hyperlinks as node content (denoted through a small red arrow). This facility is used to our advantage to link *Purchase rejection test* to the external document holding its content. Likewise, corporate guidelines can find their way as wiki templates. So far, WSL only supports MS Word documents (exported as XML). At deployment time (i.e., when the WSL expression, i.e., the map is enacted), these external documents are turned into either, article content or wiki templates. The rest of this section provides a detailed account of WSL expressivity.

WSL-FreeMind Mapping

WSL is a visual language for "Wiki Scaffolding" on top of FreeMind. This implies: (*i*) setting a mapping between the WSL metamodel (see Figure 3.2) and the FreeMind metamodel (see Figure 3.4), and (*ii*) a set of constraints that restricts FreeMind maps to be compliant WSL expressions.

First, we introduce the FreeMind metamodel. FreeMind uses a XML schema to denote what is a *valid* map. Figure 3.4 depicts the FreeMind metamodel obtained from this XML schema. A *Map* is a compound of *Nodes* (there is a root node and its descendants). Nodes have a *Text* that represents its title and might hold a *link* to an external document (local or remote) as well as a set of properties mainly referring to rendering concerns. For instance, the *Style* attribute 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. Tree structures are constructed using *Edges*. An edge is a graphical connector that relates a node with its immediate descendants. In addition, *Arrowlinks* are

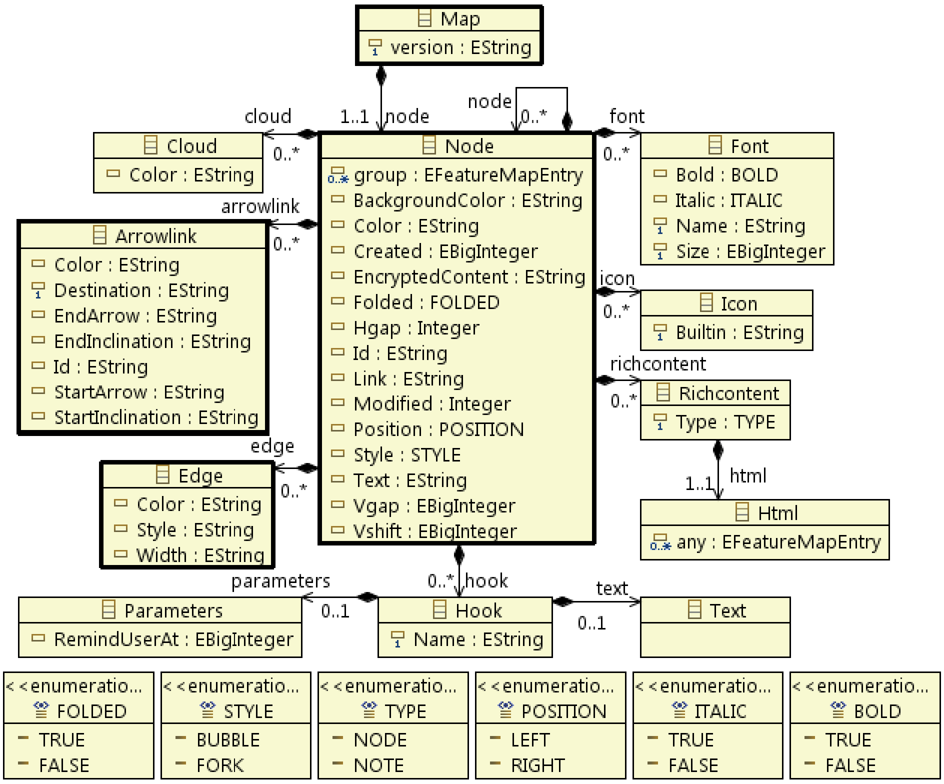


Figure 3.4: Metamodel for mind map drawing as set by FreeMind. Nodes are connected through either *edges* (a node and its descendants) or *arrowlinks* (two nodes no matter their position in the map).

also connectors but in this case, the connection is between two arbitrary nodes (this enables mind maps to support graph-like structures). Finally, *Icons* and *Fonts* can be associated with nodes in an attempt to reflect the underlying semantics of the node (e.g., a user identifies in red colour important nodes). Of course, this semantics resides in the users' head. The remaining elements (e.g., *Richcontent*, *Hook*, etc.) are not reflected in WSL.

Table 3.1: *WikiScaffolding*-to-*FreeMind*-to-*MediaWiki* mapping.

WSL	FreeMind	MediaWiki
Scaffolding	“*root*” node	main page[5]
Organigram	*Organigram* bubble node	n.a.
Role	child of *Organigram* node	wiki group
People	grandson of *Organigram* node	wiki user & user page
Presentation	*Presentation* bubble node	wiki skin[6]
logo	*logo* node	wiki logo
wikiSize	*wikiSize* node	wiki skin
wikiEditFreq	*wikiEditFreq* node	wiki skin
	with “*traffic light*” icons	
navigationPane	“*list*” icon	navigation in sidebar
searchPane	“*magnifier*” icon	search in sidebar
toolboxPane	“*refine*” icon	toolbox in sidebar
indexPane entry	“*look here*” icon at *Item*	element in the navigation bar
Restriction	*Restriction* bubble node and “*priority*” icons 0..9	blacklisted pages for groups[7]
denial	child of “*Restriction*” node	wiki permission.
Item		
title	node text	page title
category Item	fork node	category page
article Item	bubble node	article page
template Item	child of *Template* node	template page
event Item	child of *Event* node	calendar extension[8]
discussion	“*stop-sign*” icon STOP	talk page for that page
RSSfeed	“*flag*” icons	RSS generator for that page[9]
text	child with “*info*” icon i or linked files	page content
Link		
relatedWith Link	arrowLink connector	inter-page hyperlink *[[page]]* *[[:Category:parentCat]]*
belongsTo Link	edge connector	page-category hyperlink *[[Category:parentCat]]*
templatedBy Link	arrowLink connector	template-page hyperlink *{{template}}*
scheduledFor Link	edge connector	event-to-page link in the calendar widget

[5] CategoryTree extension `www.mediawiki.org/wiki/Extension:CategoryTree` (accessed December 2012).

[6] MediaWiki skins include monobook (default), vector (e.g., used by Wikipedia), etc. WSL completes the offer with cavendish, rilpoint, guMax, guMaxDD and guMaxv.

[7] Blacklist extension `www.mediawiki.org/wiki/Extension:Blacklist` (accessed December 2012).

[8] Barrylb extension `www.mediawiki.org/wiki/Extension:Calendar{_}(Barrylb)` (accessed December 2012).

[9] WikiArticleFeeds extension `www.mediawiki.org/wiki/Extension:WikiArticleFeeds` (accessed December 2012).

Once the elements of a FreeMind map are introduced, we proceed to indicate how WSL metamodel elements are going to be depicted using these FreeMind elements. FreeMind extensibility would have allowed us to introduce our own symbols and icons. However, we strive to stick with FreeMind notation (including icons) to minimize the gap to what FreeMind users are accustomed to. Table 3.1 (first two columns) indicates this mapping:

- *Scaffolding* class. The "*root*" node is the FreeMind counterpart of this class.

- *Organigram* class. A bubble node with title *Organigram* denotes the origin of the organigram hierarchy. *Roles* are represented as nodes having *Organigram* as parent. Likewise, *people* (e.g., employees) are interpreted as nodes having *Organigram* as grandparent.

- *Presentation* class. A bubble node with title *Presentation* denotes this class. Boolean properties are captured as icons on *Presentation* (i.e., *navigationPane*, *searchPane* and *toolboxPane*). Value-based attributes are represented as children nodes: *Logo* (captured as a link to an image file), *wikiSize* and *wikiEditFreq*. The last two attributes are decorated with traffic-light icons to account for their values.

- *Restriction* class. A bubble node with title *Restriction* denotes this class. A WSL *restriction* is a triple: *subject* (i.e., an *Item* node), *grantee* (a *Role* node), and a *denial* (i.e., *READ* or *EDIT*). We resort to "*priority*" icons to denote those elements that compose a restriction unit. That is, mind map nodes decorated with the same "*priority*" icon belong to the same *restriction*. Due to the availability of icons in FreeMind[10], *permissions* are limited to ten (*"priority"* icon ⓪..⑨).

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

- *Content* class. There is not a FreeMind counterpart for the *Content* class as such. Rather all nodes in the mind map except for *Organigram*, *Presentation*, *Restriction*, *Event* and *Template* (and their descendants) stand for *Content Items*. The node title behaves as an identifier; so that two FreeMind nodes placed differently but with the same title, stand for the same *Item*. This allows the *Content* graph to be flattened as a FreeMind tree.

- *Item* class. *Items* are typed as *CATEGORY*, *ARTICLE*, *TEMPLATE* and *EVENT*. *Category Items* are denoted as *fork* nodes (i.e., nodes with *FORK* style). *Article Items* are captured as *bubble* nodes. Next, *Template Items* are children of the *Template* node. These nodes can either hold the page text content (i.e., *text* attribute) themselves as a child with the "*info*" icon ℹ or point to external documents from where the content is obtained at compile time (only txt and MS Word as XML exported files in the current version). Finally, *Event Items* are children of the *Event* node. As for the boolean properties, *discussion, rssFeed* and *indexPaneEntry*, the affected *Items* (regardless of their type) are decorated with the "*stop sign*" icon 🛑, a "*flag*" icon 🚩 and "*look here*" icon ✎, respectively.

- *Link* class. *Links* are classified as *RELATEDWITH*, *BELONGSTO*, *TEMPLATEDBY* and *SCHEDULEDFOR*. FreeMind offers two kinds of connectors: *Edges,* which are the default arcs connecting a node with its child, and *ArrowLinks*, which are arcs connecting two nodes anywhere in the map. *Edges* are interpreted as *belongsTo* links when they connect an *Item* to a *category Item* (e.g., Figure 3.3, arc from *Database design* to *Deliverables*) and as *scheduledFor* when they connect an *Item* to an *event Item* (e.g., Figure 3.3, edge from *Requirement analysis* to *01/19/2011*). As for *ArrowLinks*, they sustain (*i*) *RelatedWith* links when they relate an *Item* to another *Item* (e.g., Figure 3.3, arc from *Software design* to *Database design*) and (*ii*) *TemplatedBy* links when the ingoing node stands for a

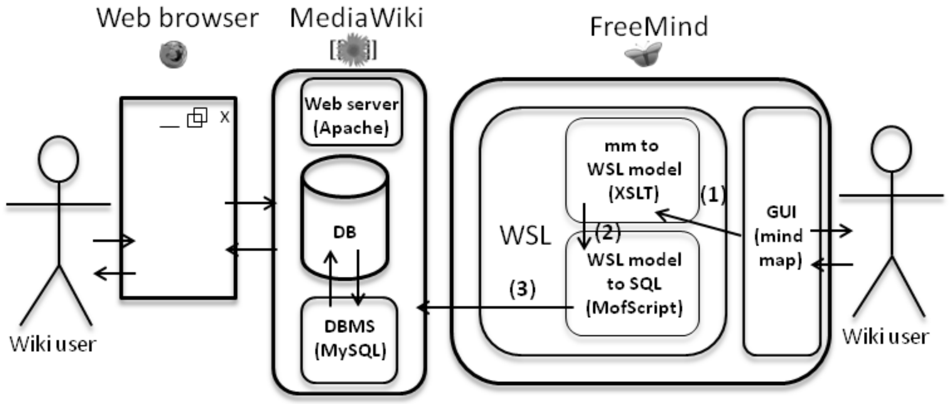


Figure 3.5: WSL architecture: FreeMind as an interface of MediaWiki.

template Item (e.g., Figure 3.3, arc from *Purchase entry UC* to *UseCaseTemplate*).

Constraints

A WSL expression is a compliant FreeMind mind map. However, the opposite does not hold. Some mind maps might not deliver a compliant "Wiki Scaffolding", where compliance is determined by WSL's abstract syntax (Figure 3.2). Therefore, WSL maps are a subset of the possible maps that can be drawn in FreeMind. Specifically, FreeMind mind maps are internally represented as XML files along an XML schema. On top of it, WSL imposes an additional set of constraints that ensures that maps account for compliant scaffoldings (i.e., conform to the WSL abstract syntax). For instance, an *organigram* node should at least have one *role* node. If this is not the case, despite being a perfectly valid mind map, the system raises an error, and the map can not be "exported" as a wiki installation (more examples of errors in Section 3.6.2).

3.5 WSL Implementation

The WSL engine is implemented as a FreeMind plugin. Figure 3.5 outlines the architecture. As previously mentioned, wikis are supported by wiki engines (e.g., MediaWiki). Although files are also an option (e.g., DokuWiki), wiki engines tend to store the wiki content in a database. A "Wiki Scaffolding" is the initialization of a wiki project. Therefore, a "Wiki Scaffolding" initializes a database. In other words, the enactment of a WSL expression causes a set of tuples to populate the MediaWiki database. This process goes as follows (see Figure 3.5):

1. The mind map (i.e., a *.mm* file) is turned into a WSL model (i.e., a *.xmi* file along the WSL metamodel). Since FreeMind maps are XML files, this transformation is realized as a *XSLT* transformation, which is natively supported by FreeMind.

2. This WSL model is next turned into a *SQL* script, through a *MOFScript*[11] model-to-text transformation.

3. This *SQL* script is run against the MediaWiki DBMS[12] (i.e., *MySQL*). In addition, realizing the scaffolding might require specific MediaWiki extensions. Specifically, *Blacklist*, which restricts access to specified pages in a black list; *Calendar*, for event rendering in a calendar; *CategoryTree*, which gives a view of the wiki structure as a tree*; EmailPage*, to send wiki pages by email; and *WikiArticleFeeds*, for turning wiki changes into *RSS* and *Atom* feeds.

As any other plugin, this architecture raises evolution and scalability concerns:

- *Evolution*. The WSL engine might be affected by (*i*) changes in MediaWiki (or its extensions), (*ii*) changes in the underlying

[11] `http://eclipse.org/gmt/mofscript/` (accessed December 2012).

[12] All the connection parameters (database name, db user login, db password, db host name) are obtained from the MediaWiki configuration file *LocalSettings.php*, which is provided by the user in the "*WSL Configuration...*" option.

database schema (this impacts the MOFScript transformation), and (*iii*), changes in the FreeMind metamodel (this impacts the *XSLT* transformation). This is certainly true. But, how real is this threat? Both FreeMind and MediaWiki are stable platforms backed by thousands of installations. In addition, wikis can be upgraded once deployed. Remember that a WSL expression is used just to initialize the wiki. Once the scaffolding is deployed, users can upgrade the wiki to the newest version if required. Notice that the wiki can next evolve in ways that contradict the scaffolding (e.g., new users or templates can show up), but this does not erode the benefits that scaffolding brings at the onset. All in all, wiki refactoring is certainly an issue, and it is addressed in Chapter 4.

- *Scalability*. Although it is not the aim of scaffolding to offer a complete wiki map but just a blueprint, large projects can require large scaffoldings. This can lead to cluttered WSL maps. Fortunately, FreeMind offers view-like mechanisms that permit to filter map nodes based on content and relationships. Testing stakeholders can filter those nodes containing the string "*test*", whereas template-minded stakeholders can restrict the view to those nodes related with a template.

3.6 WSL Deployment

This section describes the common lifecycle of a WSL expression: edition, verification and enactment, and provides some hints about the installation of the WSL engine.

3.6.1 Edition

WSL expressions are edited as mind maps in FreeMind. To give users a head start, the canvas can be initialized with a "skeleton" that draws the

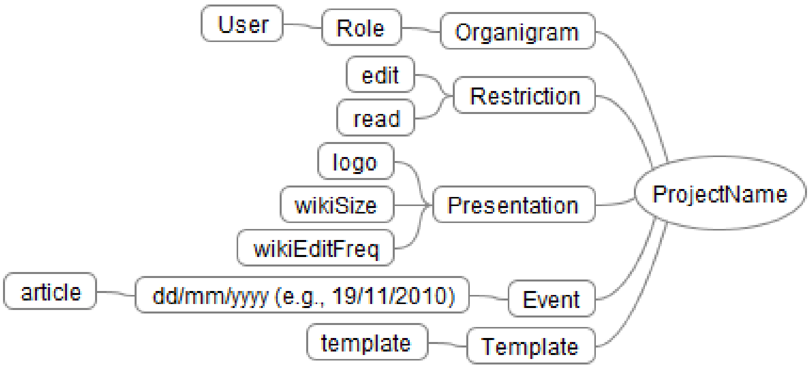


Figure 3.6: WSL skeleton: a basic WSL template to get going.

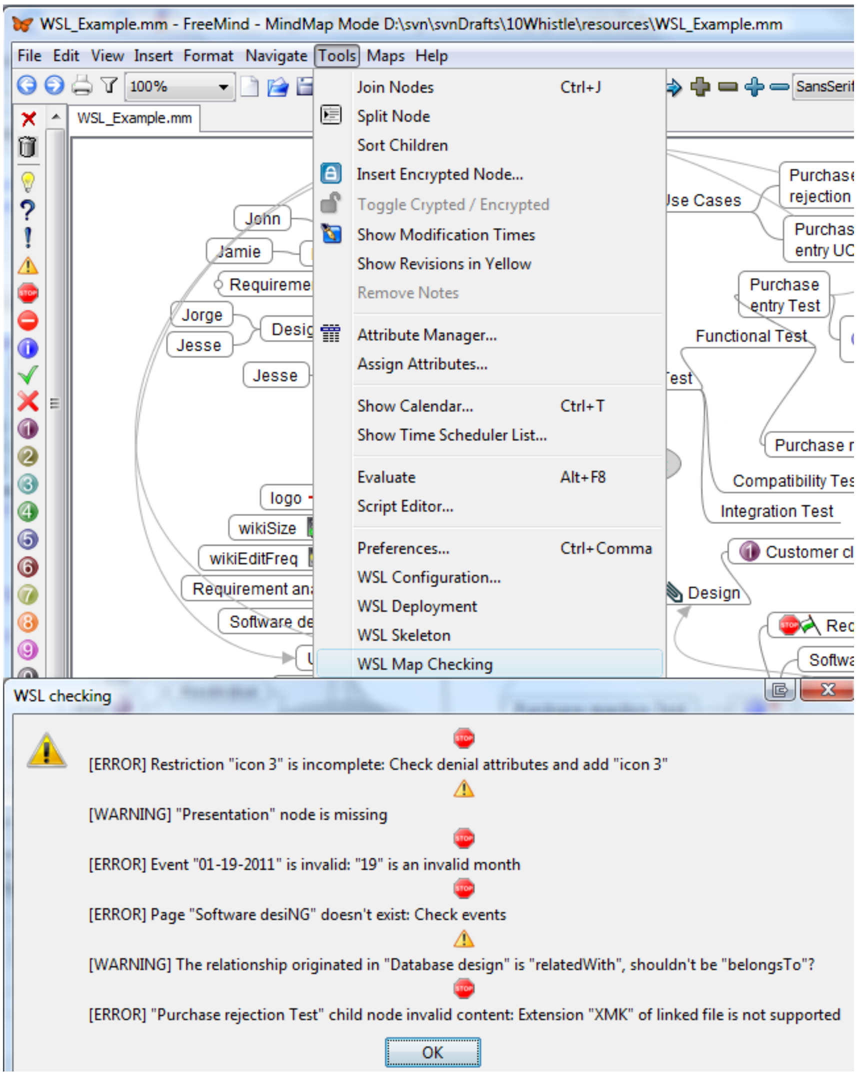
main elements of a scaffolding map (see Figure 3.6). From then on, users are free to handle the scaffolding as any other map. Notice however, that not all maps are scaffolding maps. This moves us to the next subsection.

3.6.2 Verification

WSL maps are a subset of FreeMind maps, i.e., WSL metamodel imposes additional constraints on top of the FreeMind metamodel. Such constraints can be verified on user request or at enactment time.

Figure 3.7 provides a snapshot of the "*Tools*" menu now extended to address WSL maps: "*WSL Configuration...*" permits to configure parameters for the MediaWiki installation; "*WSL Deployment*" causes the generation of the wiki instance from the WSL specification; "*WSL Skeleton*" provides a FreeMind map with the basic WSL nodes (e.g., *Organigram*, *Restriction*, etc.) so that misspells are prevented; and finally, "*WSL Map Checking*" triggers WSL map verification.

Figure 3.7 depicts the verification outcome for our sample problem (see Figure 3.3). Messages can be either warnings or errors. For our sample, two warnings are noted. One informs about the lack of the *Presentation* node which, in this example, is due to a misspelling (e.g., *Presentationnn*). The other warning notifies about a common mistake in wiki construction: setting a *relatedWith* relationship between an article and a category. This is an odd situation that could be mistaken with the *belongsTo* relationship, and so is it indicated. As for errors, they prevent the wiki from being generated. For our sample case, these errors include: a misspelling of an event date (e.g., *01/19/2011*); referring to a non-existent node (e.g., *Software desiNG*); partial definition of a restriction (i.e., either the denial, the employee or the article is missing) (e.g., restriction ③); unsupported document extension (e.g., extension *"XMK"* is not supported).



[ERROR] Restriction "icon 3" is incomplete: Check denial attributes and add "icon 3"

[WARNING] "Presentation" node is missing

[ERROR] Event "01-19-2011" is invalid: "19" is an invalid month

[ERROR] Page "Software desiNG" doesn't exist: Check events

[WARNING] The relationship originated in "Database design" is "relatedWith", shouldn't be "belongsTo"?

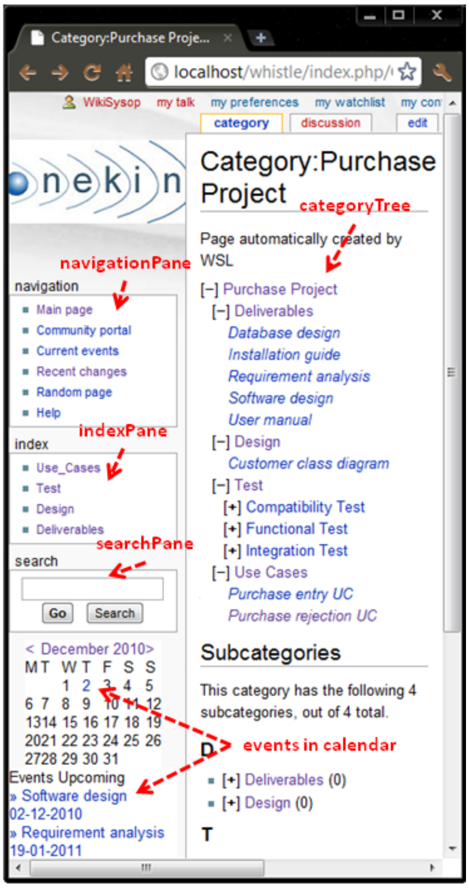
[ERROR] "Purchase rejection Test" child node invalid content: Extension "XMK" of linked file is not supported

Figure 3.7: Verifying WSL expressions. Example for the map at Figure 3.3.

3.6.3 Enactment

By selecting the “*WSL deployment*” option of the *Tool* menu (see Figure 3.7), the current map is turned into a wiki installation in MediaWiki. This means that around 1,000 LOC (mainly *SQL* statements) are automatically generated for the current example.

Figures 3.8 and 3.9 provide three screenshots of the generated pages: the former with the main page (illustrating the use of the *CategoryTree* and *Calendar* extensions), the *Purchase rejection Test* article page (which is obtained from a Word XML document) and the latter with the *Purchase Rejection UC* (which follows the *UseCaseTemplate* also externally obtained). For the purpose of this dissertation, it is enough to show the mapping between FreeMind constructs and the MediaWiki constructs. The last two columns in Table 3.1 indicate this mapping.



Category:Purchase Project

Page automatically created by WSL

Subcategories

This category has the following 4 subcategories, out of 4 total.

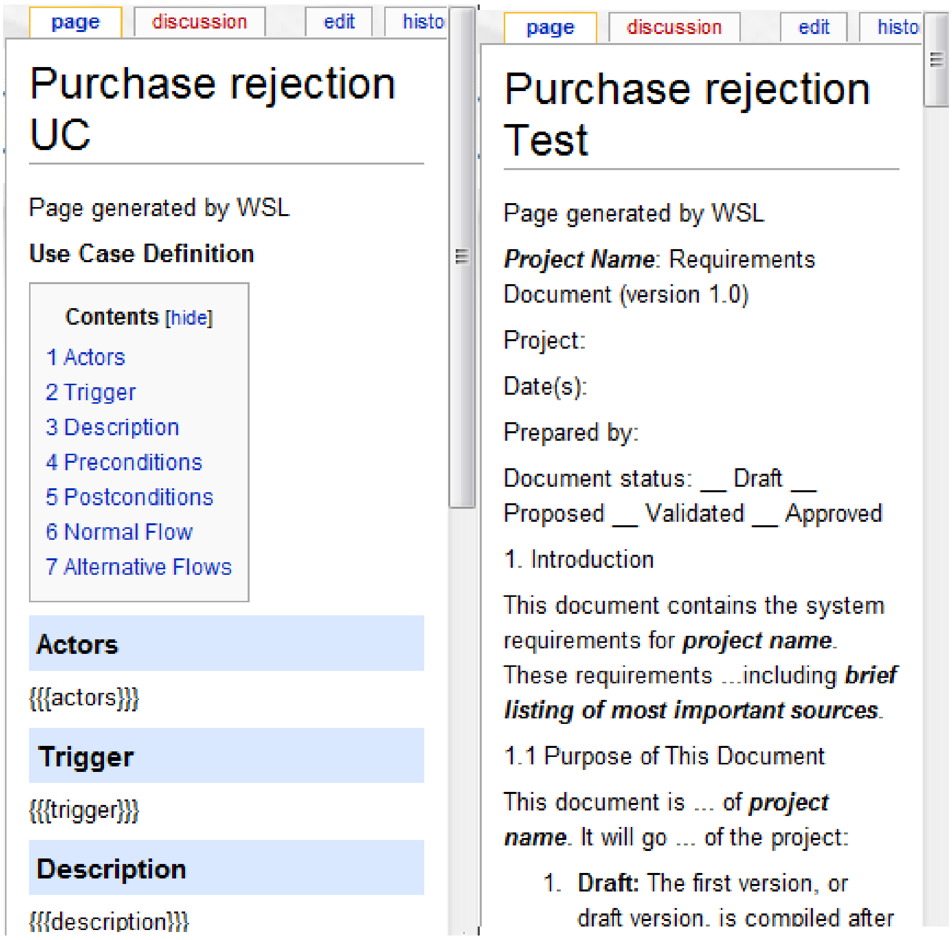
Figure 3.8: Wiki home page generated by WSL.

3.6.4 Installation

For installation, proceed as follows: (*i*) download and install FreeMind[13], (*ii*) download and install the WSL engine[14], and finally (*iii*) download

[13] http://freemind.sourceforge.net/wiki/index.php/Download (accessed December 2012).

[14] http://www.onekin.org/wsl (accessed December 2012).



Purchase rejection UC

Page generated by WSL

Use Case Definition

Contents [hide]

1 Actors
2 Trigger
3 Description
4 Preconditions
5 Postconditions
6 Normal Flow
7 Alternative Flows

Actors

{{{actors}}}

Trigger

{{{trigger}}}

Description

{{{description}}}

Purchase rejection Test

Page generated by WSL

Project Name: Requirements Document (version 1.0)

Project:

Date(s):

Prepared by:

Document status: __ Draft __ Proposed __ Validated __ Approved

1. Introduction

This document contains the system requirements for ***project name***. These requirements ...including ***brief listing of most important sources***.

1.1 Purpose of This Document

This document is ... of ***project name***. It will go ... of the project:

1. **Draft:** The first version, or draft version, is compiled after

Figure 3.9: Template and article pages as generated by WSL for the *"Purchase Project"* example.

a WSL sample[14]. WSL has been tested against MediaWiki 1.16.1 and FreeMind 0.9.0. A detailed explanation and a video can be found at `www.onekin.org/wsl`.

3.7 Discussion through Related Work

To the best of our knowledge, there are not other works that tackle the task of aligning wikis with the corporate setting. Nevertheless, there is an interesting point to debate: *validation*.

Validation is normally conducted through usability studies for the task at hand. For our purposes, this task is the drawing of WSL maps using FreeMind. However, there is nothing special about WSL maps when compared with traditional mind maps. Therefore, WSL usability is that of FreeMind[15]. However, this is not enough. Validation has also to do with appropriateness, i.e., how the metaphor introduced by the tool matches the mental model of users. This is more than just drawing maps. We want to gain some insights about whether the notion of "Wiki Scaffolding" (no matter how it is apprehended) can be valuable to wiki communities. For this purpose, this section conducts a literature review about experiences on using wikis. For each case study, we identify matters related with "Wiki Scaffolding", we depict the WSL maps for each case and finally, we generate the MediaWiki installations[16]. Besides illustrating WSL, each example highlights a scaffolding advantage (in italic). Table 3.2 outlines how different scaffolding matters are realized in these scenarios.

3.7.1 Scaffolding to Promote User Engagement

Prompt user engagement has been identified as a main success factor for wikis [Col09]. "Wiki Scaffolding" gives users an initial setting where some artefacts (e.g., categories, templates, articles) are available from the start. Figure 3.10 depicts a WSL map along the experiences reported in [Col09]. Cole mentions six areas that are know from the start. They could be represented as either articles (e.g., *Paradigm shift*) or categories (e.g., *Development techniques*). Regarding the comment of a student "*there aren't any useful guidelines or tips that could be used*", content about wiki usage (e.g., a "*wikis for dummies*" internal report, or URLs to appropriate places) might be included as page text just by linking that file to the WSL node. Furthermore, FAQ collected in the classroom might be made

[15] Interesting enough, FreeMind was listed in 2009 as a Community Choice Award Finalist for its usability `http://sourceforge.net/blog/cca09/` (accessed December 2012).

[16] Available for inspection at `www.onekin.org/wsl` (accessed December 2012).

Table 3.2: Setting matters that could impact wiki operation.

Scaffolding	Academic wiki	Gaming Community wiki	Veterinary Education wiki	Software Project wiki
Glossary	Keywords of the taught subject	Game jargon characters, worlds, weapons	Clinical taxonomies pathology, drugs, viruses	Software development keywords: use cases, tests
Content	Syllabuses, exams, FAQ	Story line, tricks	Patents, terms of service, sponsors	Technical manuals, README
Events	Exam dates, assignment deadlines	Game releases	—	Planned meetings, project milestones
Guidelines	Exam patterns, assignment guidelines	Weapon explanation, character features	Animal features, treatment steps, student's page	Use case, deliverable template
RssFeed	Doubts	Patch release, announcements, online news	—	Requirement updates
Discussions	FAQ, tough themes	New features, support	Drugs in trial, innovative sponsors	Requirements, test results
Email	Teamwork	Bug communication	teamwork	Meeting with customer
Restrictions	Certain students edit assignments, only lecturers edit exams	Developers' pages	Discussions only by qualified people	Requirements set only by stakeholders and analysts
Roles	Student, lecturer	Developer, player, tester, resellers	Student, nurse, veterinarian	Stakeholder, analyst, designer, coder
Presentation	University logo	Game "look&feel"	—	Organizational image

readily available at the onset. In addition, communication mechanisms (e.g., email, RSS feed and discussion pages) can be added to promote all, student collaboration (e.g., do you know an answer to a common doubt?), encourage participation (e.g., do you agree with the present year assessment method?), and incite the work group (e.g., could we improve our individual grade by working together?).

Based on previous teaching experiences, articles which are expected to raise a debate, can be created with either a companion discussion page (i.e., "*stop-sign*" icon 🛑) or a RSS feed (i.e., "*flag*" icon 🚩). This is the case of the articles *ISD methodologies* and *future directions*. Event though, the articles are empty, the scaffolding already provides the infrastructure to initiate the discussion. In addition, some articles might need to follow

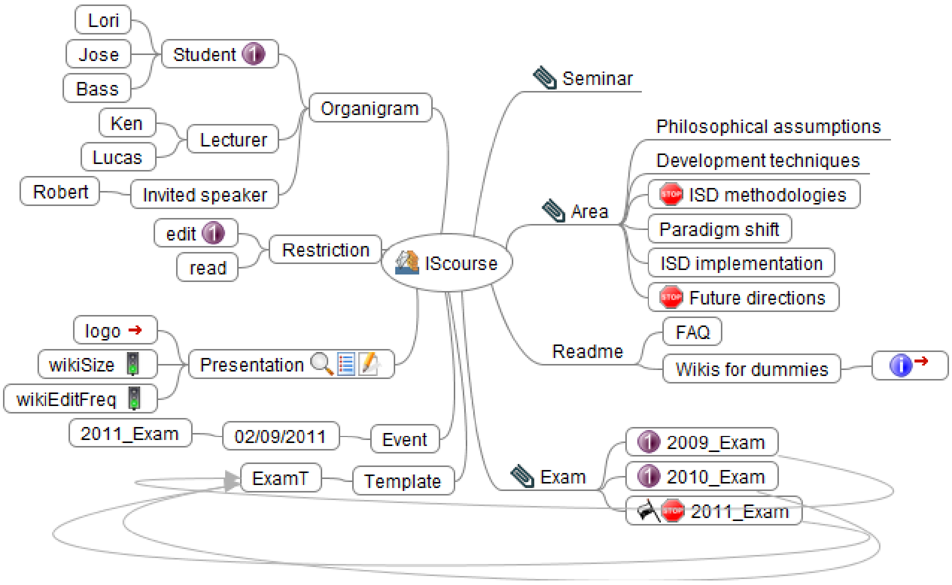


Figure 3.10: WSL scaffolding for a wiki to support student engagement. Output available at `www.onekin.org/wsl/IScourse`. Username: Lori, Password: 12345

some guidelines. An obvious example is that of exams. Templates can be used to guide template-aware articles. The example shows an arrow link from exam articles to the *ExamT* template. Access rights are defined that prevents contributors, belonging to the *Student* role, from editing the *2010_Exam* and *2009_Exam* articles. Exam articles can be qualified by an event. Finally, the expected size and editing frequency are both low as denoted by the green "*traffic lights*" icons 🚦.

3.7.2 Scaffolding to Mirror Existing Organizational Practices

Organizational wikis frequently need to mirror (and follow) existing organizational practices. Introducing wikis in organizations is not easy [GP10]. Stuff might lack the motivation to learn yet another new technology. After all, other collaboration tools may already exist in the organization including email, distribution lists, intranets, etc. "Wiki

Scaffolding" forces to ponder on those practices and resources which might need to be migrated to/integrated into the wiki.

This situation is illustrated by wikis supporting software projects [Lou06]. From a scaffolding perspective, characteristics of relevance include (see Figure 3.3): (*i*) distinct stakeholders work together to organize, track and publish project documentation; (*ii*) wikis act as a version control system to keep track of changes; (*iii*) wikis are useful as *discussion* means (e.g., *Requirements analysis* node); (*iv*) they also provide *rssFeeds* to advise changes (e.g., *Installation guide* node), *email* capabilities for notifications, project milestones as *events* (e.g., a meeting for the *Software design* node), scheduling capabilities, etc. This collaborative management of the project documentation does not occur in a vacuum, but normally adheres to some "work of practice" existing in the company. This includes a role organigram (e.g., *Requirement Engineer*, *Design Engineer*, etc.) where contributions and permissions might depend on the user role (e.g., *Coder*s are not allowed to *edit* the *Customer class diagram* as denoted by the "*priority*" icon ❶), glossaries (e.g., terms such as *Use Case*, *Functional Test* or *Compatibility Test* might be used to categorize wiki content) or company guidelines for artefact production (e.g., a common example is that of use cases).

3.7.3 Scaffolding as a Way to Engage Management

In order to provide value to the organization, wikis have to solve a clearly specified problem and be aligned with the organizational strategy [ST09]. Unfortunately, organizational wikis are in many cases a grass root phenomenon whereby the wiki is introduced by an individual employee or a small group within the organization without the support of management. This bottom up approach frequently fails in having a strategic intent. More to the point, a lack of strategy might result in no clear guidelines about what to contribute, how to contribute and who should make the contribution. An example is reported in [HDW10] where a wiki failure was due to

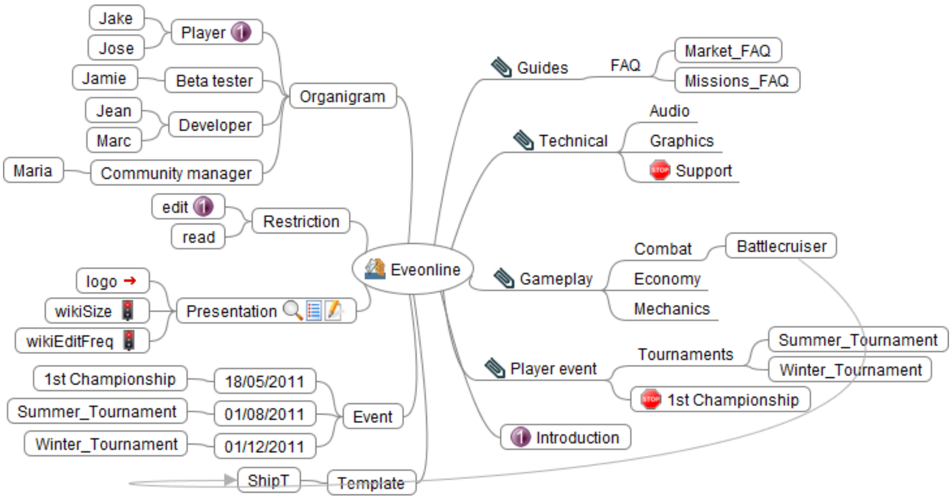


Figure 3.11: WSL scaffolding for a video-gaming wiki. Output available at `www.onekin.org/wsl/Eveonline`. Username: Jake, Password: 12345

an ambiguity in the wiki's aim: some users saw the wiki as a project documentation repository whereas others used it for glossary entries. This led to confusion and dissensions on the wiki's intent.

From this perspective, "Wiki Scaffolding" forces to have a blueprint before releasing the wiki for contribution. Thinking about how the wiki will fit into the existing information ecosystem helps to devise the aim of the wiki in advance. In addition, management support would be facilitated if scaffolding is captured through intuitive means that ease self-edition, sharing or discussion. This favours the use of mind maps.

This situation is illustrated by a video-game community (e.g., `www.eveonline.com`) (see Figure 3.11). The wiki intent is to offer a share space for both consumers and providers of video games to communicate new insights about potential enhancements and new game releases. Contributors are players (a.k.a. gamers) who discuss, share, and edit *content*, *guidelines*, documentation, background and resources (i.e., *glossary*) about their favourite video games. Besides players, developers and testers (i.e., *roles*) also participate to gain insights from the players

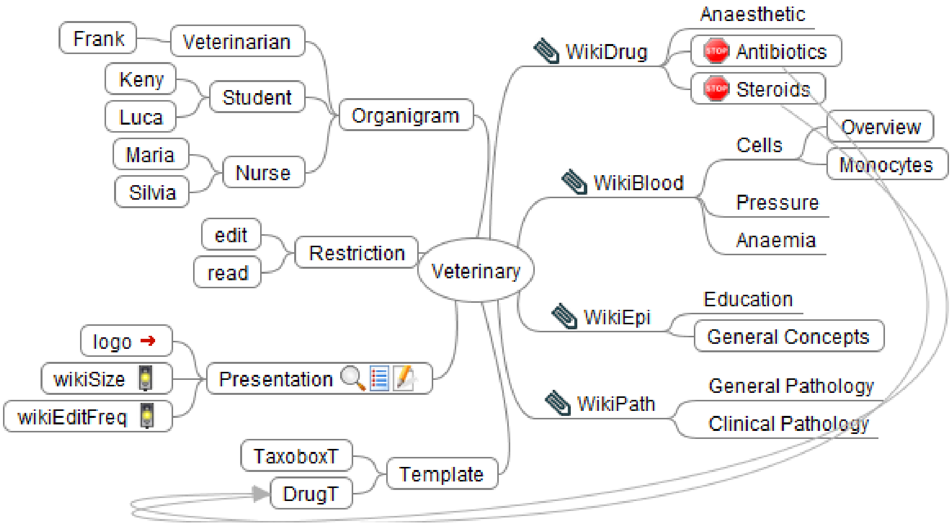


Figure 3.12: WSL scaffolding for a wiki to support veterinary education. Output available at `www.onekin.org/wsl/Veterinary`. Username: Keny, Password: 12345

about how to improve their products[17]. There exist some *restrictions* to both avoid misunderstandings and keep organizational policies untouched (e.g., *Players* are not allowed to *edit* the *Introduction* node as denoted by the "*priority*" icon ➊). Direct communication (through *discussions*, *email* notifications or *RSS* subscriptions) permits developers to know first-hand the players' opinion about new features, bugs, ideas, etc. Common *guidelines* about how to explain game items are represented as templates (e.g., a *Battlecruiser* is a kind of *ship*, so the namesake template is used).

3.7.4 Scaffolding as a Wiki Map

The "rules of practice" which govern a site (i.e., roles, access rights, templates, etc.) should be easily accessible to newcomers. So far, this information is scattered around the wiki, and frequently, hidden in administrative pages. At best, a *README* page can provide some textual description of these practices. From this perspective, a "Wiki

[17] `http://wiki.eveonline.com` (accessed December 2012).

Scaffolding" can play the role of an initial *practice sitemap*. Traditional site maps provide a kind of interactive table of contents, in which each listed item links directly to its counterpart sections on the website. Some wiki engines (e.g., MediaWiki) readily provided such map for categories. One step in the same direction would be the use of "scaffolding maps": an HTML representation of a "Wiki Scaffolding" that permits to readily access the wiki's practices. Notice however, that this will require to keep the scaffolding in sync with the wiki (i.e., new roles, terms, rights, etc.), and to conceive "Wiki Scaffolding" as a supporting infrastructure for collaborative content production whose usefulness goes far beyond wiki initialization. This could be useful for communities where different roles intertwine, and the implications of belonging to a certain role (e.g., in terms of contribution obligations or access rights) should be clearly stated.

This scenario is illustrated by *WikiVet*[18] [BQBB10]. Contributors include veterinarians, veterinary students and nurses (i.e., *roles*), where anonymous users might not be allowed to edit and, sometimes, even read, pages (see Figure 3.12). This restriction increases the trustworthy of the peer reviewed material since all the editors are knowledgeable about veterinary. Main categories (i.e., *WikiDrug*, *WikiBlood*, *WikiEpi* and *WikiPath*) pertain to the main index pane (i.e., "*look here*" icon ✎). WikiVet aims to create a veterinary curriculum, e.g., viruses, drugs (i.e., *glossary*), patents, sponsors (i.e., *content*). Some content has a common structure (i.e., *guidelines*) e.g., both *Antibiotics* and *Steroids* follow the *DrugT* template.

3.8 Conclusions

This chapter introduces the notion of "Wiki Scaffolding" as a means for corporate strategies to permeate wiki construction. "Wiki Scaffolding" is realized as mind map drawing to preserve wikis' openness. The result

[18] `http://en.wikivet.net/Veterinary_Education_Online` (accessed December 2012).

is WSL, a visual DSL on top of FreeMind. By taping into FreeMind as the conduit for the WSL concrete syntax, we expect non-technical communities to benefit from "Wiki Scaffolding". Potential benefits include facilitating the alignment of the wiki with organizational practices, promoting management engagement, enhancing the visibility of the wiki's practices, or promoting employee participation through direction setting.

WSL constructs are based on a literature survey about the use of wikis in companies. However, the use of corporate wikis is at its inception. It is likely that social conventions and incentives will emerge and evolve to guide contributors, resolve disputes and help manage wikis. As these issues find support in wiki engines, WSL constructs will need to be extended. In addition, we have so far focused on the feasibility of the approach and its interest in different scenarios. Additional evidences are needed to claim that "Wiki Scaffolding" succeed on better aligning wikis to corporate strategies as well as engaging users through direction setting. We plan to deploy WSL in organizations that have already been exposed to wikis to collect evidences about the advantages brought by the scaffolding. In so doing, we hope to introduce scaffolding as another step in the wiki ideal of removing "accidental complexity" from technology, and letting ordinary users directly manage and construct their own knowledge.

The focus of this work relied on the challenges during wiki initialization, now the next step is to face the challenges that happen in an evolving wiki. The following chapter delves into the details.

Parts of this chapter have already been published:

- Oscar Díaz, Gorka Puente. "Wiki Scaffolding: Aligning Wikis with the Corporate Strategy". In *Information Systems* journal, 2012. **JCR**, Impact factor 1.595.

- Oscar Díaz, Gorka Puente. "A DSL for Corporate Wiki Initialization". In *23rd International Conference on Advanced Information Systems Engineering* (*CAiSE'11*), London, UK, 2011. Acceptance rate 13%. **Best paper award**.

- Oscar Díaz, Gorka Puente. “Wiki Scaffolding: Helping Organizations to Set Up Wikis”. In *7th International Symposium on Wikis and Open Collaboration* (*WikiSym'11*), Mountain View, California, USA, 2011. Acceptance rate 42%.

Chapter 4

Wiki Refactoring through Mind Map Manipulation[1]

> "Basically, I'm not interested in doing research and I never have been. I'm interested in understanding, which is quite a different thing."
>
> – *David Blackwell.*

4.1 Overview

The organization of wikis tends to deteriorate as time goes by. Rearranging categories, constructing articles and even, moving article sections (i.e., wiki refactoring) are cumbersome tasks, which discourage the layman. But, it is the layman (knowledge workers and often not *tech-savvy* people) who writes the articles, knows the wiki content, and detects refactoring opportunities. The goal of this work is to empower this layman with refactoring capabilities.

This chapter addresses the following research question: *how to improve the refactoring affordances of current wiki engines.* Affordance is a

[1] Parts of this chapter have been previously presented [PD12, DPA11, PDA13].

perceived opportunity for action. For doing so, this work aims at improving refactoring affordance of these laymen by (*i*) abstracting from the current low-level wiki interactions into domain-specific constructs, and (*ii*), providing tools tuned to both the refactoring endeavour and the target audience (i.e., knowledge workers). To this end, this chapter introduces *WikiWhirl* as a DSL (Sections 4.5 and 4.6). WikiWhirl models wikis as mind maps, and refactoring operations as mind map manipulations. Some refactoring scenarios (Section 4.2) ground the specifics of refactoring when in a wiki setting: the wiki structure, the refactoring operations, the invariants and the refactoring notices (Section 4.3). Next, a study shows how these operations are supported in MediaWiki as to evidencing the limitations of current wiki engines (Section 4.4). Results from a controlled experiment suggest that WikiWhirl outperforms traditional wiki front-ends in three main affordance enablers: global understandability, productivity and automatic compliance of refactoring good practices (Section 4.8). Finally, conclusions (Section 4.11) end the chapter.

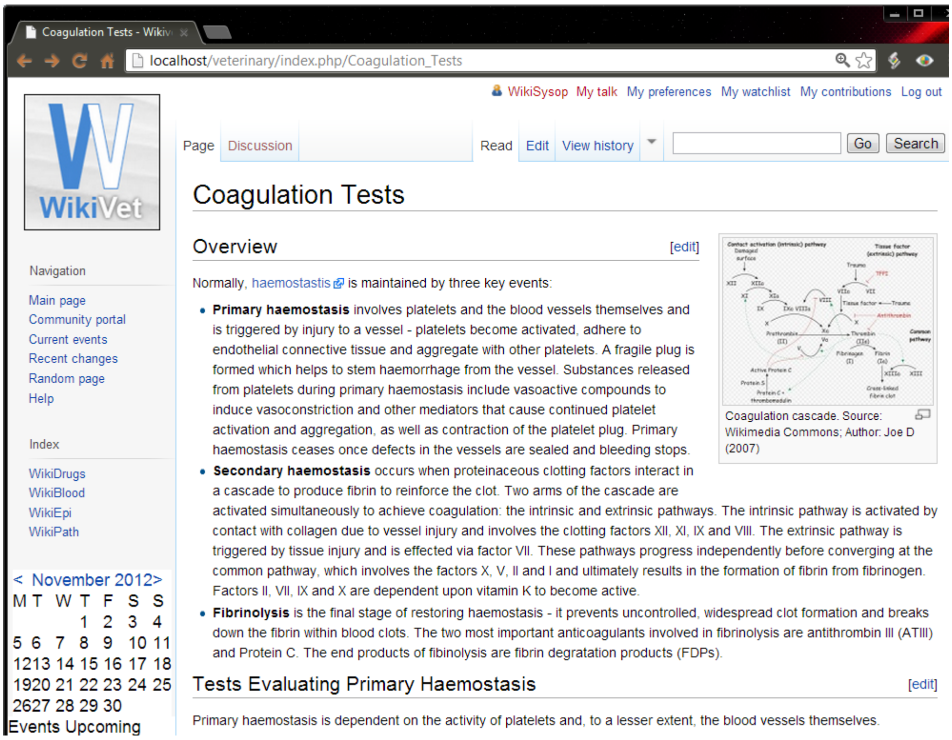
WikiWhirl does not achieve anything that cannot be obtained by directly interacting through the MediaWiki front-end. The difference stems from refactoring affordance (who can understand/do the refactoring?), productivity (how long does it take?), and consistency (i.e., whether refactoring is conducted in the same way, following good practices, no matter the user). WikiWhirl is available to download at `www.onekin.org/wikiwhirl` and the source code at `https://sourceforge.net/projects/wikiwhirl`.

4.2 Motivating Scenarios

WikiVet[2] [BQBB10] is a wiki for the veterinary domain. Figure 4.1 shows the traditional MediaWiki view of the article *Coagulation Tests*[3]. This view

[2] `http://en.wikivet.net/Veterinary_Education_Online` (accessed December 2012).

[3] We have worked locally with a reproduction of this wiki.



Coagulation Tests

Overview

Normally, haemostastis is maintained by three key events:

- **Primary haemostasis** involves platelets and the blood vessels themselves and is triggered by injury to a vessel - platelets become activated, adhere to endothelial connective tissue and aggregate with other platelets. A fragile plug is formed which helps to stem haemorrhage from the vessel. Substances released from platelets during primary haemostasis include vasoactive compounds to induce vasoconstriction and other mediators that cause continued platelet activation and aggregation, as well as contraction of the platelet plug. Primary haemostasis ceases once defects in the vessels are sealed and bleeding stops.
- **Secondary haemostasis** occurs when proteinaceous clotting factors interact in a cascade to produce fibrin to reinforce the clot. Two arms of the cascade are activated simultaneously to achieve coagulation: the intrinsic and extrinsic pathways. The intrinsic pathway is activated by contact with collagen due to vessel injury and involves the clotting factors XII, XI, IX and VIII. The extrinsic pathway is triggered by tissue injury and is effected via factor VII. These pathways progress independently before converging at the common pathway, which involves the factors X, V, II and I and ultimately results in the formation of fibrin from fibrinogen. Factors II, VII, IX and X are dependent upon vitamin K to become active.
- **Fibrinolysis** is the final stage of restoring haemostasis - it prevents uncontrolled, widespread clot formation and breaks down the fibrin within blood clots. The two most important anticoagulants involved in fibrinolysis are antithrombin III (ATIII) and Protein C. The end products of fibinolysis are fibrin degratation products (FDPs).

Coagulation cascade. Source: Wikimedia Commons; Author: Joe D (2007)

Tests Evaluating Primary Haemostasis

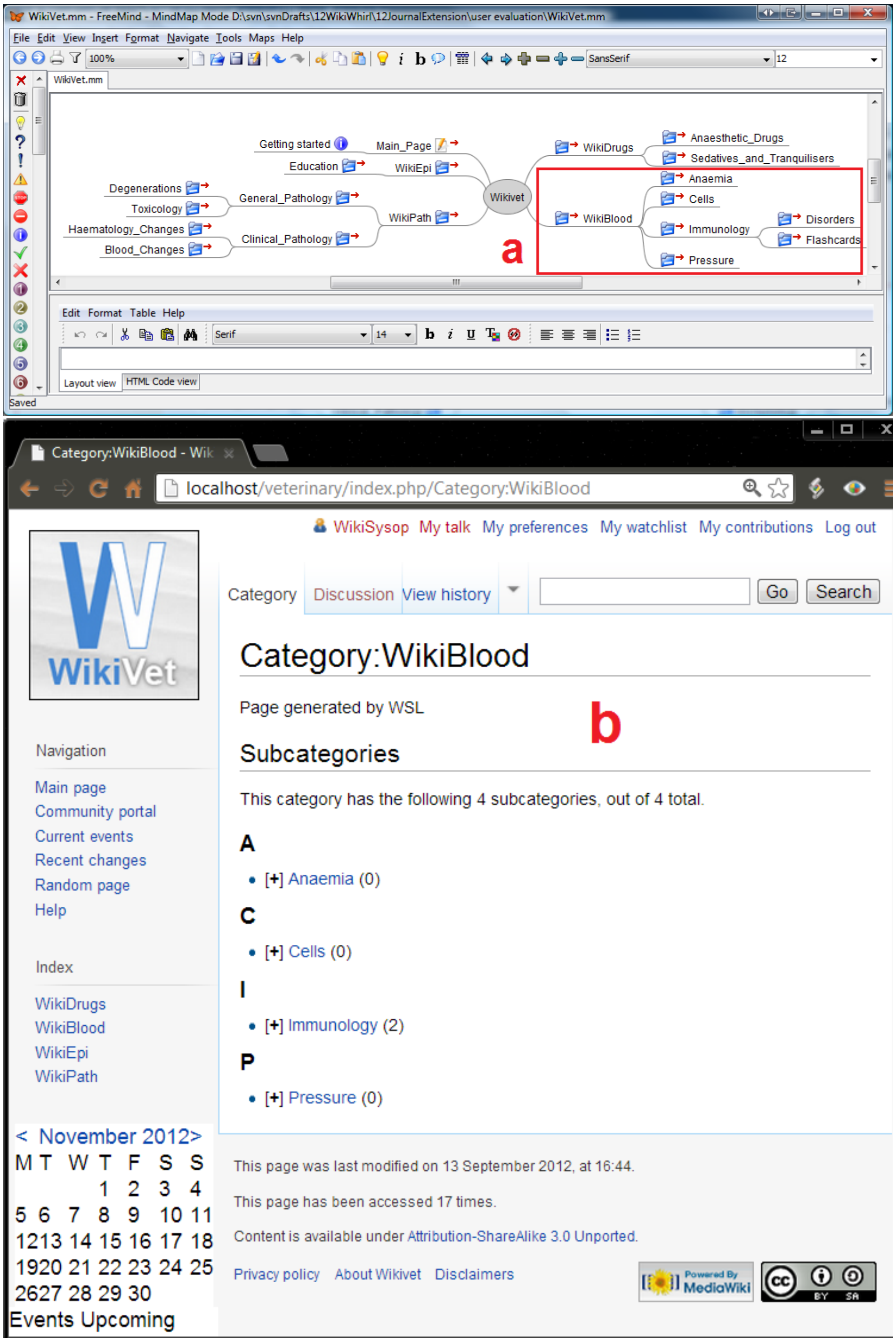
Primary haemostasis is dependent on the activity of platelets and, to a lesser extent, the blood vessels themselves.

Figure 4.1: MediaWiki view of the article *Coagulation Tests*.

is article-centric. However, “Wiki Refactoring” is about the structure of the wiki, i.e., how content is scattered among the hierarchy of categories and articles. This structure is not apparent in Figure 4.1. You can dig into the article content, looking for links to related articles, or move to the bottom of the page to see how this article is categorized. But again, these insights are specific to the article at hand. If a global view about WikiVet content is sought, users are required to navigate (either by hyperlink or tabbing navigation) through different pages for making the “deep structure” emerge into their minds. This deep structure is the subject matter of refactoring.

WikiWhirl moves this structure at the forefront by providing an alternative way of interacting with the wiki. Figure 4.2a depicts the deep structure for WikiVet (partial view), automatically obtained from the wiki database.

Broadly, pages are turned into nodes while hyperlinks become edges.



Category:WikiBlood

Page generated by WSL

Subcategories

This category has the following 4 subcategories, out of 4 total.

A

- [+] Anaemia (0)

C

- [+] Cells (0)

I

- [+] Immunology (2)

P

- [+] Pressure (0)

This page was last modified on 13 September 2012, at 16:44.

This page has been accessed 17 times.

Content is available under Attribution-ShareAlike 3.0 Unported.

Figure 4.2: (a) The WikiWhirl view: the *WikiBlood* category is depicted as a tree where all descendents are visible, and (b) the MediaWiki view: the *WikiBlood* category is rendered as a page where only immediate descendents are visible.

The icon of each node denotes its role: the "*info*" icon ⓘ stands for article sections, the "*edit*" icon ✎ denotes articles, and the "*folder*" icon 📁 represents categories. Clicking on any of the nodes moves the user to the traditional wiki front-end for the page at hand (i.e., the rendering of Figure 4.1). On top of this structure, WikiWhirl introduces a set of refactoring operations. Next, we introduce three scenarios to highlight the importance of this alternative representation: wiki initialization, structure refactoring and content refactoring. As we go along, we characterize the refactoring endeavour.

4.2.1 Wiki Initialization

Wiki stakeholders should envision a starting point in order to promote initial participation. The paralysis of facing an empty wiki and the lack of explicit statements about the wiki's purpose might prevent grassroot initiatives from "getting off" the ground [LDP^{+}11]. This suggests the convenience of a "wiki scaffolding", i.e., a preliminary category structure that serves to categorize the wiki's content at the onset [DP12] (Chapter 3). Wiki users are able to start by brainstorming about this initial structure. For the sample domain (i.e., veterinary), these preliminary categories might include:

- a category *WikiDrugs* with subcategories *Anaesthetic_Drugs* and *Sedatives_and_Tranquilisers*,
- a category *WikiBlood* with subcategories *Anaemia, Cells*, *Immunology* and *Pressure*; the subcategory *Immunology* has two subcategories *Disorders* and *Flashcards*,
- a category *WikiEpi* with a subcategory *Education*,
- a category *WikiPath* with subcategories *General_Pathology* and *Clinical_Pathology*; *General_Pathology* has two subcategories *Degenerations* and *Toxicology*; *Clinical_Pathology* has two subcategories *Haematology_Changes* and *Blood_Changes*.

Figure 4.2b shows the category *WikiBlood* of WikiVet using the MediaWiki front-end while Figure 4.2a shows the WikiVet counterpart in WikiWhirl. For an experienced user, creating these categories directly through MediaWiki took 7 minutes and 30 seconds. The very same user using WikiWhirl achieved the same result in 5 minutes and 2 seconds. As these figures point out, the difference is not so much about time. After all, there is almost a one-to-one mapping between the WikiWhirl "create category" operation and its counterpart interaction in MediaWiki. For wiki initialization, the real breakthrough comes from visual clarity, and its impact on helping consensus.

The notion of consensus sits at the very core of the wiki movement. Consensus is reached when all members of a team are willing to support a decision, even though a particular decision may not reflect an individual's choice of action. Consensus benefits from a clear understanding of each other's opinions and potential points of friction. Refactoring decisions are not an exception. The wiki content and structure are the result of a consensus among the stakeholders that emerge as different perspectives are considered, and the community becomes more acquaintance with the wiki corpus. A clear representation of the problem space is then a main enabler of consensus. Different studies report on the benefits of using mind maps to structure matters during brainstorming [BG10]. Wikis can somehow be regarded as accommodating online and asynchronous brainstorming sessions in the sense that participants are also encouraged to contribute with their own view and background.

By using mind maps, WikiWhirl permits stakeholders to discuss and try different preliminary structures for their wiki. As noted in [McH12],

> "*if a wiki has too loose of a structure, users will not understand or be motivated to contribute. On the other hand, if the structure of categories is too rigid, users may find the wiki does not meet their expectations for content collaboration.*"

This first structure will start an organic growth, as the next scenario shows.

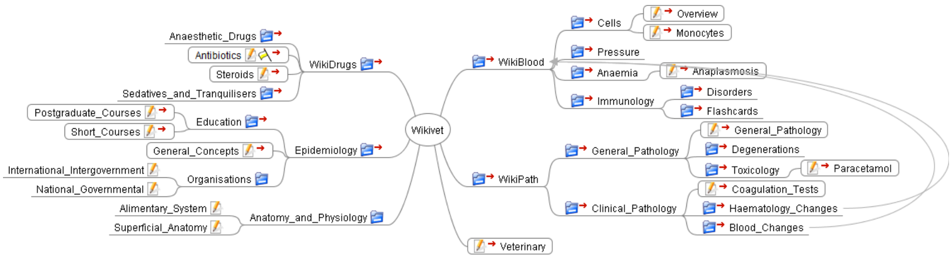


Figure 4.3: Structure refactoring: constructuing the corpus of the wiki.

4.2.2 Structure Refactoring

As time goes by, participants know the wiki's topics, and diverse perspectives begin to emerge. This growth pattern is characterized as "organic" in the sense that it is auto-regulated by the community itself [Cun06]. This results in an increase in the number and size of the articles. Furthermore, the structure expands as new categories are introduced to realize complementary, overlapping or divergent classification criteria for article organization. Focusing on structure refactoring, issues about the "conceptual consistency" and "syntactic consistency" might arise (a.k.a. "harvesting wiki consensus" in [HSB07]). How to reach such consensus is outside the scope of this work.

Back to the running example, the initial wiki structure has mutated to the one in Figure 4.3, after conducting the following operations:

- Create
  - a category *Organisations* with two sub-articles *International_Intergovernmental* and *National_Governmental*,
  - a category *Anatomy_and_Physiology* with two sub-articles *Alimentary_System* and *Superficial_Anatomies*.
- Categorize
  - the category *Organisations* with the category *WikiEpi*,

    - the category *Infectious_Agents* with the category *Anatomy_and_Physiology*.

- Uncategorize
    - the article *Paracetamol* from the category *WikiDrugs* and categorize it with the category *Toxicology*,
    - the article *Anaplasmosis* from the category *Degenerations* and categorize it with the category *Anaemia*.
- Rename
    - the article *Superficial_Anatomies* to *Superficial_Anatomy*,
    - the category *WikiEpi* to *Epidemiology*.
- Drop the category *Infectious_Agents* and all its descendants
    - the articles *Parasites* and *Viruses*,
    - the category *Bacteria* and its subcategory *Mycoplasmas*,
    - the category *Fungi* and its subcategory *Mycoses,*
- Annotate the category *Antibiotics* as "*catdiffuse*" (denoted with the yellow-flag icon). Annotations serve users to guide other fellows on how to accomplish future extensions of the category at hand (see next section).

For an expert user, these operations took 6 minutes and 4 seconds if achieved directly through the MediaWiki front-end. This figure dropped to 3 minutes and 1 second in case of using WikiWhirl. Even though productivity gains are obtained, these figures focus on *doing* refactoring but not on *conceiving* refactoring (i.e., understanding what to do). Knowledge refactoring is, above all, a matter of apprehending the subtleties of knowledge construction. From this perspective, mind maps offer a more intuitive representation than the mere listing of the wiki categories (see Figure 4.2b). Indeed, when comparing maps versus hyperlinks (i.e., the traditional MediaWiki front-end), evidences are reported about subjects who received the map had significantly less feeling of disorientation and more perception of the content structure than those in the hyperlink group [SK06]. These references ground the use of mind maps as an appropriate representation not only for *doing* but also *understanding* how to conduct "Wiki Refactoring". But refactoring in wikis is not only limited to categories and article links. Refactoring also impacts on how content is arranged as sections within articles.

4.2.3 Content Refactoring

Content refactoring might be due to the tangling and scattering of topics among distinct articles (rather than a one-to-one relationship between topics and articles). This is operationalized as splits and merges upon articles. In addition, articles' names (a.k.a. topics or titles) can also be subject to mutation as terminology agreements are reached. Finally, additional cross-referencing links might need to be created. We illustrate this scenario by conducting the following operations in our running wiki:

- Split
  - the article *Steroids* to a new article *Non_Steroids*; move the sections *Mechanism_of_Action* and *Actions* of the article *Steroids* to the article *Non_Steroids*,
  - the article *General_Concepts* to a new article *Specific_Concepts*. Move the section *Quantitative_dat*a of the article *General_Concepts* to the article *Specific_Concepts*.
- Merge
  - the articles *Postgraduate_Courses* and *Short_Courses* to a new article *Courses*,

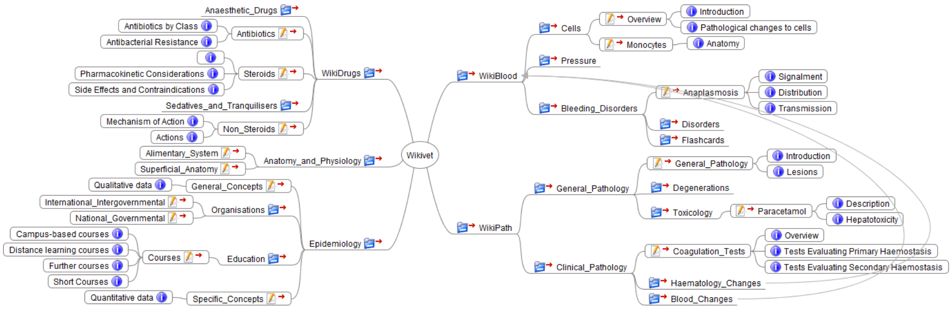


Figure 4.4: Content refactoring: rearranging content along the wiki .

  - the categories *Anaemia* and *Immunology* to a new category *Bleeding_Disorders*.

- Move

  - the section *Anatomy* of the article *Overview* to the article *Monocytes*,
  - the section *Pathological_changes_to_cells* of the article *General_Pathology* to the article *Overview*.

Figure 4.4 depicts the outcome. Time wise, previous operations took 14 minutes 13 seconds when conducted directly through the MediaWiki front-end. This figure drops to 2 minutes 39 seconds when using WikiWhirl. As expected, the more abstract the operations, the wider the time gaps. Splits and merges involve a set of low-level interactions over the MediaWiki front-end. What is a single *drag-and-drop* of a node in the mind map involves the edition of different pages in MediaWiki. In addition, the piecemeal fashion in which these operations are conducted in MediaWiki raises two main concerns: affordance and consistency. The former has to do with the number of artefacts, guidelines and interactions the user has to be aware of to refactor, keeping in mind that complexity undermines “openness” (i.e., who can conduct refactoring). On the other hand, consistency refers to whether the question “how is refactoring conducted” gets the same answer to matter the user. Next section attempts to provide common understanding about “Wiki Refactoring”.

4.3 Understanding Wiki Refactoring

Refactoring acts upon a corpus. The nature of this corpus impacts how refactoring is conducted. Hence, before introducing the refactoring process and the operations, next section outlines the nature of the wiki corpus.

4.3.1 The Wiki Corpus

A corpus (a.k.a. body of knowledge) is “a collection of all the available knowledge on a topic, or all the published material on a subject”. The scope of this topic depends on the wiki at hand. It can range from all-human-knowledge (e.g., Wikipedia) to data about the *Genome* project [CL12] or management of software projects [ADM09].

Wikis support the development of this corpus, i.e., the lifecycle of knowledge construction from embryonic ideas to well-structured comprehensive documentation where distinct articles might be in different stages of their lifecycle. Even though wikis can keep finalized articles, their added-value rests on the other stages of the article lifecycle: collaborative knowledge formation. This implies that the wiki’s structure itself is “work in progress”. We can start with a stub structure (i.e., a scaffold), which gradually evolves as the understanding of the domain growths. Hence,

> “*the purpose of the wiki’s structure is not to build a taxonomy of the world’s knowledge, but to help users to locate and make the wiki’s content grow.*”

Structure is realized as *wiki categories*, basically tags in the sense that the reason of their use must be evident from the text of the categorized article[4].

[4] The term ’tag’ in MediaWiki is used to denote markup tags for extensions `www.mediawiki.org/wiki/Manual:Tag_extensions` (accessed December 2012).

As stated in Wikipedia,

> "*these tags create links at the bottom of the page that take you to the list of all pages in that category, which makes it easy to browse related articles*[5]*.*"

This refers to the role of categories as browsable spaces in which the wiki's corpus can be partitioned and be dynamically re-arranged just by changing the category. These browsable spaces can be disposed along membership relationships so that if logical membership of category C1 implies logical membership of category C2, then C1 should be made a subcategory of C2 (or C2 is the parent category of C1). Such arrangement is referred to as *category hierarchies,* and they constitute the cornerstone of "Wiki Refactoring".

> "*Even though category hierarchies can be regarded as 'embryonic taxonomies', they lack any formal semantics. They limit themselves to facilitate location of articles through tag navigation. No inference power is built in.*"

4.3.2 Refactoring Operations

Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behaviour [Fow99]. This definition raises two questions: (*i*) how wikis can be restructured, and (*ii*) what this "external behaviour" is i.e., the invariant to be kept during "Wiki Refactoring".

The Operations

There are three main reasons to include a refactoring operation[6]. First, the operation is natively supported by most wiki engines. This includes:

[5] `www.mediawiki.org/wiki/Help:Categories` (accessed December 2012).

[6] When describing operations, we use "page" to refer to either an article or a category.

- *create* i.e., the introduction of a new wiki page that plays the role of either an article or a category,

- *categorize* i.e., characterizing the content of the page through a tag (i.e., category),

- *uncategorize* i.e., removing a tag (i.e., category) from a page,

- *article rename* i.e., changing the article's title, which is used to singularized this article; it impacts the article's URL,

- *drop* i.e., removing a page from the wiki.

Second, the operation as such is not supported by the wiki engine but is documented as part of Wikipedia's good practices:

- *Category rename*. Unlike articles, it is not possible to rename a category in MediaWiki[7]. It is necessary to create a new category and change the category tag manually on every page, and redirect the old to the new category.

- *Split.* Split is a refactoring process documented by Wikipedia whereby part of the content of a page is migrated to a new page. The two main reasons for split are size and content relevance[8]. If either the page becomes too large or its content seems to diverge between different purposes, then it is considered a split. From an authorship perspective, it is a requirement of Wikipedia licensing that attribution is given to the original author(s), and deletion of that content should be avoided.

- *Merge.* Merge is a refactoring process documented by Wikipedia whereby the content of two pages is collapsed into a new

[7] `www.mediawiki.org/wiki/Help:Categories` (accessed December 2012).

[8] `http://en.wikipedia.org/wiki/Wikipedia:Splitting` (accessed December 2012).

one. Rationales include "the unnecessary duplication of content, significant overlap with the topic of another article, and minimal content that could be covered in or requires the context of a page on a broader topic"[9].

Third, the operation is introduced based on our own experience:

- *Section move*. At least during the inception, wikis are in constant evolution. Articles are created, merged, split or deleted as the community gains insights. During this process, we found that articles might be a too coarse-grained unit of rearrangement but rather, sections might better fit as the unit of exchange.

The Invariants

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. Seeking a database parallelism, logical independence is a solid principle of database operation whereby changes in the database schema should minimally disturb client applications. This notion of logical independence rises from any *shared* 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, the aforementioned refactoring invariant is realized through two independence principles, namely:

- *Readership independence*: *readers should be able to reach the same content after or before the refactoring*. This principle preserves

[9] `http://en.wikipedia.org/wiki/Wikipedia:Merging` (accessed December 2012).

the content but not where the content is placed. Refactoring can rearrange the very same content along a different set of articles and categories. Such rearrangement should be traceable so that users can easily find the new location, and potentially, reverse malicious refactorings[10]. In addition, since content might be subject to bookmarking, readership independence also preserves URL addresses upon changes on the articles' title.

- *Authorship independence*: *authors should keep their attribution no matter where the content is finally placed.* Acknowledging the authorship has been reported as a main motivator of contributions [ASR$^+$10]. It is also one of the Wikipedia's good practices. Wiki refactoring must preserve authorship.

To support these principles, additional artefacts are introduced as part of the wiki realization: *Talk* artefacts (a.k.a. discussion pages) and *Revision* artefacts. The former hold discussions about the content of the associated page without interfering with content editing (e.g., talk pages might be used to publicize refactoring changes on the associated articles). On the other side, *Revision* artefacts keep a trace of the most recent edits. In this way, users can monitor and review the work of other users, allowing mistake correction and vandalism fight. These pages 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. The use of these artefacts to ensure both readership independence and authorship independence is discussed in Section 4.5.

[10] From this perspective, "readership independence" sustains one of the wiki hallmarks: observability. To counteract potential misbehaviour, the community should be able to detect and reverse malicious editions. Likewise, readership independence ensures refactoring changes to be traceable, and hence reversible.

4.3.3 The Process of Wiki Refactoring: Requirements

We advocate for "Wiki Refactoring" to follow two main wiki principles (Chapter 2), namely: open and observable.

Open. It implies facilitating user 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 a DSL that help users to conduct refactoring in high-level terms. Even better, such DSL should be graphical to reduce even further the learnability cost while engaging a larger number of users in the refactoring duty.

Observable. It 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. The appropriateness of a refactoring action (e.g., splitting an article) cannot be generally set by some formal verification but validated by the community. The test for "Wiki Refactoring" is whether the community backs the change. This introduces a double communication flow between the refactoring person and the community (observability) and vice versa, from the community to the refactoring person (noticeability). The former implies to keep track of refactoring changes as well as providing pervasive peer-review mechanisms. To this end, a refactoring system should support *notices,* i.e., announcements about a certain refactoring event (e.g., *split notices*, *merge notices* or *move notices* to inform the wiki community about the namesake operations). This complements *revision notices* (a.k.a. *edit summaries*) that are kept as part of the history of the artefacts to help others understand the intention of the edit.

In this way, the community can detect and reverse malicious refactoring actions. We include here "*redirect notices*" whereby the reader is redirected to another page. This can handle alternative syntactic representations of the same topic 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. Notices are main enablers of readership and authorship independence.

Collaborative refactoring benefits from the community to inform about future refactoring actions. Notices can also be used to spot refactoring needs that should be eventually conducted by someone else. This approach is extensively used for articles by introducing boilerplate messages for various issues like copyright violation, neutrality disputes, etc., using a simple shortcut command realized as a wiki template. Some examples follow[11]:

- *Catneeded* notice. It suggests for the article to be categorized so that it can be listed with similar pages.

- *Catimprove* notice. It spotlights the need for additional or more specific categories.

- *Catdiffuse* notice. It indicates that any article added to this category should eventually be moved to the appropriate subcategories (diffuse) when sufficient information is available. If this subcategory does not exist yet, either create the subcategory or leave the article in the parent category for the time being.

Among these requirements, we regard 'openness' as a pre-requisite for the other. Spurring openness will bring observability as a trust enabler. Openness implies lowering the barriers for layman participation. Among the main enablers sits the perceived affordances of current wiki engines.

[11] The notices are mainly based on Wikipedia `http://en.wikipedia.org/wiki/Wikipedia:Template_messages` (accessed December 2012).

4.4 Perceived Affordance for Refactoring

This section analyses refactoring affordances in current wiki engines. Perceived affordance is a sensed opportunity for action [Nor02]. For example, in the context of a website, any digital native perceives that an underlined text indicates a hyperlink; and, consequently an opportunity to follow (click), and reach information that relates to the hyperlinked word or phrase.

Affordability is consubstantial to the wiki's open principle. Indeed, affordable edition is seen by many as the main breakthrough introduced by wikis. Likewise, refactoring will be affordable or will not be. However, affordance is not an absolute measure. Rather, affordances contextualize refactoring within a given representative user (i.e., the *user* dimension), framed by a certain socio-cultural environment (i.e., the *social* dimension) and conducted through a given tool (i.e., the *technical* dimension) [Vat10, MF12]. Specifically, this work takes knowledge workers as the wiki users, corporations (rather than open Internet) as the environment that hosts the wiki, and MediaWiki as the wiki engine to conduct the refactoring. Even though the analysis focuses on this wiki engine, the outcomes are generalizable to other wiki engines.

4.4.1 About the Tool: MediaWiki

As most wiki engines, MediaWiki front-end favours article centricity rather than corpus centricity. That is, the MediaWiki front-end situates the article at the centre of edition, navigation and location. Rationales might be that MediaWiki was conceived for Wikipedia, where the wiki supports an encyclopaedia-like way of reading, arranging and locating information. Traditional encyclopaedias are mainly used for pinpointing a specific topic (i.e., the biography of a given person, the description of a city), and then, moving to entries that depart from this first article of interest. The wiki twist comes from making the encyclopaedia articles openly editable, and

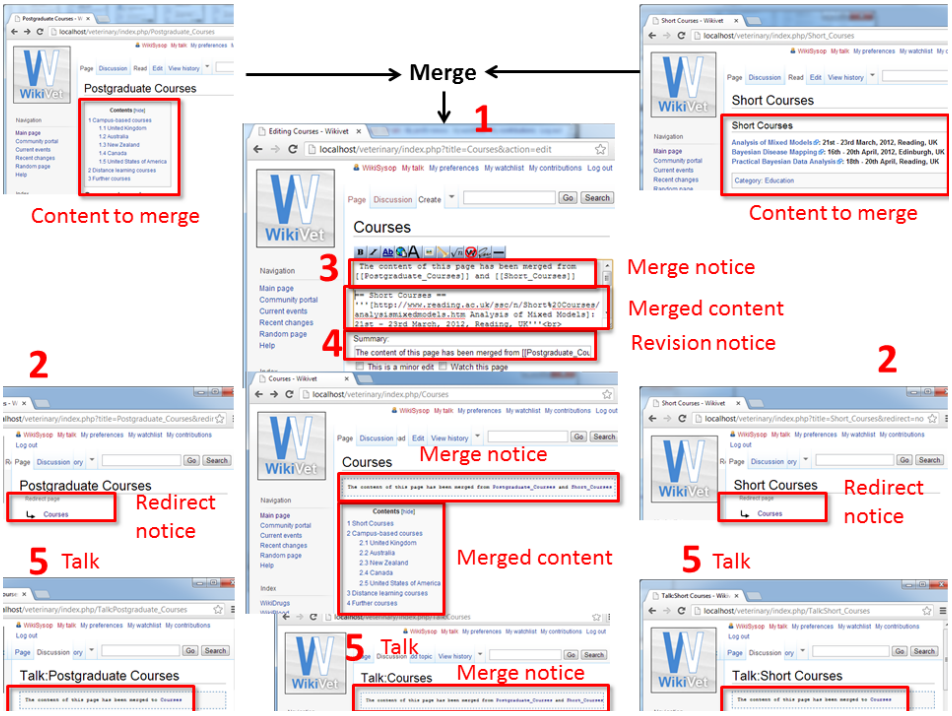
Table 4.1: Refactoring affordances for MediaWiki in terms of time (#clicks) and cognitive load (#artefacts).

Refactoring operation	**Time load**	**Cognitive load**			
	# clicks	Revision notice	Talk	Refactoring notice	Redirect notice
Create	3	✓			
Categorize	2	✓			
Uncategorize	2	✓			
Art. rename	2	✓	✓		✓
Drop	2	✓			
Cat. rename	5	✓	✓		✓
Split	6	✓	✓	✓	
Merge	9	✓	✓	✓	✓
Section move	5	✓	✓	✓	✓

offering the means for collaborative edition. This “context of use” grounds the current MediaWiki front-end. However, this work moves the focus to a different “context of use” i.e., refactoring in corporate wikis.

We quantify refactoring affordances for MediaWiki in terms of (*i*) the number of interactions needed to conduct a refactoring operation, and (*ii*) the number of different artefacts that need to be manipulated to achieve a refactoring goal. Table 4.1 shows the outcome. As an example, Figure 4.5 outlines the different interactions and notices the user has to go through to fulfil a merge operation. The user (1) starts by creating a new page (e.g., *Courses*) where to move the content of the merging articles (e.g., *Short_Courses* and *Postgraduate_Courses*). To ensure readership independence, (2) a *redirect notice* should be introduced in the merging articles. To ensure authorship independence, (3) a *merge notice* is introduced in the merged article to indicate its origin. To facilitate observability, (4) a *revision notice* (a.k.a. edit summary) is introduced and the corresponding (5) talk pages are accordingly updated for the community to be aware of the change.

The bottom line is that the current MediaWiki front-end makes refactoring convoluted. Users should not only invest time but also need to be aware of the process and notices involved. This jeopardizes affordability, and hence, openness, raising the barrier of *wiki literacy*.



The content of this page has been merged from [[Postgraduate_Courses]] and [[Short_Courses]]

Figure 4.5: Article merge through MediaWiki.

This issue is exacerbated in environments where the wiki structure either evolves frequently or the cost of disorganization is high. This can be the case of corporate wikis.

4.4.2 About the Environment: Organizations

Corporate wikis are catching on as lightweight knowledge management tools (for a survey of wikis in enterprise refer to [LDP+12]). Indeed, the Intranet 2.0 Global Survey reports that the 61% of the respondent companies (1,401 participants) were somehow using wikis [War11]. Corporate wikis tend to be smaller than open wikis. An estimate is for corporate wikis to contain an average of 1,500 articles [SB09]. This reduction in size however, hides that corporate users tend to be interested in a larger number of wiki pages than e.g., users of Wikipedia. Apart from cosmetic editing (e.g., typo corrections), wikipedians tend to contribute in few more than one article[12]. By contrast, the interest of corporate employees frequently expands along different wiki pages. This makes both clear structure and easy navigation more prominent in corporate wikis than in open wikis. Specifically, Lykourentzou et. al [LDP+12] state that "*structure seems to also play an important role to the success or failure of a corporate wiki implementation. That is, poor structural support often seems to result to laborious information insertion and retrieval, navigation difficulties and information duplication*". In a working setting, these pitfalls impact the productivity of knowledge workers.

4.4.3 About the User: Knowledge Workers

Open wikis tend to be grounded in altruistic collaboration. By contrast, contribution in corporate wikis is not always a selfless activity at spare time but part of the duties at working time. This also implies that users' time is more valuable for the organization. In this regard, *the tool's user-friendliness might improve the efficiency and efficacy of knowledge workers*. Indeed, this feature is identified as a major technological enabler of the success of a corporate wiki [LDP+12]. In addition, "*attribution of authorship seems to be the affordance that is most appropriate modification for corporate settings, and users believe that introducing an attribution mechanism will increase their involvement*" [YA12].

Figure 4.6 outlines the previous discussion along the three-dimensional affordance space. Technically, wiki engines incorporate facilities to categorize or rename as well as artefacts for hosting discussions and tracing within the wiki (e.g., talk pages). However, some refactorings expand along different interactions throughout distinct wiki pages. As a result, users conducting refactoring are expected to have additional technical skills besides article editing. Good synthesis skills as well as a good

[12] `http://www.aaronsw.com/weblog/whowriteswikipedia` (accessed December 2012).

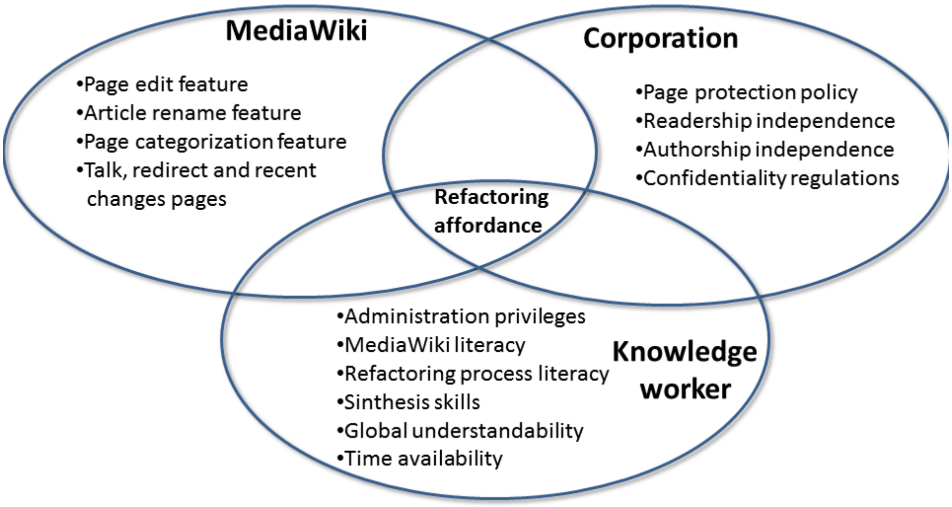


Figure 4.6: Refactoring affordances.

knowledge about the wiki corpus are also recommended. Finally, these interactions are framed by a set of regulations or good practices about preserving authorship and readership. The company's organigrams or confidentiality regulations might impose additional restrictions about who can refactor and how refactoring can be conducted. The wiki engine does not preclude some "socially" incorrect refactoring (e.g., not preserving authorship). Social surveillance might then be needed to ensure socially acceptable refactoring.

We contend that the current MediaWiki front-end requires users to be literate about the intricacies of refactoring while leaving to the community the detection of socially-disturbing refactoring practices. Leveraging MediaWiki's refactoring affordances may be conceived as moving some duties from either the user-affordance dimension or the social-affordance dimension to the technical dimension (see Figure 4.6). We focus on the user-affordance dimension by moving refactoring expertise from the user to the refactoring tool. Next, we introduce a DSL for "Wiki Refactoring": WikiWhirl. The construction of this DSL entails systematizing a set of refactoring operations (i.e., the abstract syntax) (Section 4.5), nailing these operations down into a concrete syntax (Section 4.6), and providing tool

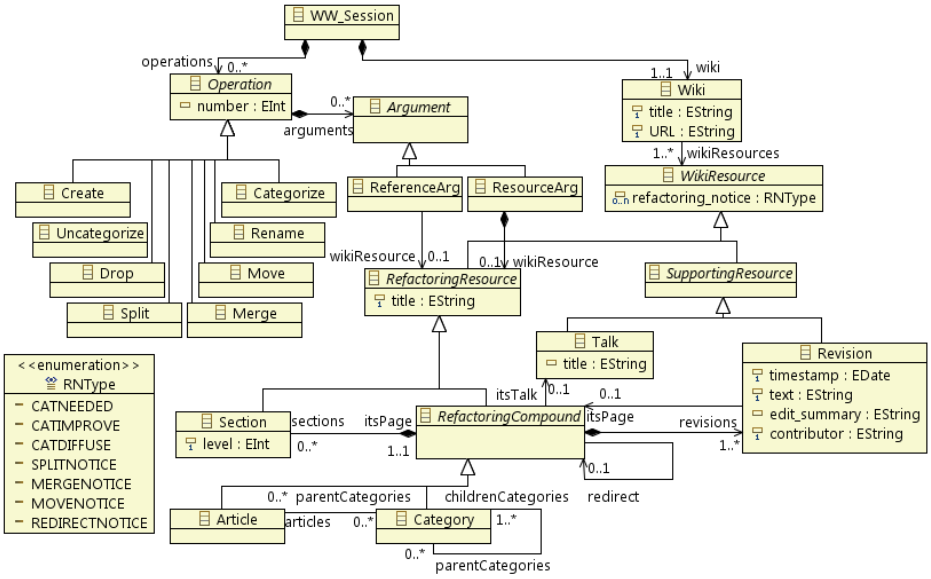


Figure 4.7: WikiWhirl metamodel (abstract syntax).

support (Section 4.7) [MHS05]. Only the final step depends on the wiki engine.

4.5 WikiWhirl: The Abstract Syntax

As previously described, 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 [Val10]. This is expressed as the DSL metamodel. We introduce the WikiWhirl DSL as the means to specify a refactoring session (see Figure 4.7).

A WikiWhirl session is captured through a *Wiki* model plus as a sequence of refactoring *Operations*. The *Wiki* model conceives the wiki as a composition of *WikiResources*, which are characterised by *refactoring_notices*. *Refactoring_notices* are boilerplate messages that spot some refactoring matter for the resource at hand. Subsection 4.3.3

introduced six such notices.

Resources are classified as *RefactoringResources* (i.e., the subject matter of the refactoring operations) and *SupportingResources* (i.e., those needed to support the authorship and readership independence principles). Refactoring resources include *Categories*, *Articles* and *Sections*, where *Categories* and *Articles* are composed by *Sections*[13]. Notice that wiki engines do not support sections as independent artefacts but as embedded inside the wikitext of pages. However, this approach promotes sections as full-fledged classes as representatives of the article structure. As for the *SupportingResources*, they include *Talks* and *Revisions*. On the other hand, a refactoring operation (*Operation*) acts upon existing *RefactoringResources* (i.e., *ReferenceArg*) or creates a new *RefactoringResource* (i.e., *ResourceArg*). *RefactoringResources* have a title, which is a user-given string that singularized the resource. Operation classes are those introduced in Subsection 4.3.2.

This abstract syntax permits to describe refactoring sessions. Broadly, a textual representation of one such session would be:

Split(Steroids, 'Steroids_split');

Rename(Steroids_split, 'Non-Steroids');

Move(Mechanism_Of_Action, Steroids, Non_Steroids);

where *Steroids* and *Non_Steroids* are *Article* resources, and *Mechanism_Of_Action* is a *SectionResource.* However, the important point is not so much about the syntax but how these primitives behave. Using design-by-contract, the Table 4.2 characterizes these operations in terms of pre-conditions and post-conditions. The invariant is to keep the *Section* set inalterable: sections can be re-arranged but never deleted. Notice however that this invariant only covers parts of the independence principles for "Wiki Refactoring" as set in subsection 4.3.2, namely:

[13] Only first level sections are considered (denoted as '== *sectionName* ==' in *WikiText*).

Table 4.2: Refactoring operations and their pre/post conditions.

Operation	Pre-condition	Post-condition
create(newTitle, aResource)	newTitle $\neq$ null; Wiki.wikiResources. contains(newTitle) = false	Wiki.wikiResources. contains(newTitle) = true aResource.title=newTitle
categorize(aResource, aCategory)	aResource $\in$ RefactoringCompound; aResource $\neq$ aCategory; aResource, aCategory $\neq$ null; aResource.parentCategories. contains(aCategory) = false	aResource.parentCategories. contains(aCategory) = true
uncategorize(aResource, aCategory)	aResource $\in$ RefactoringCompound; aResource, aCategory $\neq$ null; aResource.parentCategories. contains(aCategory) = true	aResource.parentCategories. contains(aCategory) = false
drop(aResource)	aResource $\in$ RefactoringCompound; aResource.sections $=$ null;	Wiki.wikiResources. contains(aResource) = false
rename(aResource, newTitle)	aResource $\in$ RefactoringCompound; aResource, newTitle $\neq$ null; Wiki.wikiResources. contains(newTitle) = false	Wiki.wikiResources. contains(newTitle) = true; aResource.title=newTitle; aResource.refactoring_notice = RENAMENOTICE; aResource. refactoring_notice = REDIRECTNOTICE;
split(aResource, newResource)	aResource $\in$ RefactoringCompound; Wiki.wikiResources. contains(newResource) = false	Wiki.wikiResources. contains(newResource) = true; aResource.refactoring_notice = SPLITNOTICE; newResource. refactoring_notice = SPLITNOTICE;
merge(aResource1, aResource2, newResource)	*resources* $\in$ RefactoringCompound; Wiki.wikiResources. contains(newResource) = false	Wiki.wikiResources. contains(newResource) = true; newResource.sections=aResource1. sections $\cup$ aResource2.sections newResource.title = aResource1.title + aResource2.title; aResource1. refactoring_notice = MERGENOTICE; aResource1. refactoring_notice = REDIRECTNOTICE; aResource2.refactoring_notice = MERGENOTICE; aResource2.refactoring_notice = REDIRECTNOTICE; newResource.refactoring_notice = MERGENOTICE;
move(aSection, sourceResource, targetResource)	*resources* $\in$ RefactoringCompound; sourceResource.sections. contains(aSection) = true; targetResource $\neq$ null;	sourceResource.sections. contains(aSection) = false; targetResource.sections. contains(aSection) = true; sourceResource.refactoring_notice = MOVENOTICE; targetResource. refactoring_notice = MOVENOTICE;

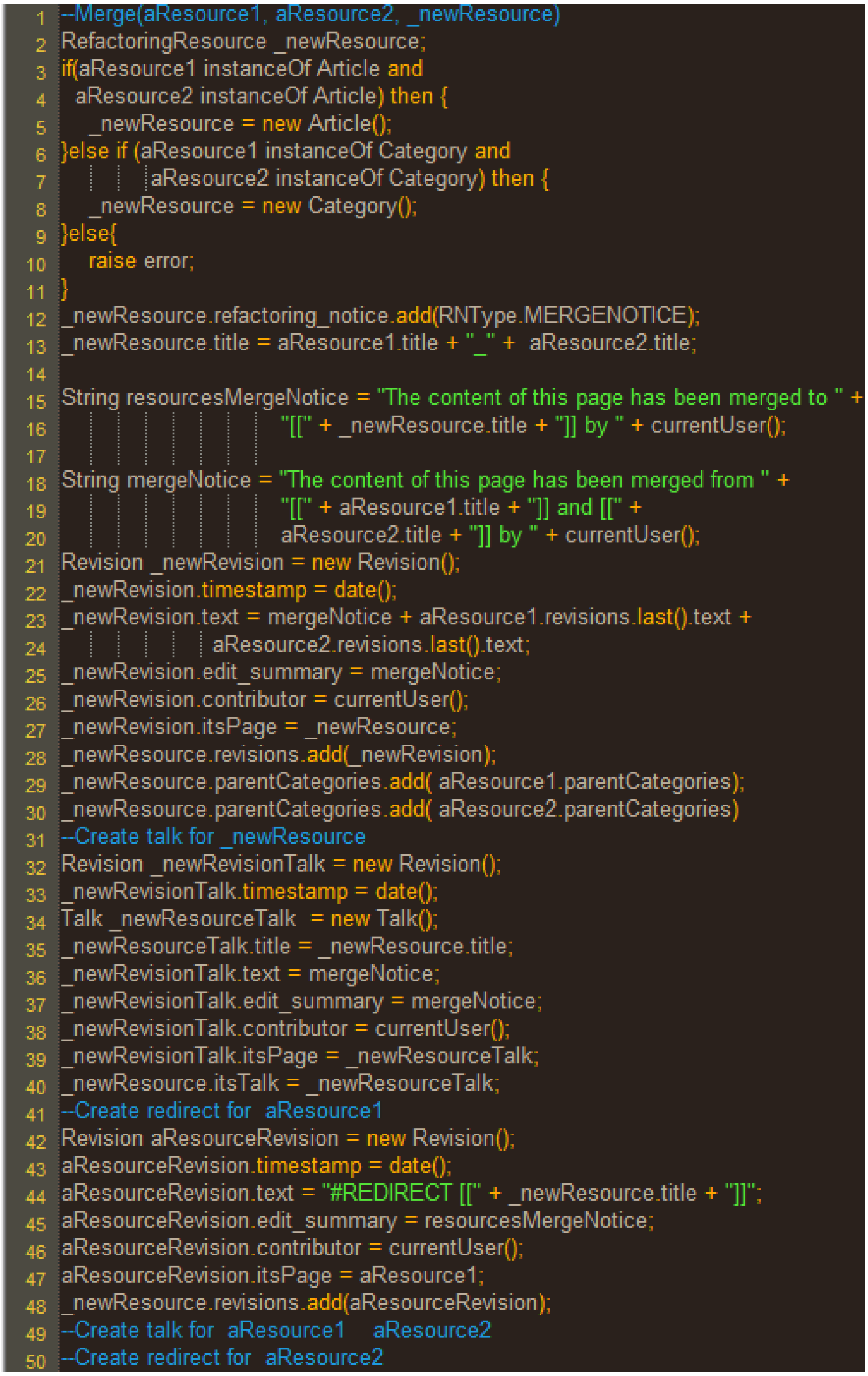
- *Readership independence*: *readers should be able to reach the same content after or before the refactoring*. This principle preserves the content but not where the content is placed. Such rearrangement should be traceable so that users can easily find the new location, and potentially, reverse malicious refactorings.

- *Authorship independence*: *authors should keep their attribution no matter where the content is finally placed.* Acknowledging the authorship has been reported as a main motivator of contributions [ASR+10]. It is also one of the Wikipedia's good practices. Wiki refactoring must preserve authorship.

Such principles are to be engineered as part of the operational semantics of the refactoring operations. Figure 4.8 provides the operational semantics for *Merge(aResource1, aResource2, _newResource)*:

- (lines 1-13) The *_newResource* is created either as an *Article* or a *Category*, whose *title* is created by concatenating the titles of the merging resources (line 13). A *refactoring_notice* with type MERGENOTICE is added (line 12).

- (lines 14-28) This *_newResource* is associated with its first revision (line 28). This implies first, create a revision (line 21), whose content is that of the merging resources plus the merge notice (lines 23-24). *Authorship independence* advises that attribution is given to the original author. This recommendation is followed by adding the *mergeNotice* that notes the origin of the resource (e.g., "*The content of this page has been merged from [[aResource1]]] and [[aResource2]]] by Admin*") (lines 18-20). Square brackets denote the URL of the article at hand. *Readership independence* suggests that when an article is merged, an *edit_summary* should be left in the *_newResource* (line 25) i.e., content that helps identify the merge operation in the page's history.



```
--Merge(aResource1, aResource2, _newResource)
RefactoringResource _newResource;
if(aResource1 instanceOf Article and
  aResource2 instanceOf Article) then {
    _newResource = new Article();
}else if (aResource1 instanceOf Category and
          aResource2 instanceOf Category) then {
    _newResource = new Category();
}else{
    raise error;
}
_newResource.refactoring_notice.add(RNType.MERGENOTICE);
_newResource.title = aResource1.title + "_" +  aResource2.title;

String resourcesMergeNotice = "The content of this page has been merged to " +
                                "[[" + _newResource.title + "]] by " + currentUser();

String mergeNotice = "The content of this page has been merged from " +
                                "[[" + aResource1.title + "]] and [[" +
                                aResource2.title + "]] by " + currentUser();
Revision _newRevision = new Revision();
_newRevision.timestamp = date();
_newRevision.text = mergeNotice + aResource1.revisions.last().text +
                      aResource2.revisions.last().text;
_newRevision.edit_summary = mergeNotice;
_newRevision.contributor = currentUser();
_newRevision.itsPage = _newResource;
_newResource.revisions.add(_newRevision);
_newResource.parentCategories.add( aResource1.parentCategories);
_newResource.parentCategories.add( aResource2.parentCategories)
--Create talk for _newResource
Revision _newRevisionTalk = new Revision();
_newRevisionTalk.timestamp = date();
Talk _newResourceTalk  = new Talk();
_newResourceTalk.title = _newResource.title;
_newRevisionTalk.text = mergeNotice;
_newRevisionTalk.edit_summary = mergeNotice;
_newRevisionTalk.contributor = currentUser();
_newRevisionTalk.itsPage = _newResourceTalk;
_newResource.itsTalk = _newResourceTalk;
--Create redirect for  aResource1
Revision aResourceRevision = new Revision();
aResourceRevision.timestamp = date();
aResourceRevision.text = "#REDIRECT [[" + _newResource.title + "]]";
aResourceRevision.edit_summary = resourcesMergeNotice;
aResourceRevision.contributor = currentUser();
aResourceRevision.itsPage = aResource1;
_newResource.revisions.add(aResourceRevision);
--Create talk for  aResource1    aResource2
--Create redirect for  aResource2
```

Figure 4.8: *Merge* operational semantics.

- (lines 29-30) The *_newResource* is categorized along the lines of the merging resources.

- (lines 31-40) Collaborative refactoring implies refactoring decisions to be publically noted. WikiWhirl notes the refactoring through talk pages associated with the resources (*aResource1*, *aResource2* and *_newResource*), which can eventually frame discussions about this refactoring change. These lines create a talk page for the *_newResouce*, and associates this talk page with a revision describing the refactoring. Likewise, similar talk pages are associated with both source resources, *aResource1*, *aResource2*.

- (lines 41-48). *Readership independence* also requires a mechanism to avoid dangling references. The source resources have to dynamically redirect readers to the new location of the content.

Following lines present a brief for the rest of the WikiWhirl operations, while omitting simple operations such as create, rename article, drop, categorize, etc.:

- *Rename* (*aCategory*, *newTitle*). To rename *aCategory*, a new category with title *newTitle* is created with the content of *aCategory*, which becomes a redirect page to the category *newTitle*. In addition, all the category tags linking to it must be renamed, to pertain now to the category *newTitle*. A talk page opens a way to discuss the new change.

- *Split (aResource, newResource)*. The title of the *newResource* results from concatenating the title of *aResource* with "*_split*" (e.g., "*Steroids_split*"). As for authorship and readership independence, a comment in the edit summary of the *newResource* must be made to note the split operation, as well as to create the corresponding discussions in the affected pages.

- *Move (aSection, sourceResource, targetResource).* Here, the same recommendations are followed that for article splitting, i.e., introducing in the associated talk pages a note such as *"Section aSection from sourceArticle was copied into newArticle at timestamp"*. In addition, the *recentChanges* page of the *sourceResource* is to include a summary, noting the origin of this article (e.g., *"Section aSection move to [[targetArticle]]"*). Likewise, the *recentChanges* page of the *targetResource* should also include the summary *"Section aSection being moved from [[sourceName]]"*).

The aforementioned metamodel constructs need a textual or visual representation counterpart. This is the aim of the concrete syntax.

4.6 WikiWhirl: The Concrete Syntax

There may be different concrete syntaxes for the same abstract syntax where each concrete syntax might stand for the entirety or a subset of the abstract syntax. Here, we focus on the most complex part: the *Wiki* construct. A *Wiki* model is a compound of *WikiResources*. *WikiResources* can be *RefactoringResources* and *SupportingResources*. The latter are automatically inferred during the refactoring process. That is, *RefactoringResources* are not the subject of refactoring but the byproduct of this process. Basically, they keep traces and notices for the wiki community to reconstruct the refactoring process. Therefore, *SupportingResources* are not explicitly handled by the user but are created or modified as a result of operating upon the *RefactoringResources*, what makes the focus on how to specify *Wiki* models that contain *RefactoringResources*. Next, it follows a brief on the use of mind maps as an appropriate graphical notation for *Wiki* models.

While the abstract syntax addresses expressiveness, the concrete syntax cares for the DSL usability and understandability. Proposed alternatives for wiki visualization are influenced by their aims: visualize author collaboration [VWD04], depict article relationships (e.g., *Annoki* [TS10]), outline the wiki structure (e.g., *WikiNavMap* [UK07]), etc. In addition, the expressiveness of these representations should be balanced against affordance for the target audience. Therefore, we need to look into (*i*) the characteristics of the object to be handled (e.g., its size), (*ii*) how this object is to be manipulated, and (*iii*) the context of use i.e., "the users, tasks, equipment, and the physical and social environments in which a product is used" [ISO].

Wikis are graphs, where nodes stand for pages and edges denote relationships between those pages. For corporate wikis, their size is estimated an average of up to 1,500 nodes [SB09]. Next, we consider manipulations 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). This way of working calls for agile visualizations that permit to define "views" over existing wiki graphs as well as to collapse or extend these views as we gain understanding about the refactoring requirements. Finally, the context of use is that of knowledge workers who access the wiki within the boundaries of an organization. On these grounds, we next advocate for the use of mind maps as the notation for WikiWhirl expressions.

Why mind maps as for the manipulations of wikis. Wiki refactoring is akin to mind mapping. Two observations should, however, be made. First, equating nodes to wiki topics should consider the 'embryonic' nature of those topics, in the sense that they are generally not stable but evolve as the understanding about the topic develops. So "wiki nodes" are not immutable, ontological notions but ideas in the process of formation. Second, wikis are graphs whereas mind maps tend to have a radial, hierarchical disposition of nodes. However, this radial disposition of nodes facilitates 'the drilling down' and 'the rolling up' along the different radius of the map.

Why mind maps as for the target audience. Mind mapping is catching on within organizations. A survey performed to 334 respondents about the use of mind maps showed that they are mainly used for project planning, brainstorming, knowledge and project management or to-do lists [Fre10]. Interestingly enough, 21% of the respondents apply mind maps as a blueprint for the development of websites, i.e., to outline the website structure. Besides that, users perceive many benefits like the improvement of clarity of thinking, management information overload, visualizing information relationships or organizing better. An evidence of this popularity is the number of tools for mind mapping[14].

Based on these observations, we select mind maps as the notation for *Wiki* models.

4.6.1 Wiki Models as Mind Maps

The different elements that conform a mind map are explained in the Chapter 3, Section 3.4.2. Next, the Table 4.3 shows the mapping of these elements (Figure 4.9) to the Wiki constructs in Figure 4.7. *WikiResources* are classified as *Categories*, *Articles* and *Sections*. This categorization is denoted through icons: the "*category*" icon , the "*article*" icon and the "*section*" icon, respectively. Next, we describe relationships. Since some relationships are M:N (e.g., parent categories), 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*. Figure 4.4 illustrates this situation. Category *Blood_Changes* belongs to two categories; the relationship with *Clinical_Pathology* is denoted by an *edge* while that of *WikiBlood* is depicted as an *arrowlink*.

[14] For a comparison refer to `www.collegedegree.com/library/college-life/99-mind-mapping` and en.wikipedia.org/wiki/List_of_concept-_and_mind-mapping_software (accessed December 2012).

Table 4.3: From abstract constructs to their realization as mind map primitives.

WikiWhirl primitives	**Mind map** primitives	**FreeMind** primitives
"*Wiki*" class	Root node	Root node
"*title*" property	node title: Text	node title: Text
"*URL*" property	Link	Link
"*Category*" class	Node with a "*category*" icon	Node with icon 📁
references:		
first "*parentCategories*"	edge	edge
rest "*parentCategories*"	arrowlink	arrowlink
"*Article*" class	Node with an "*article*" icon	Node with icon 📝
references:		
first "*parentCategories*"	edge	edge
rest "*parentCategories*"	arrowlink	arrowlink
"*Section*" class	Node with a "*section*" icon	Node with icon ⓘ
"*itsPage*" reference	edge	edge
"*catneeded*" notice	Node with "*catneeded*" icon	Node with "*flag-red*" icon 🚩
"*catimprove*" notice	Node with "*catimprove*" icon	Node with "*flag-blue*" icon ⚑
"*catdiffuse*" notice	Node with "*catdiffuse*" icon	Node with "*flag-yellow*" icon ⚑

Finally "*catneeded*", "*catimprove*" and "*catdiffuse*" *notices* are mapped to the namesake icons. The rest of the notices are not to be provided by the user but automatically generated as part of the operational semantics of the refactoring operations. Therefore, they do not have a counterpart in the concrete syntax.

4.7 WikiWhirl: Tool Support

Rather than developing our own visual editor for mind maps, we capitalize on FreeMind. As previously commented (Chapter 3), FreeMind supports

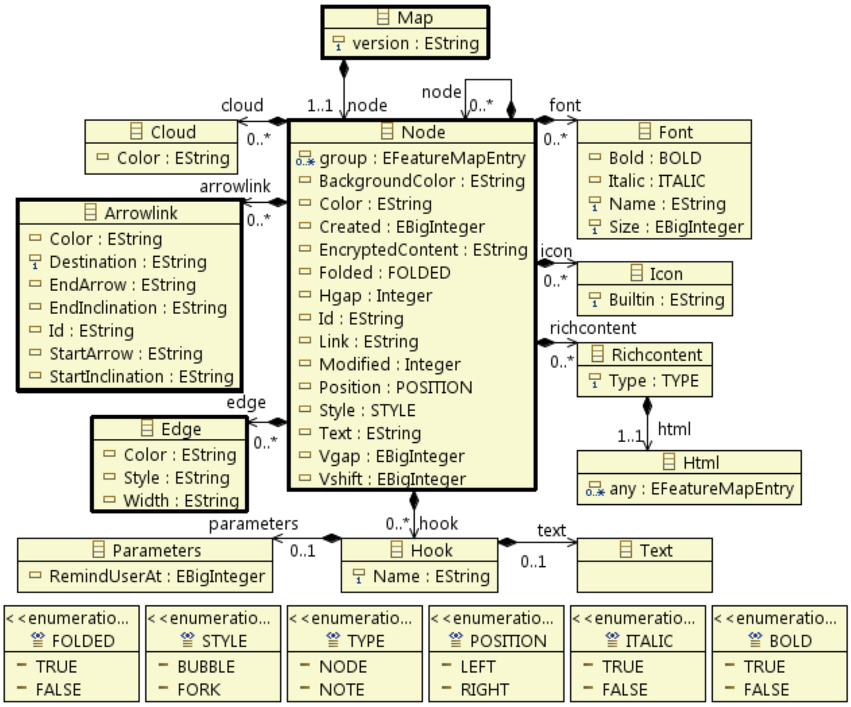


Figure 4.9: FreeMind XML Schema for map drawing (represented as a Ecore metamodel).

easy edition and visualization of mind maps (e.g., nodes are easily moved around, branches collapsed, etc.). Choosing an existing tool not only speeds up development, learnability is the big plus on the hope that the target audience (i.e., employees in corporate wikis) may already be familiar with FreeMind.

Turning FreeMind into a refactoring platform implies for FreeMind to become: (*i*) an editor of Wiki maps, (*ii*) an enactor of refactoring operations, (*iii*) an interpreter of refactoring operations, and (*iv*), a workplace for refactoring sessions.

4.7.1 FreeMind as an Editor of Wiki Maps

Section 4.6 addressed the suitability and the sufficient expressiveness of mind maps to capture *Wiki* models. Now, the focus is on a particular

Table 4.4: Mapping WikiWhirl operations to FreeMind interactions.

WikiWhirl operation	**FreeMind interaction**	**FreeMind Error message if precondition violated**
create(newTitle, Page)	*click* ’create node’	sections cannot be created; newTitle cannot be null; newTitle already exists
categorize(aResource, aCategory)	*draw* an arrowlink to a category node or *drag* a node to a category node	sections cannot be categorized; aCategory cannot categorize itself; aCategory must exist
uncategorize(aResource, aCategory)	*click* ’remove’ an arrowlink to a category node or *move* a node to the root node	sections cannot be uncategorized; aCategory must exist
drop(aResource)	*click* ’remove’ a node	sections cannot be removed; aResource has section descendants
rename(aResource, newTitle)	*edit* the text of a node	sections cannot be renamed; newTitle cannot be null; newTitle is already taken
split(aResource, newResource)	*select* node + *right-mouse click* “WikiWhirl split”	sections cannot be split; newResource cannot be null; newResource already exist
merge(aResource1, aResource2, newResource)	*select* node aResource1 + *select* node aResource2 + *right-mouse click* “WikiWhirl merge”	sections cannot be merged; newResource cannot be null; newResource already exist
move(aSection, sourceResource, targetResource)	*drag* aSection node + *drop* aSection node to targetResource node	aSection must pertain to sourceResource; targetResource must exist

implementation of mind maps. FreeMind uses *XML Schema* to specify maps. Figure 4.9 shows the metamodel counterpart of such schema (explained in Chapter 3 Section3.4.2). Table 4.3 shows the mapping with the FreeMind primitives. For instance, “*catneeded*”, “*catimprove*” and “*catdiffuse*” *notices* are mapped to the “*flag-red*” icon 🚩, “*flag-blue*” icon 🚩 and “*flag-yellow*” icon 🚩 available in FreeMind, respectively.

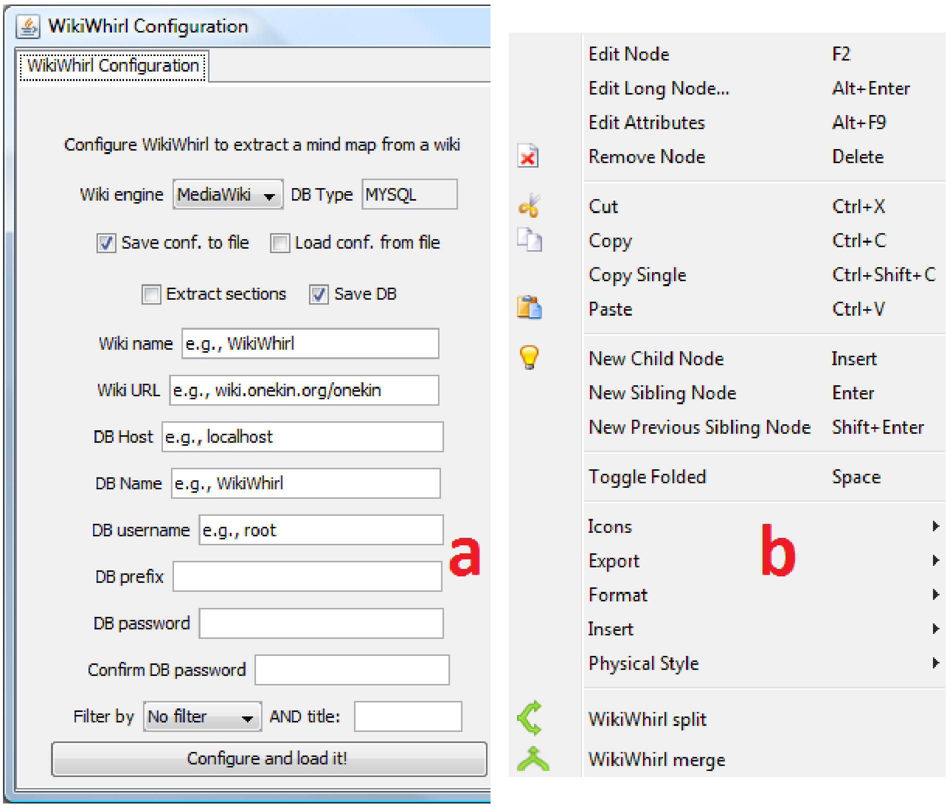
However, this mapping is not enough. A WikiWhirl’s *Wiki* model is a compliant FreeMind mind map. However, the opposite does not hold. Some FreeMind maps might not deliver a compliant *Wiki* model, where compliance is determined by the WikiWhirl’s abstract syntax. Therefore, WikiWhirl maps are a subset of the possible maps that can be drawn in FreeMind. FreeMind uses a *XML Schema* to denote what is a valid map. On top of it, a set of constraints ensures that mind maps account for compliant *Wiki* models (i.e., conform to the WikiWhirl abstract syntax).

4.7.2 FreeMind as an Enactor of Refactoring Operations

The implications of FreeMind as an enactor of refactoring operations are two-fold. First, FreeMind front-end should provide a way for the user to conduct the operation. For instance, creating a *category* implies to position the cursor in a node, right click and select “new child node”, name the newly created node and finally, add the *“folder”* icon 📁. Most of the WikiWhirl operations have a direct mapping to FreeMind interactions (e.g., *click, drag, drop, move*) (see Table 4.4). Only *Merge* and *Split*, do not have a direct counterpart. As a result, we extend the right-mouse-click menu with two options, *WikiWhirl split* and *WikiWhirl merge* (see Figure 4.10b). Once a node (or two to merge) is selected, the user can right-click on the mouse to perform the desired operation (i.e., merge or split).

However, some FreeMind interactions though possible, might be invalid from a refactoring perspective: nodes that stand for sections can only be dragged but never removed; nodes that denote articles/categories can be deleted only if they do not contain sections, etc. That is, preconditions of WikiWhirl operations should be obeyed. Basically, preconditions play a similar role to structural constraints: they restrict FreeMind to conform to the WikiWhirl semantics. This leads us to the second implication.

FreeMind acts as a requester of WikiWhirl operations on behalf of the user. Along with the design-by-contract principles, the requester should guarantee that preconditions are met before enacting the operation. That is, WikiWhirl should ensure that the user interaction complies with the refactoring operations preconditions. Figure 4.4 indicates how preconditions are turned into error messages. Figure 4.11 shows two examples where users are prevented from violating WikiWhirl semantics (even though the interactions are totally valid FreeMind interactions). The error messages describe the cause of the violation in terms of WikiWhirl semantics. As wisely noted in [BAGB11], any user interface is a realization of a language with two directions: human-computer and



WikiWhirl Configuration

Configure WikiWhirl to extract a mind map from a wiki

Wiki engine: MediaWiki DB Type: MYSQL

☑ Save conf. to file ☐ Load conf. from file

☐ Extract sections ☑ Save DB

Wiki name: e.g., WikiWhirl

Wiki URL: e.g., wiki.onekin.org/onekin

DB Host: e.g., localhost

DB Name: e.g., WikiWhirl

DB username: e.g., root

DB prefix:

DB password:

Confirm DB password:

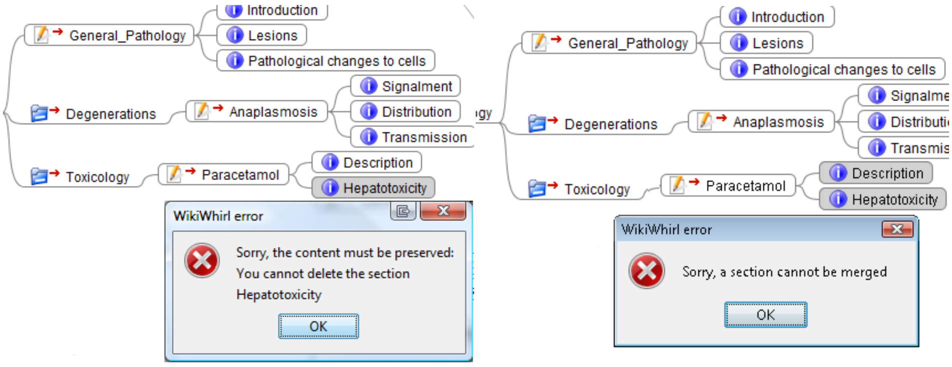
Filter by: No filter AND title:

Figure 4.10: Turning FreeMind into a "Wiki Refactoring" tool. (*i*) The configuration menu, and (*ii*) the popup menu (right mouse click).

computer-human, since the feedback from the computer needs also to be interpreted by the humans. In this way, WikiWhirl acts as a learning tool about good practices in refactoring.

4.7.3 FreeMind as an Interpreter of Refactoring Operations

FreeMind becomes an interpreter of refactoring operations. The operational semantics is now concretized for a given wiki engine. We choose MediaWiki. The constructs *Article*, *Category*, *Talk*, etc. find a specific realization in terms of the database schema of MediaWiki. Refactoring operations become transactions over the wiki database, and



Sorry, the content must be preserved: You cannot delete the section Hepatotoxicity

Sorry, a section cannot be merged

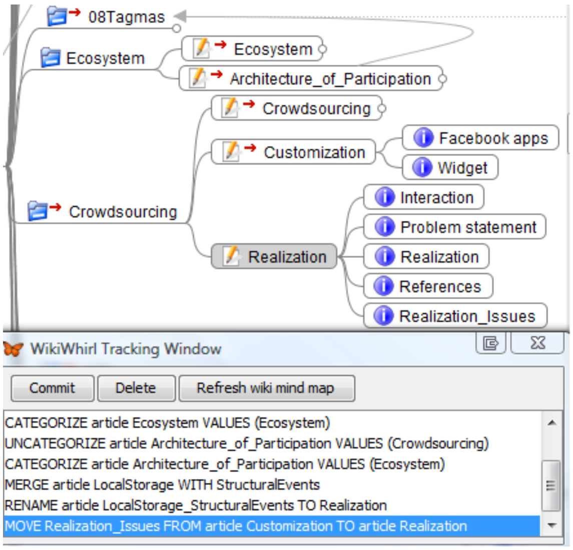
Figure 4.11: Two scenarios that raise refactoring errors: (left) removal of the section node "Hepatotoxicity" violates the pre-condition of the *drop* operation, and (right), merging two node sections violates the pre-condition of the *merge* operation.

the operational semantics is realized as an SQL script upon the MediaWiki tables. Appendix B shows the SQL script for the *merge* operation.

4.7.4 FreeMind as a Workplace for Refactoring Sessions

In this section does not look so much at the *definition* of WikiWhirl but at *using* WikiWhirl. By experiencing different refactoring scenarios, we come up with additional requirements for FreeMind to smoothly support the process of refactoring a wiki. Requirements include: (*i*) dynamic loading of *Wiki* models, (*ii*) refactoring transactions, and (*iii*) FreeMind-MediaWiki roundtrips.

Loading of Wiki models. Most scenarios do not start with an empty wiki but with a wiki that already exists. This requires an "import" utility that obtains a *Wiki* map out of an existing wiki. To this end, FreeMind is extended with an import utility that loads the wiki corpus from a MediaWiki installation. Figure 4.10a shows the configuration menu for such utility along two parameters: the *configuration service* and the *load mode*. The former indicates the database service parameters (i.e., database name, db user login, db password, db host name). The latter, i.e., *load mode*, filters the type of nodes to be loaded for the sake of efficiency.



```
CATEGORIZE article Ecosystem VALUES (Ecosystem)
UNCATEGORIZE article Architecture_of_Participation VALUES (Crowdsourcing)
CATEGORIZE article Architecture_of_Participation VALUES (Ecosystem)
MERGE article LocalStorage WITH StructuralEvents
RENAME article LocalStorage_StructuralEvents TO Realization
MOVE Realization_Issues FROM article Customization TO article Realization
```

Figure 4.12: The WikiWhirl tracking window traces the refactoring operations.

Nodes can be filtered by either name or type. The former permits to focus on some part of the wiki based on the category name: the category and all its descendants are loaded. By contrast, type-based filtering permits to focus on structural refactoring (type = "category") or content refactoring (type = "content") by filtering by category nodes or rather, rendering all, categories, articles and sections alike.

Refactoring transactions. FreeMind sessions tend to look more like transactions where the user takes the decision about committing the session once he observes the resulting map. FreeMind has been extended with a *tracking window* (see Figure 4.12) where user interactions are described in terms of WikiWhirl operations. This window offers three buttons: the *Commit* button that makes the changes to endure in the wiki database; the *Delete* button, that removes the highlighted operation from the trace; and the *Refresh wiki mind map* button, which restores a database dump and regenerates the mind map. Workers can play around, exchange insights, and finally, commit to the resulting structure.

FreeMind-MediaWiki roundtrips. Users tend to move back and forth

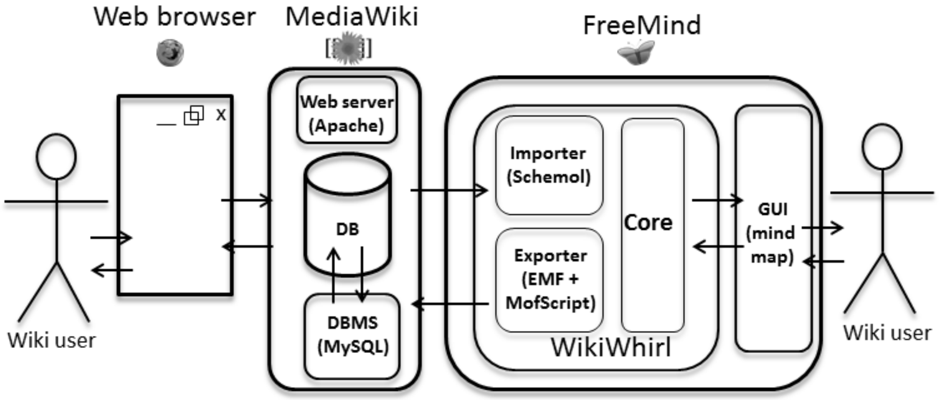


Figure 4.13: WikiWhirl architecture.

between the FreeMind view and the MediaWiki view. FreeMind permits a smooth transition between both views by turning node titles into URLs. In this way, the *Wiki* map becomes “a site map” for MediaWiki installations i.e., the map accounts for a global view of the wiki where users are a click-away from the *HTML* page in the MediaWiki front-end. This serves two purposes. First, to access the full content, should this matter for the refactoring purpose. Second, to check out the outcome of the refactoring operations. Once some refactoring operations are committed, users can move to the MediaWiki view to see the impact of such refactoring. This reinforces the perception of the WikiWhirl and the MediaWiki front-ends as complementary views of the wiki corpus.

4.7.5 Architecture of the WikiWhirl Plugin

The WikiWhirl plugin pivots around three main components (Figure 4.13):

- The *core* component, which confines FreeMind to the semantics of WikiWhirl. This includes the additional restrictions on the *Wiki* map, the validation of the pre-conditions upon user interactions (and the corresponding error messages), and the tracking window.

- The *importer* component, which extracts mind maps out of the wiki databases. For this purpose, we used Schemol (Appendix A).

- The *exporter* component, which enacts the refactoring script. This process is threefold. First, the refactoring trace (as specify in the *tracking window*) is mapped to a WikiWhirl model using the EMF persistence framework[15]. Second, the WikiWhirl model is transformed into an SQL script. This model-to-text transformation is realized through MOFScript. Finally, this *SQL* script is enacted against the MediaWiki DBMS (i.e., *MySQL*).

Wiki engines other than MediaWiki can be used. WikiWhirl could be extended by modifying the previous components. Specifically, the *importer* would need a new Schemol transformation to cope with the new database schema, and the *exporter* would need a new MOFScript transformation to generate the required SQL statements for the new DBMS.

Architecturally, the WikiWhirl plugin is a traditional database application: it queries the MediaWiki database, processes the tuples (e.g., renders the mind map), and saves some tuples back. By large, the most costly part is querying the database, specifically in two points (*i*) when sections are to be extracted out of large articles, since all the raw text of articles needs to be parsed, and (*ii*) when the category hierarchy is large, due to the need of recursively traverse the category tree. We conducted some tests to measure how WikiWhirl performs on four scenarios. The test was carried out on in a desktop PC Quad core 8GB RAM 64-bit connected to a remote database. Figure 4.14 depicts how WikiWhirl scales as the number of nodes varies[16]. These figures show that WikiWhirl handles efficiently most of the common refactoring scenarios. Worth noticing, once tuples are loaded and the map displayed, the database session is closed. This implies that keeping the map on display has no impact on the wiki database. Hence, it is possible to use the very same WikiWhirl map for

[15] `http://eclipse.org/modeling/emf/` (accessed December 2012).

[16] Actually, tuples are the artefacts being loaded that are next transformed into nodes.

Table 4.5: Performance test in four scenarios.

Scenario	**Nodes**				**Load time**
	Categories	Articles	Sections	Total	seconds
Wiki initialization	50	0	0	50	20
Structure refactoring	5	20	0	25	6
Content refactoring	5	20	80	105	15
Refactoring complete view	70	350	800	1220	4200

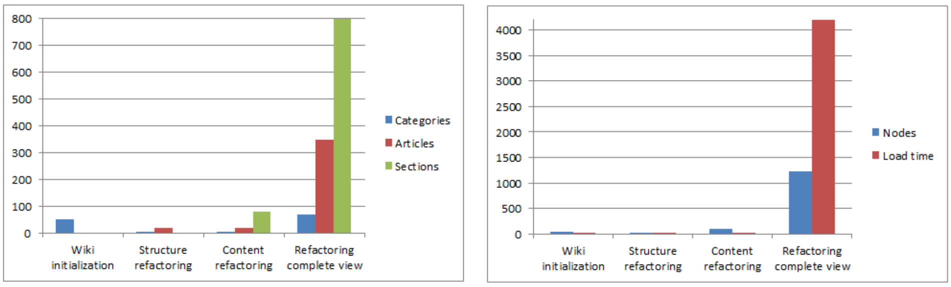


Figure 4.14: Graphics for the performance test in Table 4.5.

different refactoring sessions, hence, saving the loading phase at the onset. This might cause the map to get outdated as other users update the wiki through the MediaWiki view (e.g., deleting a category or an article). This is certainly possible. Should this happen, a transaction error will arise at the time the user attempts to save the refactoring that happens to act upon a node that has disapeared. The user would need to reload the map and start again. It is up to the user to estimate the likelihood of this error based on the potential number of users and the scope of the refactoring.

4.8 Evaluation

WikiWhirl aims to increase refactoring affordance for knowledge workers. Affordance is *a perceived opportunity for action* and perception is the organization, identification and interpretation of sensory information in order to represent and understand the environment. Here, the environment is the wiki, and the action on this environment is refactoring. WikiWhirl seeks to increase *the perceived opportunity for wiki refactoring* (the action). This is achieved by (*i*) surfacing the structure of the wiki corpus through mind maps ("the sensory information"), and (*ii*) conducting refactoring as mind map reshaping. This section evaluates to which extent this aim has been fulfilled.

Therefore, WikiWhirl is compared with MediaWiki (i.e., the baseline alternative) along tree measures: *understandability* of the wiki structure, *effectiveness* in refactoring, and *productivity* (i.e., efficacy) in refactoring. The rationales are that

> "*enhancing understanding will ease the detection of refactoring opportunities which, in turn, will impel users to refactor provided the means to do so in a effective and efficient way are available*."

If either the wiki structure is difficult to grasp or the refactoring means are cumbersome to use, then, users will not be inclined to act. This is evaluated with a controlled experiment. A controlled experiment permits to assess the existence of a cause-effect relationship between the use of WikiWhirl and an increased affordance. In software engineering, a controlled experiment can be defined as a randomized experiment (a.k.a. quasi-experiment) in which individuals or teams (*the experimental units*) conduct one or more software engineering tasks for the sake of comparing different populations, processes, techniques or tools (*the treatments*) [SHH$^+$05]. In this case, the target audience are knowledge workers who are accustomed to article editing, but with no specific wiki refactoring background. Next sections delve into the details following the experimental framework proposed in [JP05].

4.8.1 Experimental Design

First, we state the **goal of the experiment** using the Goal/Question/Metric (GQM) method [Bas92]:

- *analyze* the wiki refactoring affordance of knowledge workers working with WikiWhirl
- *for the purpose of* comparing it with a baseline alternative (MediaWiki)
- *with respect to their* effectiveness, global understandability and productivity
- *from the point of view of* a researcher trying to assess WikiWhirl
- *in the context* of a case study on selected representative WikiWhirl operations.

This goal introduces three **dependent variables**, namely:

1. *Global understandability.* It is defined as the effort required for the reading and correct interpretation of the artefact at hand (e.g., a wiki) [Pat08]. Measuring understandability is still open to debate. As for how understandability is measured, following the guide defined in [Pat08] to test the understandability of different notations, we introduce a set of multiple choice questions regarding the structure and the semantic content of a wiki. Both, the number of questions answered in the allocated time, and the number of such questions that were correctly answered are evaluated.

2. *Effectiveness.* It is defined as "the capability of the software product to enable users to achieve specified goals with accuracy and completeness in a specified context of use" [ISO]. To measure accuracy and completeness, a point will be assigned to each action that follows the good practices in the course of a refactoring

operation. For instance, 7 points can be earned for a *merge* if the user performs the following tasks: create new article, copy and paste content of each source article, add summary, create discussion, and create redirects from both source articles to the new one. The more points the participant obtains, the more complete and accurate the task outcome. In addition, we also measure whether the participants have been able to complete the task in the allocated time.

3. *Productivity.* It is defined as "the capability of the software product to enable users to expend appropriate amounts of resources in relation to the effectiveness achieved in a specified context of use" [ISO]. In this context, productivity is used for its relation with time availability. Productivity will be measured using task completion time.

The **independent variables** include WikiWhirl as the tool to be measured, and MediaWiki as the baseline alternative. Hence, the experiment follows a unifactorial design where the rest of the parameters are controlled. These **controlled parameters** include:

1. *The participants' previous exposure to* MediaWiki. We control this variable through a randomized block design where MediaWiki knowledge is the blocking variable [Spe93]. That is, participants are divided into three groups depending on their previous MediaWiki skills, and only inside these groups are they randomly assigned to the WikiWhirl or MediaWiki groups. This ensures a comparable MediaWiki background in both experimental groups.

2. *The subjects' familiarization with the domain used for the wiki in the experiment.* We ensure equal understandability by selecting a wiki from the veterinarian domain where the participants have a similar and scarce knowledge.

3. *The experimental setting.* The very same laboratory, experiment start time, observers and trainers will be used for both groups.

Hypotheses. The goal above leads to the following hypotheses:

- *H1null* Using WikiWhirl has no impact on *effectiveness* when contributors perform refactoring operations on a wiki.

- *H1alt* Using WikiWhirl has a significant impact on *effectiveness* when contributors perform refactoring operations on a wiki.

- *H2null* Using WikiWhirl has no impact on *global understandability* when contributors perform refactoring operations on a wiki.

- *H2alt* Using WikiWhirl has a significant impact on *global understandability* when contributors perform refactoring operations on a wiki.

- *H3null* Using WikiWhirl has no impact on *productivity* when contributors perform refactoring operations on a wiki.

- *H3alt* Using WikiWhirl has a significant impact on *productivity* when contributors perform refactoring operations on a wiki.

Participants. Participants were recruited among the Computer Science Faculty and Ph.D. students of the University of the Basque Country. The only prerequisite is to have experience on wiki editing with MediaWiki. Participants from this sample can be assumed to have a similar background, while having an above average command of software and new technologies (see Subsection 4.8.4).

Tasks. Three tasks were designed:

1. *Comprehension Task.* In order to measure global understandability, participants are confronted with a subset of the WikiVet wiki (see Figure 4.3), and a set of questions to assess to which extent they apprehend the deep structure of this wiki (see Appendix C). The questionnaire was prepared along the guidelines of [Pat08].

2. *Structure-refactoring Task.* Based on this initial WikiVet wiki, participants will conduct distinct structure refactoring operations: create, categorize, uncategorize, rename and drop. The description of these tasks corresponds to the one introduced in Subsection 4.2.2.

3. *Content-refactoring Task.* Based on the initial WikiVet wiki, participants are told to conduct distinct content refactoring operations: split, merge and move. The description of these tasks corresponds to the one introduced in Subsection 4.2.3.

Note that wiki initialization is left out of the experiment, as wiki initialization does not *per se* involve refactoring. It was excluded to shorten the experiment and avoid negative position effects (e.g., participants getting bored or tired) [Sar05].

Instrumentation. As aforementioned, a randomized block design is used using previous MediaWiki knowledge as the blocking variable. To this end, a questionnaire on MediaWiki knowledge and the frequency of its use by participants was designed (see Appendix C). A 50 minute training session was created for both groups. This training teaches how to perform refactoring operations with either MediaWiki or WikiWhirl depending on the group. For the experiment itself, handouts were written with textual instructions (e.g., what operation to perform on the wiki, when to take note of the time, etc). The slides used in the training (the same number of slides for both groups), will also be provided to participants. Two additional questionnaires were created, one to evaluate global understandability (see Appendix D) and another to gather the final results (see Appendix E).

Data Collection Procedure. Three different ways of data collection are envisioned. First, two observers will be present on each group to gather qualitative data on the execution. Second, execution time and questionnaire responses will be manually introduced by participants using the corresponding online questionnaire (see Appendix E). Third, authorship and readership errors will be manually collected by the observers after the execution of the experiment by recovering database dumps and logs from the computers used by participants. All this data will be collected anonymously. The data from the different sources (i.e., previous MediaWiki knowledge, understandability questionnaire, final questionnaire and computer data) for the same participant will be linked using a code.

Analysis Procedure. Descriptive statistics will be used to characterize the sample and to evaluate the participants' experience. Moreover, *t-test* analyses will be performed to assess differences among the MediaWiki and WikiWhirl groups. In case of statistically significant differences, *Cohen's d* will be used to calculate the size of the effect [Coh88]. IBM SPSS Statistics 20 for Windows[17] will be employed to perform the different analyses.

4.8.2 Execution

This section summarizes the execution of the aforementioned experiment conducted in December 2012. A call was issued to the faculty and Ph.D. students of the Faculty of Computer Science of the University of the Basque Country asking for volunteers to participate in the experiment. The call specified that wiki contributors were sought, so knowledge of how to edit an article was a must. It also stated that participants of the experiment would learn about wiki refactoring and enjoy a drink and a snack after completion. Thirteen people answered the call, three lecturers and ten Ph.D. students. Before the experiment they answered the questionnaire regarding MediaWiki knowledge. When processing their answers, an outlier who had a considerable experience in wiki refactoring was identified. Out of 34 possible points in the questionnaire, he obtained 27 while the second person with more MediaWiki experience obtained 16. Hence, the outlier was eliminated from the sample. As a result, a fairly homogeneous group of 12 participants was obtained, with an

[17] `www-01.ibm.com/software/analytics/spss/products/statistics` (accessed December 2012).

average of 10.33 points (out of 34) and a standard deviation of 2.57 in MediaWiki background. All participants indicated they knew how to edit an article and 75% knew how to create one, but 50% never actually perform the operation. Only two participants reported they knew how to create categories and none of them knew how to delete articles. None ever performed refactoring operations (i.e., reorganize existing articles and categories).

To ensure these 12 participants were evenly distributed among the MediaWiki group and the WikiWhirl group, a randomized block design was used. Participants were classified along three clusters of four people each based on the points obtained in the questionnaire: 7 to 9, 10 to 11 and 11 to 16. Within each cluster, participants were randomly assigned to either the MediaWiki group or the WikiWhirl group.

Hardware homogeneity was ensured by using the very same laboratory for both groups. All participants used computers with the same features (i.e., Intel Core 2 1.86 GHz, 3 GB RAM and Windows XP Professional SP3) and a clean installation of MediaWiki version 1.16.1. The individual MediaWiki installations are required to allow each participant to perform her own refactoring operations. On top of this, the WikiWhirl participants also had FreeMind 0.9.0 and WikiWhirl 0.3 installed. Each group was cited at the same hour (4.30 pm) of two different days. The same observers participated.

Before the experiment began, every participant signed an informed consent. Both groups received a 50’ training on wiki refactoring. The same trainer gave a brief introduction on wikis, wiki evolution and wiki refactoring (10’) and then explained the different operations that are performed during refactoring (40’). As the operations were being described, participants executed them in a local installation each had for learning purposes. This training installation is based on Evelopedia, a wiki on online games[18]. This training was followed by the three experimental tasks (see Subsection 4.8.1).

[18] `http://wiki.eveonline.com/` (accessed December 2012).

Table 4.6: Participant Background along a Likert scale from 1 (none) to 5 (expert).

Item	Mean	St. Dev.
Programming	4.00	0.74
Web Engineering	3.75	0.62
Web 2.0	3.33	0.89
Collaborative Systems	2.41	0.51
Mind Maps (WikiWhirl participants)	2.14	1.12

4.8.3 Analysis

This section analyses the data collected from the experiment. Descriptive statistics are described first, then hypothesis testing is carried out followed by an analysis of other parameters that might have influenced the result.

Descriptive Statistics

Participant background. Twelve participants from the Faculty of Computer Science of the University of the Basque Country participated in the experiment (3 lecturers and 9 Ph.D. students). The majority of participants were male (58.3%). Regarding age, the average was 31.5 years with a standard deviation of 6.17. With respect to their background, participants had to rate themselves using a Likert scale from 1 (none) to 5 (expert). Results can be seen in Table 4.6. Note that participants working with WikiWhirl were asked their background with mind maps, as it could have an impact on their performance.

Dependent variables (see Table 4.7)

1. *Global understandability*. It is measured through the comprehension task. Specifically, we count the number of answers fulfilled in the allocated time (20') and, out of them, the number of correct answers.

2. *Effectiveness*. It is measured through the refactoring tasks. Through a questionnaire, participants indicated to which extent they considered that the tasks have been completed in the allocated time

Table 4.7: Experimental Measures.

		MediaWiki		**WikiWhirl**	
Variable	**Metric**	Mean	St. Dev	Mean	St. Dev
Global Understandability	# Answers (out of 14)	6.00	1.09	14.00	0.00
	# Correct Answers	4.50	0.84	12.50	1.05
Effectiveness	# Struct. Ref. Completed (range: '1' fully, '0' none)	0.33	0.52	1.00	0.00
	# Struct. Ref. Points (range: '23' fully, '0' none)	14.67	4.80	23.00	0.00
	# Cont. Ref. Completed (range: '1' fully, '0' none)	0.50	0.55	1.00	0.00
	# Cont. Ref. Points (range: '36' fully, '0' none)	20.83	6.46	36.00	0.00
Productivity	elapsed time for the comprehension task (minutes)	20.00	0.00	12.17	3.66
	elapsed time for the structure-refactoring task (minutes)	18.17	3.61	9.67	1.36
	elapsed time for the content-refactoring task (minutes)	17.17	3.19	9.71	3.12

(20') where "1" indicates completion and "0" non completion. It is worth noticing that in the MediaWiki group only one third of the participants considered that they had completed the structure refactoring task. Moreover, we also measured to which extent the task was correctly achieved, i.e., preserving authorship and readership independence (e.g., including redirects, notices, etc). In this case, WikiWhirl participants obtained all the possible points, as these actions are transparently performed by WikiWhirl.

3. *Productivity*. We measured the time spent in each task: the comprehension task, the structure-refactoring task and the content-refactoring task.

Table 4.8: Affordance Test Outcome.

Metric	t	p	d
Struct. Ref. Completed	-3.162	0.025	-1.825
# Struct. Ref. Points	-4.250	0.008	-2.454
Cont. Ref. Completed	-2.236	0.076	
# Cont. Ref. Points	-5.748	0.002	-3.319

Hypothesis Testing

This section tests the three hypotheses regarding global understandability, effectiveness and productivity. Hypotheses are evaluated using a t-test. The outcome is displayed using tables where the *t* and *p* columns present the *t* value in the t-test and the significance, respectively. In metrics where statistically significant differences were found, the effect size was calculated using *Cohen's d* [Coh88]. The values are presented in the last column of these tables.

We begin by analyzing **effectiveness**. The hyphothesis reads as follows:

- *H1null* Using WikiWhirl has no impact on *effectiveness* when contributors perform refactoring operations on a wiki.

- *H1alt* Using WikiWhirl has a significant impact on *effectiveness* when contributors perform refactoring operations on a wiki.

Table 4.8 summarizes the results. Regarding the structure refactoring task, the table shows statistically significant differences, i.e., WikiWhirl helps participants to fullfil timely (*"Struct. Ref. Completed"* metric) and completely and accurately (*"# Struct. Ref. Points"* metric) the task at hand. On the other hand, the content refactoring task exhibits no statistically significant difference for the *"Cont. Ref. Completed"* metric (i.e., $p > 0.05$), although such difference is found for the *"# Cont. Ref. Points"* metric. This is most interesting as it seems to suggest that MediaWiki participants thought they had fullfilled the task at hand but without following the good practices of preserving authorship and

Table 4.9: Global Understandability Test (H2).

Metric	t	p	d
# Answers	-17.889	0.000	-10,328
# Correct Answers	-14.606	0.000	-8.433

readership independence. For the metrics where statistically significant differences were found (i.e., $p < 0.05$), the effect size was calculated using *Cohen's d*. A value of this parameter above 0.8 in absolute values denotes a large effect size as a result of the independent variable (i.e., the introduction of WikiWhirl) [Coh88]. Except for the *"Cont. Ref. Completed"* metric, which indicates a subjective perception of participants, the other metrics show considerable effect sizes and soundly sustain the case that the null hypothesis can be rejected, and hence,

> *WikiWhirl has a significant impact on effectiveness for wiki contributors*

The second hypothesis is related to **global understandability**:

- *H2null* Using WikiWhirl has no impact on *global understandability* when contributors perform refactoring operations on a wiki.
- *H2alt* Using WikiWhirl has a significant impact on *global understandability* when contributors perform refactoring operations on a wiki.

Table 4.9 presents the results. Statistically significant differences were found in both the number of answered questions ("# Answers" metric) and the number of them that were correct ("# Correct Answers" metric). *Cohen's d* shows large effect sizes for both metrics. Hence, the null hypothesis is rejected and we infer that using

> *WikiWhirl has a significant impact on global understandability when contributors perform refactoring operations*

Table 4.10: Productivity Test (H3).

Metric	t	p	d
Understandability	5.248	0.003	3.030
Struct. Ref.	6.212	0.000	3.586
Cont. Ref.	4.389	0.001	2.534

The significant differences between MediaWiki (4.5 mean) and WikiWhirl (12.5 mean) on correctly answering the test in the 20' frame (see Table 4.7) highlight the difficulty of easily grasping the wiki corpus with current wiki front-ends. As a conjeture, this situation might be due to wikis being iniatially thought for open encyclopedia-like wikis. Having a global understanding of the wiki corpus (even a subpart of it) was not a priority. Wikipedia is article minded rather than corpus minded. It sits the article at the center for editing, searching and navigating. However, corporate wikis look at wikis not only as archives of knowledge for later referral but also as enablers of knowledge formation. This might require to step back and look at the structure of the corpus before delving into a particular article. What is needed is a fluent mechanism that permits switch between both views.

The last hypothesis is related to **productivity**.

- *H3null* Using WikiWhirl has no impact on *productivity* when wiki contributors perform refactoring operations on a Wiki.

- *H3alt* Using WikiWhirl has a significant impact on *productivity* when wiki contributors perform refactoring operations on a Wiki.

Table 4.10 shows how statistically significant differences were found in the time participant spent in each task. *Cohen's d* again shows large effect sizes for the three metrics that were evaluated. Hence, the null hypothesis is also rejected and we infer that

> *WikiWhirl has a significant impact on productivity when contributors perform refactoring operations*

Table 4.11: Mind Map Knowledge Comparison.

		Low Knowledge		High Knowledge		Comparison	
Variable	**Metric**	Mean	St. Dev	Mean	St. Dev	t	p
Global Understandability	# Answers	14.00	0.00	14.00	0.00		
	# Correct Answers	12.00	0.82	13.50	0.71	-2.191	0.094
Affordance	Struct. Ref. Completed	1.00	0.00	1.00	0.00		
	# Struct. Ref. Points	23.00	0.00	23.00	0.00		
	Cont. Ref. Completed	1.00	0.00	1.00	0.00		
	# Cont. Ref. Points	36.00	0.00	36.00	0.00		
Productivity (mins)	Understandability	11.75	4.57	13.00	1.41	-0.359	0.738
	Struct. Ref.	9.25	0.96	10.50	2.12	-1.072	0.344
	Cont. Ref.	9.75	3.77	8.00	1.41	0.604	0.578

Parameter Influence

Next step was to check whether previous background on mind mapping would favor the positive results obtained by the WikiWhirl group. Hence, WikiWhirl users were asked to rate their background in the matter using a Likert scale from 1 (none) to 5 (expert). The results from the four participants with low previous knowledge (2 or less in the scale) and the two ones with higher knowledge (3 or more) were compared.

Table 4.11 presents the results. In the cases of number of answers, completion of structure and content refactoring and the associated points results are exactly the same for both groups (hence, no t-test could be performed). This shows that there were no differences among both groups. For the rest of the metrics, no statistically significant differences were found between the results of both groups either.

Overall, the results support the hypotheses regarding the gains obtained by empowering knowledge workers with a tool to ease refactoring. We can conclude that the use of WikiWhirl significantly improves wiki refactoring affordance for wiki contributors by improving global understandability, by reducing refactoring procedure skills and by reducing the need for time availability as it increases productivity.

4.8.4 Threats to Validity

A main concert for internal validity in our case is the appropriateness of the sample size. Johnson et al. suggest six participants per group as the minimum required for a controlled experiment [Joh92]. Our experiment accounted precisely for six subjects in each group. Even though our results show statistically significant differences and large effect sizes, larger groups are needed to corroborate these findings.

A second issue concerns the participants' background. So far, the experiment was conducted in a controlled environment, where participants were closely guided in the tasks. Moreover, it took place in an academic setting where participants have a strong background in computer science. Even though the refactoring tasks as such tend to be similar no matter whether the setting is academic or business oriented, the user motivations might vary. This leads us to external validity. More evaluations need to be conducted in a real business setting with a more diverse sample of participants. We reckon that productivity and authorship preservation would be even more appreciated in a business setting. However, this has yet to be proven. All in all, this experiment provides first evidences about the benefits of empowering wiki contributors with high-level refactoring tools for wikis.

4.9 Wiki Refactoring Backed by the Community: Ballots

During this chapter, it has been taken for granted that refactoring changes compliant with good practices will be backed by the wiki community. However, ask yourself: *what happens when a refactoring change must be backed by the whole wiki community*?. This section presents an alternative idea as a matter of potential discussion.

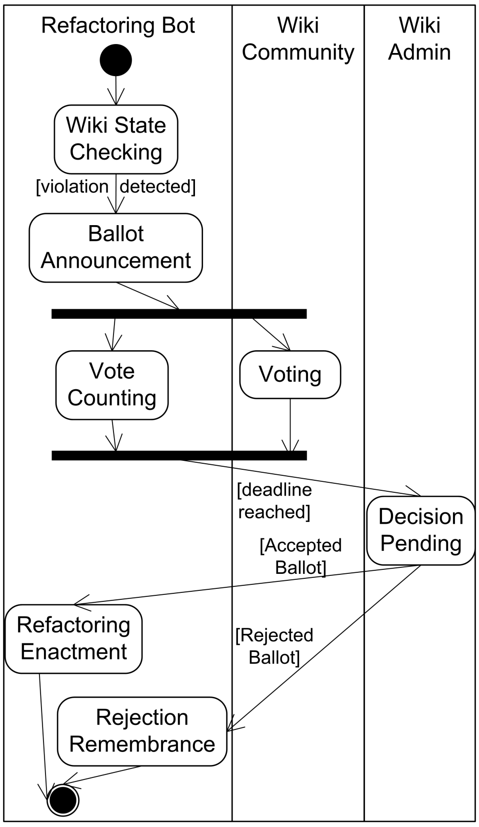


Figure 4.15: The Ballot Process.

Currently, refactoring in wikis is commonly done in wikis either by dedicated people (i.e., assigned or self-assigned) or by bots. Roles of people are known as *wikigardeners*, *wikignomes* or just *maintainers* [Mad08]. Their duties include: fixing typos, improving navigability and readability, categorizing or rearranging pages, splitting long pages, adding links, etc. These roles remind to mow the grass with a scythe, which is a manual work, whereas WikiWhirl advocates for an assisted approach, like using a lawnmower to mow the grass.

On the other hand, bots[19] are the most common mechanism for automating repetitive tasks. Wikis have many routine tasks to be done and many of them are too cumbersome to be performed by users (e.g., mass

[19] `http://en.wikipedia.org/wiki/Wikipedia:Bots` (accessed December 2012).

edits or check copyright violations). Bots are tools that take care of article maintenance.

Unfortunately, and unlike traditional refactoring, no regression test exists to check the validity of the refactoring output. Some changes, even if compliant with good practices, can still require to be backed by the community which ends up bearing the maintenance burden. This calls for a semi-automatic approach where "refactoring bots" (i.e., bots in charge of performing changes) interact with wiki users to confirm the upgrades. As an example, consider a guideline that states that a hierarchy of categories has *too much structure in depth* (i.e., too many nested categories with few articles). A user may think that merging some of those categories would improve the overall structure. However, the goodness of this outcome much depends on the user's mental model, and hence, it is debatable. In this setting, we propose the *ballot process*.

The ballot process

The ballot process intertwines actions from the refactoring bot (*"rebot"*) and the wiki community as follows (see Figure 4.15):

1. The rebot monitors whether the wiki's current state violates some guidelines.

2. If so, the rebot notifies this situation to the wiki community by setting a discussion page.

3. The discussion page creation launches two parallel activities. First, the wiki community can now cast votes in the ballot for the issue at hand. Second, the rebot periodically monitors the discussion, and counts the votes.

4. If the ballot deadline is reached, the ballot is moved to the pending state. Now, the wiki administrator accepts or rejects the proposal based on the ballot outcome by using the categorizing the page in question with *AcceptedBallots* or *RejectedBallots*.

5. If the ballot is accepted then, the rebot conducts the change.

6. If the ballot is rejected then, the rebot records the decision to avoid future ballots on this issue.

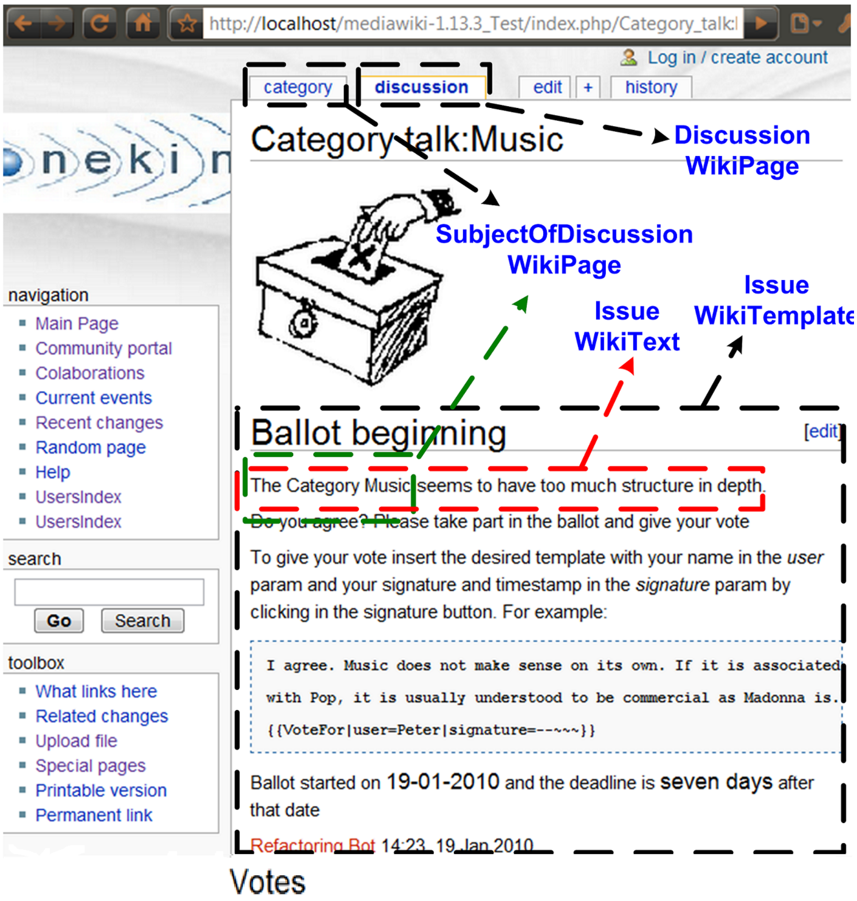
Figure 4.16 shows an example where the generated discussion page by the rebot communicates that the category *Music* presents *too much structure in depth* (i.e., this category is in a hierarchy of categories too much nested in the wiki structure). After the creation of the discussion page, users can vote in that page. Users can also comment on their votes to support their reasons.

In this situation it is not clear how could fit WikiWhirl with a ballot process, however this opens new paths to research and discuss as future work.

4.10 Related Work

Wiki Visualization. Lykourentzou et al. [LDP+12] highlight that wikis usually exhibit poor structure. On account of this, distinct tools try to improve the user experience by providing new textual or graphical representations. However, and as highlighted by Lengler and Eppler [LE07], there might not be only one appropriate visualization method for a body of knowledge. The selection of the visualization method depends on the requirements to meet. The visualization has to emphasize the aspect of the wiki we are interested on, examples include *Sonivis*[20], *Wiki Explorator*[21], *Sioc MediaWiki*[22], *WikiTracer*[23], *HistoryFlow* [VWD04], *WikiNavMap* [UK07]) or *Annoki* [TS10]. *Annoki* [TS10] is a set of extensions for MediaWiki to add extra functionality that has two interesting visualizations: *WikiMap* and *wiEGO*. *WikiMap* shows elements

[20] http://sonivis.org (accessed December 2012).
[21] www.kinf.wiai.uni-bamberg.de/mwstat (accessed December 2012).
[22] http://ws.sioc-project.org/mediawiki/ (accessed December 2012).
[23] http://wikitracer.com (accessed December 2012).



Category talk:Music

Ballot beginning

The Category Music seems to have too much structure in depth.

~~Do you agree?~~ Please take part in the ballot and give your vote

To give your vote insert the desired template with your name in the *user* param and your signature and timestamp in the *signature* param by clicking in the signature button. For example:

```
I agree. Music does not make sense on its own. If it is associated
with Pop, it is usually understood to be commercial as Madonna is.
{{VoteFor|user=Peter|signature=--~~~~}}
```

Ballot started on **19-01-2010** and the deadline is **seven days** after that date

Refactoring Bot 14:23, 19 Jan 2010

Votes

Figure 4.16: The rebot generates a discussion page.

(linked pages, authors, etc.) related only to the current shown page. Nonetheless, this limits the high level view of the whole structure, missing relations with third-levelled pages (i.e., deeper nodes in the mind map). *WiEGO* is a graphical page organizer that creates structure within wiki pages (similar to drag and drop sections in WikiWhirl) by saving a user-drawn graph as page sections. Hirsch et al. [HHGC10] coins the term "*Visual Wiki*" as a combination of a visual and a textual representation of a wiki. From this perspective, WikiWhirl offers a hybrid solution where FreeMind offers the visual view, which is linked to the textual HTML representation. However, these tools look at visualization as an end in itself whereas for WikiWhirl visualization is a means towards refactoring. Hence, WikiWhirl does not offer the detailed revision trace of *WikiTracer* or *HistoryFlow*. Neither does it offer means to visualize inter-article relationships such as *WikiExplorator* or *WikiNavMap*. The aim is to grasp the structure of the wiki corpus as captured by the wiki categories as the very first step towards refactoring. In this regard, *WikiMindMap* [Nyf09] is a tool that permits to browse *Wikipedia* articles using mind map visualization. *WikiMindMap* creates a node for each section of the article and also a node for each link of the article to other *Wikipedia* articles. The differences with WikiWhirl are (*i*) the aim (visualization *vs.* refactoring), (*ii*) the focus (sections and intra-links *vs.* wiki structure), and (*iii*) range (Wikipedia-restricted *vs.* MediaWiki wikis).

Wiki Management. Refactoring is part of wiki management. Wiki management includes a heterogeneous set of duties: wiki initialization, "Wiki Refactoring", spotting typos, etc. No matter the job, a common effort is to attempt to automatize/assist as much as possible. Bots[24] are the most common mechanism. Wikis have many routine tasks to be done and many of them are too cumbersome to be performed by users (e.g., mass edits or check copyright violations). Bots are tools that take care of article maintenance. However, other cases are more dubious, and automatic correction might be inappropriate. WikiWhirl contributes by proposing the use of DSLs as a way to facilitate engagement but leaving the control in the user's hand. We took a similar approach when addressing wiki initialization [DP11a]. Here, the aim was to depict a blueprint of the wiki as a mind map, and next, generate the wiki's initial installation. Now the problem is the other way around. WikiWhirl first extracts the wiki structure from an existing wiki; 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 initialization.

[24] `http://en.wikipedia.org/wiki/Wikipedia:Bots` (accessed December 2012).

As for refactoring. Rosenfeld et al. [RFD10] 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).

On the other hand, Ward Cunningham is working in a new type of wikis called "federated wikis" [Cun11]. "Federated" means that pages can be mixed up with other pages, if so desired. This introduces *composition by refactoring*: text, images and data can be drag and dropped between federated wikis. This mechanism is similar to the one proposed in this work, where the page sections can be drag-and-dropped between pages, in a graphical manner. However, our aim is refactoring rather than reuse.

Finally, and besides categories, Wikipedia provides additional means for grouping articles: lists and navigation templates[25]. Using pre-defined templates, a set of related articles can be referred to within a common structure. These mechanisms can be regarded as a kind of arrangement of set of articles to ease location. Unlike categories, no auto-linking is provided, i.e., including an article in a list does not automatically link back the article to this list. This showcases how MediaWiki support a basic semantics of categories which separate them from mere hyperlinks between pages. In a corporate setting, we do not regard an extensive use of these mechanisms mainly targeting bulky wikis where the sheer number of articles requires of additional organization mechanisms besides categories. Hence, WikiWhirl does not address lists and neither navigation templates.

[25] See `http://en.wikipedia.org/wiki/Wikipedia:Categories,_lists,_and_navigation_templates` for a comparison (accessed December 2012).

4.11 Conclusions

This chapter strives 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. In so doing, this work (*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, and (*iii*) introduces WikiWhirl, a DSL for wiki refactoring built on top of FreeMind. From this perspective, WikiWhirl offers a refactoring-minded alternative to traditional editing-minded front-ends as available in current wiki engines. The wiki's corpus is depicted as a FreeMind map, and map manipulations are interpreted as refactoring operations that end up being consolidated in the wiki database. First evaluations suggest that users find intuitive and less demanding to conduct wiki refactoring in terms of node rearrangement in a mind map.

According to the activity theory, the form of group collaboration may be influenced if certain affordances are promoted. WikiWhirl enhances refactoring affordances for wiki engines. It rests to be seen how collaborative refactoring is influenced by the availability of tools that reduce the skills required to participate. Similar to the introduction of affordable editing mechanisms (one of the hallmarks of wikis), improving refactoring affordances facilitates participation but also "the editing wars". Future work includes the investigation of how the structure and collaboration of wikis is affected by the empowerment of the layman with such sophisticated tools.

Parts of this chapter have already been published:

- Gorka Puente, Oscar Díaz, Maider Azanza "Refactoring Affordances in Corporate Wikis: A Case for the Use of Mind Maps", In *Enterprise Information Systems* journal, 2013. JCR, Impact factor 3.684. **Under review**.

- Gorka Puente, Oscar Díaz, "Wiki Refactoring as Mind Map

Reshaping". In *24th International Conference on Advanced Information Systems Engineering* (*CAiSE'12*), Gdansk, Poland, 2012. Acceptance rate 14%.

- Oscar Díaz, Gorka Puente, Cristóbal Arellano, "Wiki Refactoring: an Assisted Approach Based on Ballots". In *7th International Symposium on Wikis and Open Collaboration* (*WikiSym'11*), Mountain View, California, USA, 2011. Acceptance rate 42%.

Chapter 5

Wiki Customization through Web Augmentation Techniques[1]

"The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge."
– *Stephen Hawking.*

5.1 Overview

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 wiki users (e.g., Wikipedia's users) can easily transit from the public sphere to the private sphere, and vice versa. To this end, this chapter introduces *WikiLayer*, a plugin for Firefox that extends wiki rendering with augmentation

[1] Parts of this chapter have been previously presented [DAP12].

capabilities. WikiLayer permits wiki users to locally supplement articles with their own content (i.e., a *layer*). Layering additional content is achieved locally by seamlessly interspersing wiki content with the custom content. WikiLayer is driven by three main wiki characteristics: *affordability*, if users know how to edit, users know how to layer; *modularity*, evolving layers can be reduced or enlarged at users' wish; and *shareability*, layers can be shared in confidence through the wikipedian's social network (e.g., *Facebook*). This chapter motivates scenarios where readers, contributors and editors can apply WikiLayer as an augmentation mechanism.

WikiLayer is contextualized for Wikipedia for a better understanding of the cases samples, but it is also valid for any MediaWiki powered wiki. In fact, WikiLayer could perfectly fit with an initiative of the Wikimedia Foundation, the Public Policy Initiative [Wik11], and use web augmentation as an annotation way. The purpose of the initiative is to increase the use of Wikipedia as a teaching tool in University classrooms. Indeed, participants on the Wikimedia Foundation Initiative seek to encourage future Wikipedia contribution through exposure in the classroom [LOO+12]. But just reading does not seem to be enough. Understood as a learning activity, reading is commonly associated with taking notes. Different studies highlight the importance of annotation as a way to fix, relate and structure knowledge as you read [HWS07]. Annotation through web augmentation may promote 'deep reading', i.e., thinking, contrasting and contextualizing what you read for you own purposes.

The WikiLayer examples used through this chapter are available for download at **`http://webaugmentation.org/wikilayer.xpi`** and **`http://tinyurl.com/wikilayersamples`**.

This chapter starts motivating with some scenarios of use (Section 5.2) for later giving rise to the introduction of WikiLayer as a way to define layers (Section 5.3). Before the related work (Section 5.5) and conclusions (Section 5.6), WikiLayer is framed within Wikipedia (Section 5.4).

5.2 Motivating Scenarios

Wikipedia augmentation offers a backdoor for wiki users to customize Wikipedia articles with their own content. This augmented content (referred to as *layer*[2]) very much depends on their goals as readers, contributors or editors. This section introduces distinct scenarios for each of these roles. For comprehension purposes, this chapter provides some snippets of WikiLayer as it goes along. The full expressiveness of WikiLayer 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*" [AC10]. Reading Wikipedia spurs on 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 a protected way to practice editing 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, users can add their own local references for the article *XML* in Wikipedia, indicating whether a book is in the University library. Figure 5.1 depicts the XML article before and after being augmented with the following wikinote:

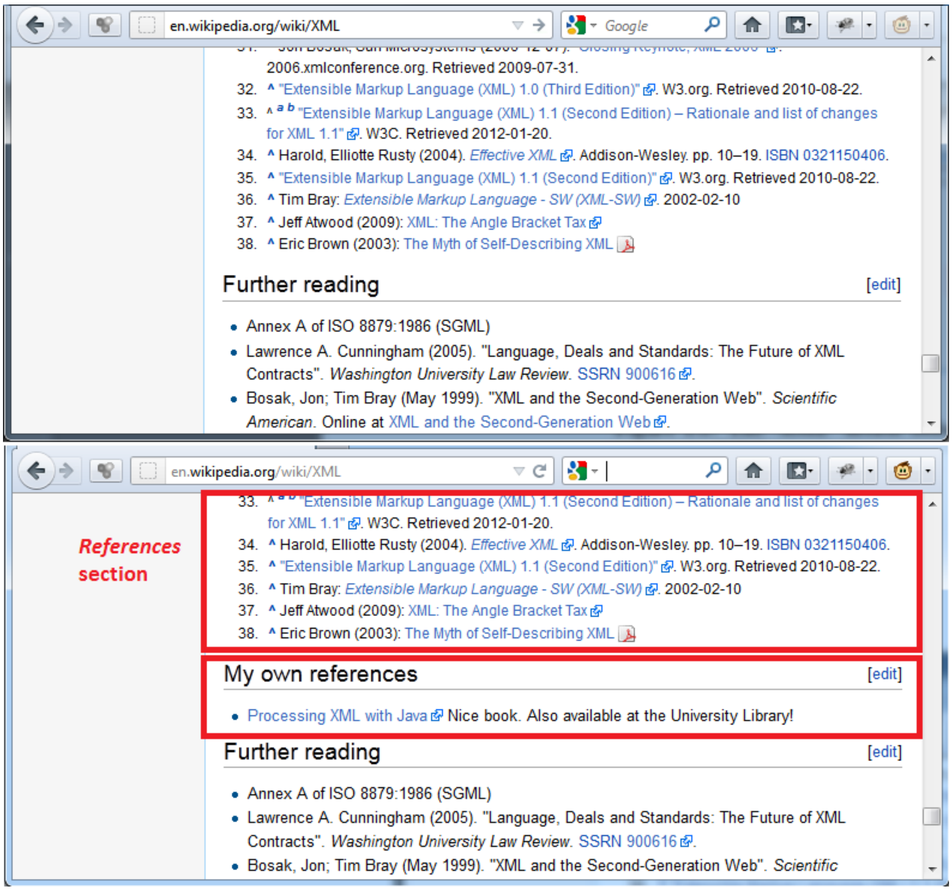
LayerOnArticle("XML").

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 the section *References* of the article XML. From now on, when navigating to the XML article, this reference will be seamlessly introduced on the fly (see Figure

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



2006.xmlconference.org. Retrieved 2009-07-31.

32. ^ "Extensible Markup Language (XML) 1.0 (Third Edition)". W3.org. Retrieved 2010-08-22.
33. ^ a b "Extensible Markup Language (XML) 1.1 (Second Edition) – Rationale and list of changes for XML 1.1". W3C. Retrieved 2012-01-20.
34. ^ Harold, Elliotte Rusty (2004). *Effective XML*. Addison-Wesley. pp. 10–19. ISBN 0321150406.
35. ^ "Extensible Markup Language (XML) 1.1 (Second Edition)". W3.org. Retrieved 2010-08-22.
36. ^ Tim Bray: *Extensible Markup Language - SW (XML-SW)*. 2002-02-10
37. ^ Jeff Atwood (2009): XML: The Angle Bracket Tax
38. ^ Eric Brown (2003): The Myth of Self-Describing XML

Further reading [edit]

- Annex A of ISO 8879:1986 (SGML)
- Lawrence A. Cunningham (2005). "Language, Deals and Standards: The Future of XML Contracts". *Washington University Law Review*. SSRN 900616.
- Bosak, Jon; Tim Bray (May 1999). "XML and the Second-Generation Web". *Scientific American*. Online at XML and the Second-Generation Web.

33. ^ a b "Extensible Markup Language (XML) 1.1 (Second Edition) – Rationale and list of changes for XML 1.1". W3C. Retrieved 2012-01-20.
34. ^ Harold, Elliotte Rusty (2004). *Effective XML*. Addison-Wesley. pp. 10–19. ISBN 0321150406.
35. ^ "Extensible Markup Language (XML) 1.1 (Second Edition)". W3.org. Retrieved 2010-08-22.
36. ^ Tim Bray: *Extensible Markup Language - SW (XML-SW)*. 2002-02-10
37. ^ Jeff Atwood (2009): XML: The Angle Bracket Tax
38. ^ Eric Brown (2003): The Myth of Self-Describing XML

My own references [edit]

- Processing XML with Java Nice book. Also available at the University Library!

Further reading [edit]

- Annex A of ISO 8879:1986 (SGML)
- Lawrence A. Cunningham (2005). "Language, Deals and Standards: The Future of XML Contracts". *Washington University Law Review*. SSRN 900616.
- Bosak, Jon; Tim Bray (May 1999). "XML and the Second-Generation Web". *Scientific*

Figure 5.1: The XML article before (top) and after (bottom) being subject to WikiLayer.

5.1). Other scenarios might be applicable to a set of articles characterized by their membership to a category. For instance, users might augment articles pertaining to the category XML with a link to *Delicious*[3] with information about who has bookmarked this article. Thus, readers can readily obtain a first feeling about the popularity of the article. The wikinote follows:

LayerOnArticle("Category:XML").

BeforeSection("1").

[3] `www.delicious.com` (accessed December 2012).

> *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*]*.")*

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

Contributors: layers as enablers of a perspective of co-existence. Collaborative knowledge building is basically a spiralled process where knowledge first emerges at individual context and then is socialized [NT95]. 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 structured along Wikipedia topics. This personal perspective might not be compatible with Wikipedia's *neutral point of view* (NPOV) policy [NPO]. Such neutrality leads to lean articles that focus on the bare essence of the topic at hand. That is, articles are devoid of any contextualized bias. Contextualization is not bad but addresses the topic from a specific perspective. For instance, the XML article restricts itself to introducing the rationales, history, criticism and core notions of this topic. This is sufficient for readers looking for an introduction to XML. However, consider a lecturer who refers to Wikipedia for those topics addressed in the classroom. Besides the bare description of the topic as found in Wikipedia, the lecturer can 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 `http://citizendium.org` where contribution might be limited to experts or gently guided by experts. In this setting, users 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. Editors manage and track content edition. Hence, the focus is not on article pages but on history pages (i.e., the revision history of each wiki page). The history page contains a list of the page's previous revisions, including the date and time of each edit, the username or IP address of the user who made it, and their edit summary. This information might be insufficient for decision taking. Rather, it only provides the main dimensions (i.e., when, who and what) for assessing editing, which might need to be supplemented with additional insights. Vandalism detection is a case in point. Recent studies [JML11, AdAMV$^+$11] 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://www.maxmind.com` and so forth. 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 edits but augmented with the Karma, should this data be significant for our analysis. Figure 5.2 compares the “real” history page and the page augmented by the following wikinote:

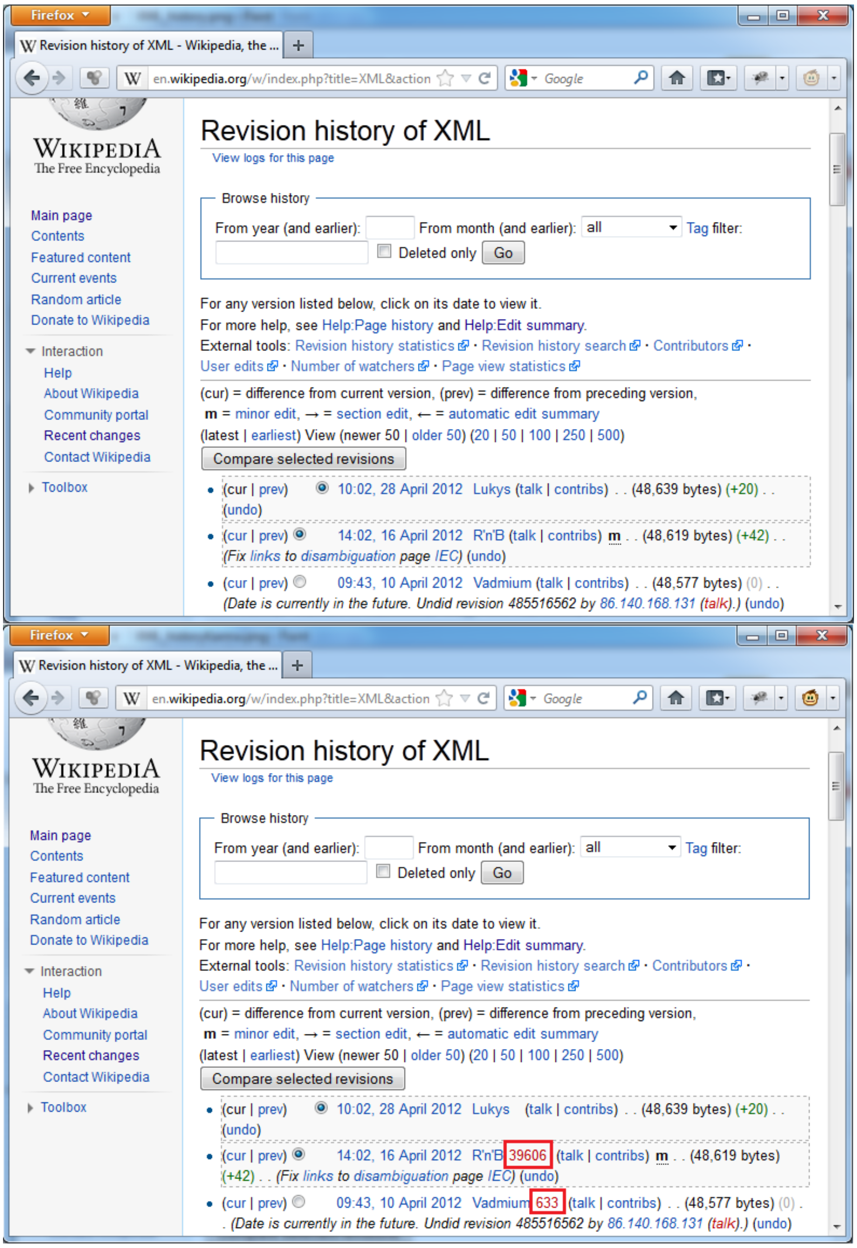
> *LayerOnHistory("XML").*
>
> *AfterUser().*
>
> *EmbedNote(extractFromPage("http://wpcvn.com/s/karma?username= $user"))*

This wikinote extends the user description in the XML history page with her Karma as obtained from `http://wpcvn.com`. In this case, the direct availability of the Karma improves the productivity of the editor.

Next section introduces WikiLayer, as a way to define layers within wikis. 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.

5.3 WikiLayer: Layers on Wikis

Web Augmentation is the act of superimposing additional directives on top of existing Web pages at run time [Bou99]. This approach is non-intrusive in the sense that the augmented website is unaware of the augmentation. This is achieved through JavaScript, using special weavers which permit a locally provided script to make on-the-fly changes to the currently loaded Web page. Weavers are available for Firefox (e.g.,



Revision history of XML

View logs for this page

Browse history

From year (and earlier): From month (and earlier): all Tag filter:

□ Deleted only Go

For any version listed below, click on its date to view it.
For more help, see Help:Page history and Help:Edit summary.
External tools: Revision history statistics · Revision history search · Contributors · User edits · Number of watchers · Page view statistics

(cur) = difference from current version, (prev) = difference from preceding version,
m = minor edit, → = section edit, ← = automatic edit summary
(latest | earliest) View (newer 50 | older 50) (20 | 50 | 100 | 250 | 500)

Compare selected revisions

- (cur | prev) ● 10:02, 28 April 2012 Lukys (talk | contribs) . . (48,639 bytes) (+20) . . (undo)
- (cur | prev) ○ 14:02, 16 April 2012 R'n'B (talk | contribs) **m** . . (48,619 bytes) (+42) . . (*Fix links to disambiguation page IEC*) (undo)
- (cur | prev) ○ 09:43, 10 April 2012 Vadmium (talk | contribs) . . (48,577 bytes) (0) . . (*Date is currently in the future. Undid revision 485516562 by 86.140.168.131 (talk).*) (undo)

Revision history of XML

View logs for this page

Browse history

From year (and earlier): From month (and earlier): all Tag filter:

□ Deleted only Go

For any version listed below, click on its date to view it.
For more help, see Help:Page history and Help:Edit summary.
External tools: Revision history statistics · Revision history search · Contributors · User edits · Number of watchers · Page view statistics

(cur) = difference from current version, (prev) = difference from preceding version,
m = minor edit, → = section edit, ← = automatic edit summary
(latest | earliest) View (newer 50 | older 50) (20 | 50 | 100 | 250 | 500)

Compare selected revisions

- (cur | prev) ● 10:02, 28 April 2012 Lukys (talk | contribs) . . (48,639 bytes) (+20) . . (undo)
- (cur | prev) ○ 14:02, 16 April 2012 R'n'B 39606 (talk | contribs) **m** . . (48,619 bytes) (+42) . . (*Fix links to disambiguation page IEC*) (undo)
- (cur | prev) ○ 09:43, 10 April 2012 Vadmium 633 (talk | contribs) . . (48,577 bytes) (0) . . (*Date is currently in the future. Undid revision 485516562 by 86.140.168.131 (talk).*) (undo)

Figure 5.2: A history page being augmented with information about Karma from `http://wpcvn.com`.

Greasemonkey), *Internet Explorer* (e.g., *IE7Pro* or *Turnabout*), *Safari* (e.g., *SIMBL* + *GreaseKit*), and natively supported in *Opera* and *Google*

Chrome. Unfortunately, JavaScript meets none of our requirements. Scripts are neither affordable (i.e., JavaScript is a convoluted programming language, ignored by most of wiki users), nor modular (i.e., scripts tend to be a bulk of code, difficult to enlarge and reduce along the wiki article evolution) and 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.

In this setting, the domain is Wikipedia augmentation and the target audience is Wikipedia users. That is, expected users are familiar with notions like article, category, edit, history, infobox or wikitext. In this way, looking at distinct scenarios of Wikipedia augmentation, and abstracting from them, we distinguish the main features of WikiLayer. Next section introduces those user visible characteristics to be tackled during Wikipedia augmentation as features of WikiLayer.

5.3.1 Features of Wiki Customization

Insights gained by looking at previous scenarios should now be made precise in terms of a feature diagram. Figure 5.3 shows such diagram for WikiLayer. The diagram states that a WikiLayer expression (a.k.a. wikinotes, the building blocks of layers) 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 like the article about XML).

- *Pointcut*. The scope delimits the pages subject to augmentation. However, a page might offer different injection points. These points

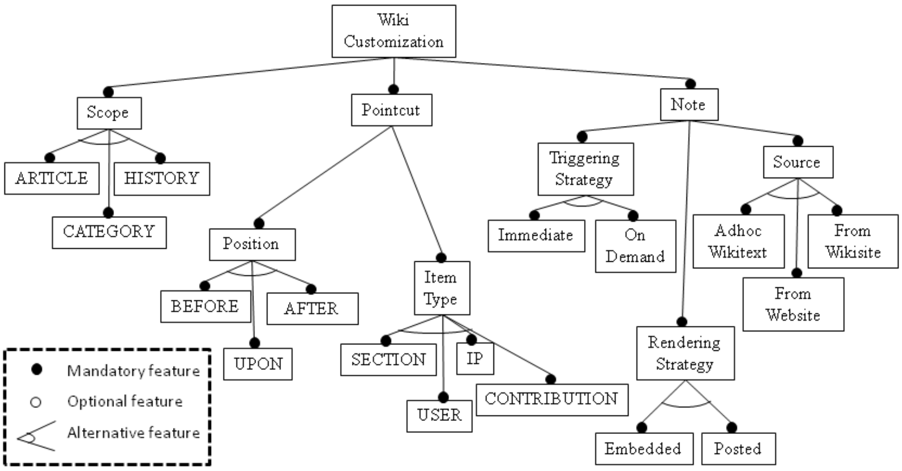


Figure 5.3: Feature Diagram: characterizing wiki customization.

are denoted after 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 basically wikitext. The source of a note can be static (i.e., directly provided by the user, *adhoc wikitext*) or dynamic (i.e., the outcome of a function). Functions permit notes to be extracted from *web sites* in general, or *wikisites* in particular. Notes also include both a *rendering strategy* and a *triggering strategy*. The former indicates whether the note is to be *embedded* or *posted* with regard to 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 wiki page. As for the *triggering strategy*, it refers to when the note is to be shown up. Matching the scope might directly lead to showing up the note (i.e., immediate strategy). However, this might entail cluttered pages if the page is heavily augmented. In this case, it might be preferable the augmentation to take place 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.).

Next subsection provides details about the description of WikiLayer expressions as layers.

5.3.2 Understanding WikiLayer Expressions

This section tackles the description of how to conceive WikiLayer expressions (i.e., wikinotes) through examples.

Setting the scope. First, users should indicate the kind of pages they are going to act upon: *LayerOnArticle()*, *LayerOnHistory()*, etc. Next, users 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 XML article (examples 1 and 2) or intentionally through category membership (example 3 acts upon all articles that belong to the XML category*).*

Setting the note. Next, users focus on the new material to be added, i.e., the note. WikiLayer supports three options: (*i*) directly provide the wikitext, (*ii*) extract the note from other wiki pages, or (*iii*) extract the note from other web pages. Examples follow:

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")*

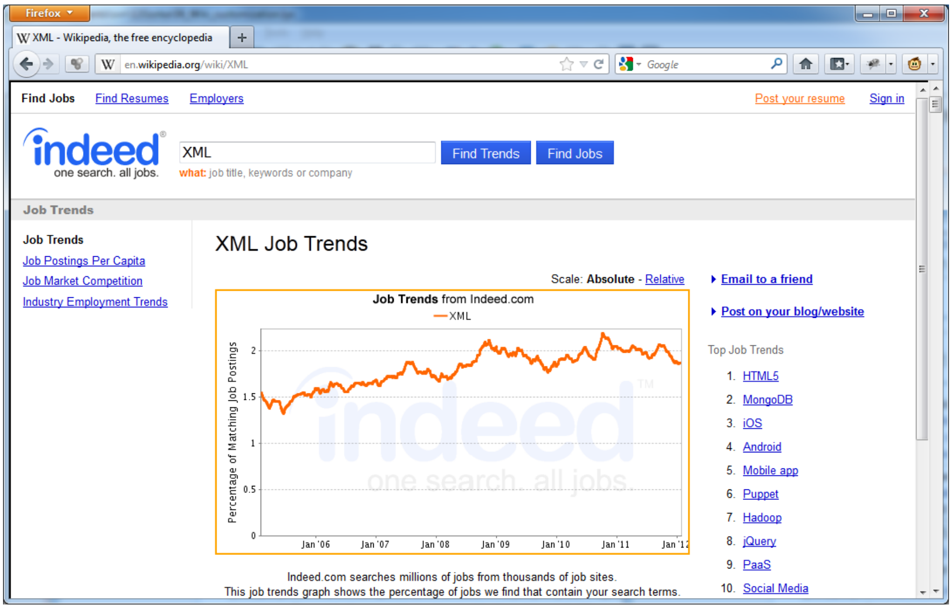
3. *extractFromPage("www.vogella.de/articles /JAXB/article.html")*

Example 1 explicitly provides the note as a piece of wikitext. If users know how to edit an article, then they know how to write a note. By contrast, examples 2 and 3 take the note from the websphere. Users do not provide the note but pinpoint to where the note can be obtained. WikiLayer uses a transclusion-like[4] approach whereby the inclusion is performed on demand at the time the wiki 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 desired region 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. Due to the lack of such items, XPath is used to pinpoint the HTML region to extract. But forcing the user to use raw XPath would hinder WikiLayer openness, since most wiki users probably ignore XPath. Therefore, we resort to programming-by-example. The first time a wikinote with an *extractFromPage()* function is enacted, the engine automatically navigates to this URL and intersperses a grid-like structure on top of the current *DOM* tree[5] (Figure 5.4). 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 (Figure 5.5). Subsequent enactments of this wikinote will directly extract this external region.

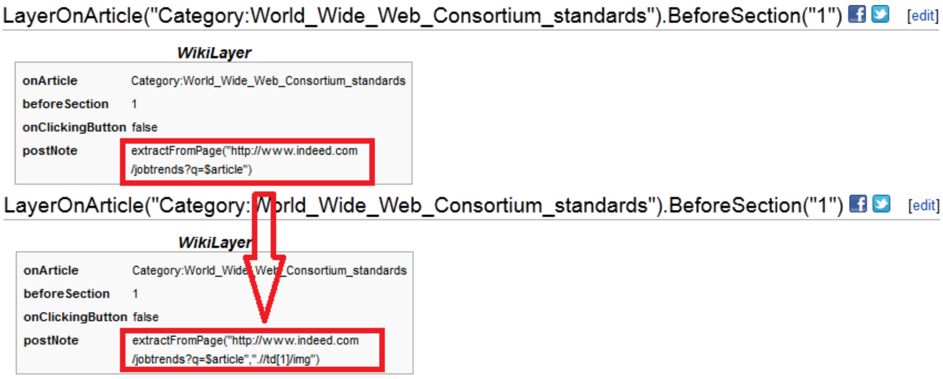
[4] Transclusion is the inclusion of the content of a document into another by reference.

[5] The runtime tree-like structure of HTML pages.



Indeed.com searches millions of jobs from thousands of job sites.
This job trends graph shows the percentage of jobs we find that contain your search terms.

Figure 5.4: WikiLayer navigates to the URL indicated in the *extractFromPage()*, and allows to select a fragment of that web site. The fragment is transformed in its XPath counterpart. In this case, the graph (highlighted by an orange box) is translated by WikiLayer to the XPath *.//td[1]/img* (Figure 5.5 down).



LayerOnArticle("Category:World_Wide_Web_Consortium_standards").BeforeSection("1") [edit]

WikiLayer

onArticle	Category:World_Wide_Web_Consortium_standards
beforeSection	1
onClickingButton	false
postNote	extractFromPage("http://www.indeed.com/jobtrends?q=$article")

LayerOnArticle("Category:World_Wide_Web_Consortium_standards").BeforeSection("1") [edit]

WikiLayer

onArticle	Category:World_Wide_Web_Consortium_standards
beforeSection	1
onClickingButton	false
postNote	extractFromPage("http://www.indeed.com/jobtrends?q=$article",".//td[1]/img")

Figure 5.5: WikiLayer expression before (top) and after (bottom) the selection of the fragment in Figure 5.4.

Setting the injection location. This addresses in-context presentation of wikinotes, i.e., how are integrated in the original wiki content. This implies to consider 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 and its clicking is necessary for the note to show up. As for the triggering strategy, it determines whether the note is to be embedded (*EmbedNote()*) or posted (*PostNote()*).

Putting the pieces together. The expression should be easy to read and understand. At this regard, setting the context right away is most important. In this case, the context is partially defined by the current page, i.e., the scope. Hence, WikiLayer starts with the scope, next the pointcuts within the scope, and finally, the note. Therefore, a wikinote looks like follows:

LayerOnArticle("XML").

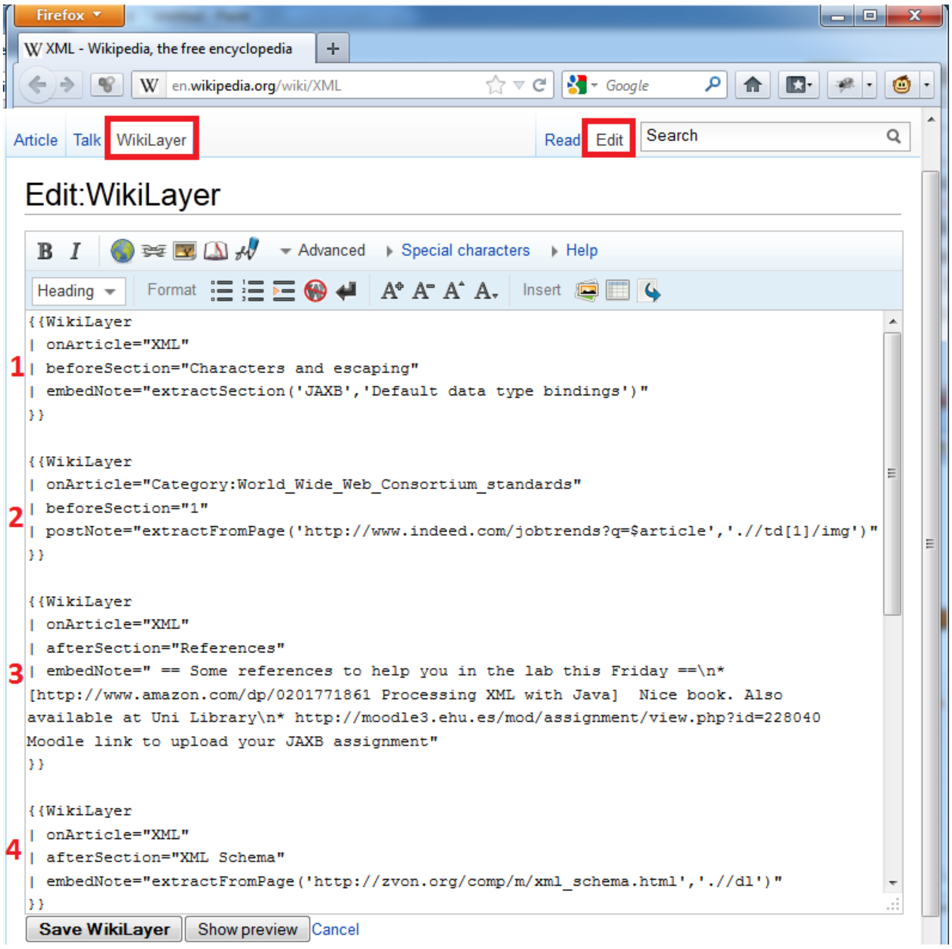
BeforeSection("XML as data type"). OnClickingButton().

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

5.4 Framing WikiLayer into Wikipedia

As previously mentioned, WikiLayer is JavaScript and it should be kept aside from wiki users. However, hiding JavaScript from wiki users' is not enough. We should strive to frame WikiLayer in its context of use, i.e., wikis. Otherwise, there is the risk that wiki users to be reluctant to skip to another platform/editor to write or maintain their layers. It should look like WikiLayer is part of the users' wiki (e.g., Wikipedia). Our aim is to make layer editing an *impulsive* action, so that editing can occur at the time and place where users consult the wiki (i.e., in the browser). Back to our sample, the user reads the XML article, she comes up with a new reference, and right at that moment, she has the momentum to improve the layer. Thus, the aim is to drive this impulse at the time it emerges. This calls for a seamless integration of WikiLayer within the wiki (specifically, Wikipedia) front-end, i.e. Web-augmenting Wikipedia with WikiLayer functionality. Such functionality includes layer edition, verification, maintenance and sharing. Next paragraphs look at how these operations can be conducted for Wikipedia.

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 5.6). By clicking on this tab, the *Read* and the *Edit* tabs become online editors for the layer. Click the *Edit* tab. Now, users are ready to provide their wikinotes. Akin to wiki editing practices, wikinotes are specified using a template-like format. Each of the clauses is declared as a template parameter. Figure 5.6 illustrates the case of a lecturer who augments the XML article for teaching purposes. This includes: (*i*) a new *section* about JAXB obtained from the Wikipedia itself; (*ii*) a new graph about the evolution of XML as a keyword found in online job posting as provided by `www.indeed.com`

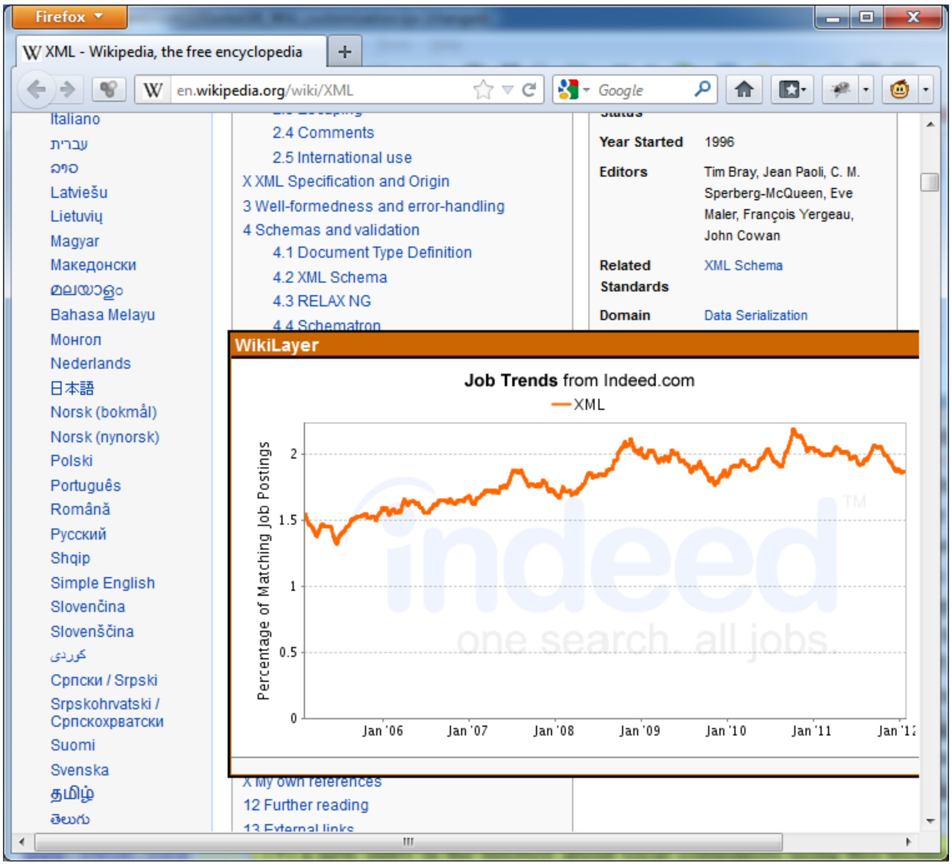


```
{{WikiLayer
| onArticle="XML"
| beforeSection="Characters and escaping"
| embedNote="extractSection('JAXB','Default data type bindings')"
}}

{{WikiLayer
| onArticle="Category:World_Wide_Web_Consortium_standards"
| beforeSection="1"
| postNote="extractFromPage('http://www.indeed.com/jobtrends?q=$article','.//td[1]/img')"
}}

{{WikiLayer
| onArticle="XML"
| afterSection="References"
| embedNote=" == Some references to help you in the lab this Friday ==\n*
[http://www.amazon.com/dp/0201771861 Processing XML with Java]  Nice book. Also
available at Uni Library\n* http://moodle3.ehu.es/mod/assignment/view.php?id=228040
Moodle link to upload your JAXB assignment"
}}

{{WikiLayer
| onArticle="XML"
| afterSection="XML Schema"
| embedNote="extractFromPage('http://zvon.org/comp/m/xml_schema.html','.//dl')"
}}
```

Figure 5.6: The tab *Edit* in the "WikiLayer" mode. Wikinotes are specified using a template while layer verification is conducted on showing preview.

(Figure 5.7); (*iii*) an extension of the *References* section with commented bibliographical entries (in terms of wikitext); and (*iv*) an extension to the "XML Schema" subsection with links to a tutorial about this topic at `www.zvon.org`. This wikinote can be downloaded 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 wikinotes. Any syntactical error will be spotted and reported in the preview. If users directly go to *Save* without preview,


Firefox
XML - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/XML
Google

Italiano
עברית
ລາວ
Latviešu
Lietuvių
Magyar
Македонски
മലയാളം
Bahasa Melayu
Монгол
Nederlands
日本語
Norsk (bokmål)
Norsk (nynorsk)
Polski
Português
Română
Русский
Shqip
Simple English
Slovenčina
Slovenščina
کوردی
Српски / Srpski
Srpskohrvatski / Српскохрватски
Suomi
Svenska
தமிழ்
తెలుగు

2.4 Comments
2.5 International use
X XML Specification and Origin
3 Well-formedness and error-handling
4 Schemas and validation
4.1 Document Type Definition
4.2 XML Schema
4.3 RELAX NG
4.4 Schematron
X My own references
12 Further reading
13 External links

Year Started	1996
Editors	Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, John Cowan
Related Standards	XML Schema
Domain	Data Serialization

WikiLayer

Job Trends from Indeed.com

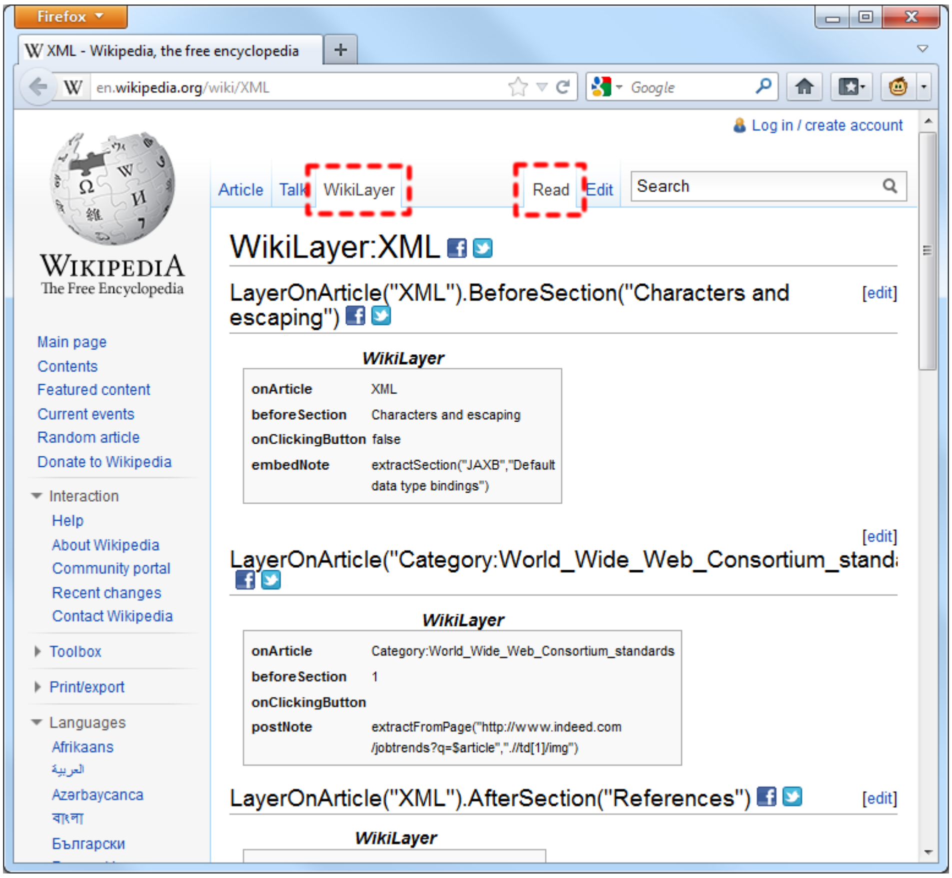
Figure 5.7: XML article enhanced with a graph from `www.indeed.com`.

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 then in the browser. This entails that the next time the page at hand is loaded, the augmentation will take place. If the wikinote extracts content from websites other than wikis, the first enactment will require to graphical set the XPath that recovers the desired HTML fragment (e.g., the job-trend graph at `http://indeed.com` in Figure 5.7). From then on, WikiLayer will seamlessly intersperse the local content (i.e., the wikinote) with the remote code (i.e., the original article).

Maintenance. Once defined, layers are prone to change. Rationales include: (*i*) layers are rooted in a moving target, Wikipedia pages (e.g., new sections, paragraphs, infobox can be added or removed, and this will likely percolate to their layer counterparts, e.g., if users refer to sections based on their numbers then, introducing a new section will no longer properly locate their layer); (*ii*) augmentation is a process. As users discover new Web resources (e.g., book references, tutorials, etc.), users might like to integrate them into a layer. WikiLayer is designed with this dynamicity on mind: wikinotes can be added/removed as easy as a paragraph of a wiki article. While on the “WikiLayer” mode, users can see which wikinotes operate on this article by clicking the *Read* tab. Figure 5.8 shows the output for our sample problem. The look mimics wiki pages but they are not wiki pages. From then on, users can add/remove wikinotes at their wish, since the layer is generated on the fly.

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 users can easily share them through Twitter or Facebook. Figure 5.9 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 have WikiLayer installed.



WikiLayer:XML

LayerOnArticle("XML").BeforeSection("Characters and escaping") [edit]

WikiLayer

onArticle	XML
beforeSection	Characters and escaping
onClickingButton	false
embedNote	extractSection("JAXB","Default data type bindings")

[edit]
LayerOnArticle("Category:World_Wide_Web_Consortium_stand

WikiLayer

onArticle	Category:World_Wide_Web_Consortium_standards
beforeSection	1
onClickingButton	
postNote	extractFromPage("http://www.indeed.com/jobtrends?q=$article","//td[1]/img")

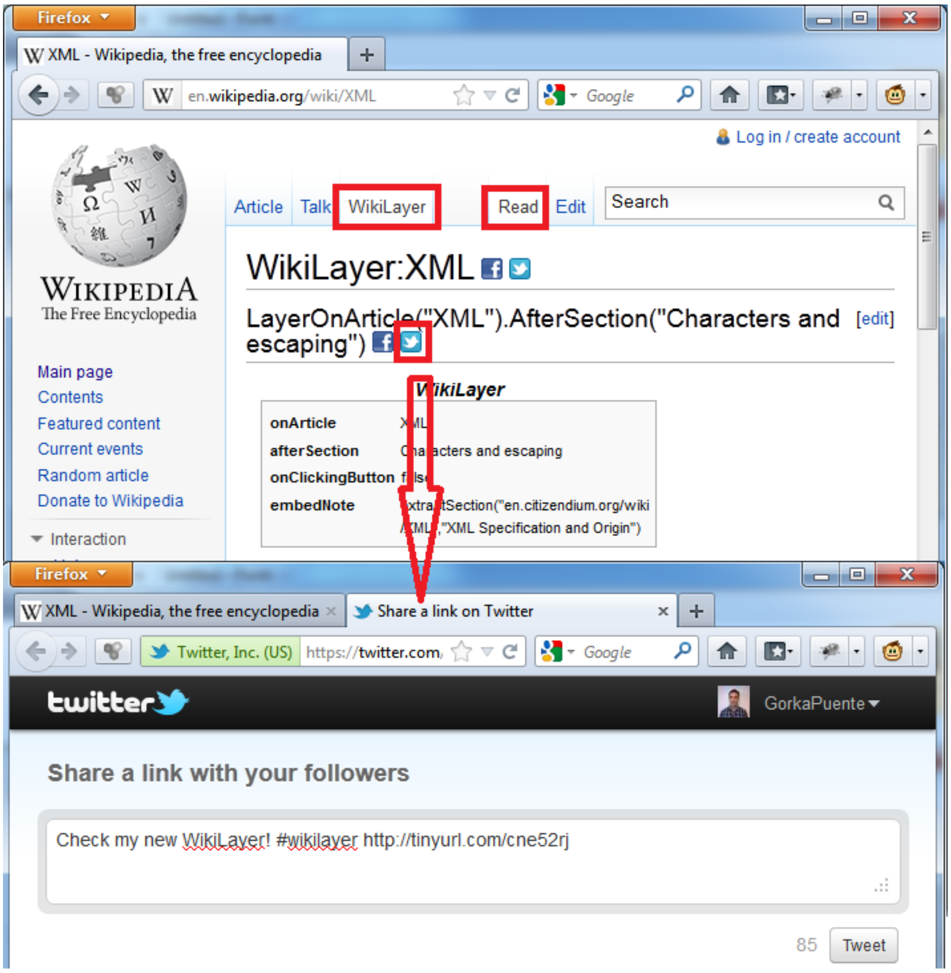
LayerOnArticle("XML").AfterSection("References") [edit]

WikiLayer

Figure 5.8: The tab *Read* in the “WikiLayer” mode. “Layer pages” kept the wiki look, although there are dynamically generated from scripts.

5.5 Related Work

This chapter can be framed within the area of Personal Knowledge Management in wikis. Along the knowledge creation spiral steps [NT95], traditional wikis support authoring and collaboration to a high extent, while semantic wikis strive to enhance knowledge reuse, finding and reminding existing knowledge through semantic annotations [OVBD06]. In addition, increasing attention is being drawn to the need of users to manage and combine both shared and personal knowledge. Users require means for creating, combining and adapting information in an isolated and guesstimated way before eventually sharing the outcome with their



WikiLayer:XML

LayerOnArticle("XML").AfterSection("Characters and escaping")

WikiLayer

onArticle	XML
afterSection	Characters and escaping
onClickingButton	false
embedNote	extractSection("en.citizendium.org/wiki/XML","XML Specification and Origin")

Share a link with your followers

Check my new WikiLayer! #wikilayer http://tinyurl.com/cne52rj

Figure 5.9: The *Read* tab in the “WikiLayer” mode. Clicking the Facebook/Twitter icon pops up a window for editing a message. The message includes a tiny URL generated, which represents the layer. By clicking on this URL, receivers can install the layer right away.

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 personal content can be shared in a pair-like way and not central server exists) to versioning-like architectures where users can start by checking out from a traditional wiki, create their own branch and then merge the content back. We compare these different approaches along a set of dimension (see Table 5.1): the *contribution*

Table 5.1: Personal Knowledge Management: wiki-based approaches.

	Contribution type	**Scope**	**Architectural style**
WikidPad	wiki	intra-wiki	standalone
P-Swooki	semantic annotation	intra-wiki	P2P
Hatta	wiki	intra-wiki	client-server
Federated Wiki	wiki/article	inter-wiki	P2P
WikiLayer	item	inter-wiki	standalone

type, what is the user contribution: a whole wiki, an article, an item, a semantic annotation, etc.; the *scope*, whether the reach is limited to a single wiki or expand across different wikis; and finally, the *architectural style* (i.e., standalone, client/server or P2P). The goal is not only to provide an exhaustive list of endeavours but outline the main differences with WikiLayer. For completeness sake, the comparison also includes Web annotation tools.

Web Annotation Tools. Being a Web applications, wikis can benefit from Web annotation tools such as *Diigo* or *A.nnotate*. Nothing wrong with it. However, the premise is that wikis (unlike other websites) should look at annotation as a mean to achieve their very own goals: reading and editing articles. While annotating a University website is not directly related with the University goals (i.e., educating), wikis are all about engaging the crowd in article contribution. From this perspective, annotation is no longer an ancillary activity but a main mean to fulfil wiki's ends. By using general-purpose Web annotation tools, opportunities brought by a wiki-specific annotation tool are missed: 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 wiki users. WikiLayer is an attempt in that direction.

Personal Wikis. A personal wiki is like a traditional wiki but with a single user. *WikidPad*[6] is a case in point[7]. 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 [TSMDM09]: 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. 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 (i.e., tags) in P-Swooki *versus* items (e.g., sections) in WikiLayer. Moreover, 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

[6] `http://wikidpad.sourceforge.net/` (accessed December 2012).

[7] For a list, refer to *`http://c2.com/cgi/wiki?PersonalWiki`* (accessed December 2012).

revision control utilities like *SVN* or *Git*. Generally, the engine can be installed locally, and periodically (normally on demand) synchronized with the server. 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. a check out), which includes the page history. Examples include *Hatta*[8], or *Firestarter*[9] for *Confluence*[10]. 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 systems whereby a new project is initiated out of a base one. From this perspective, WikiLayer can be regarded as a lightweight wiki-oriented approach to article versioning, although 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 to 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. Pioneered by Ward Cunningham [Cun11], 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

[8] `http://hatta-wiki.org` (accessed December 2012).

[9] `www.appfire.com/enterprise/firestarter` (accessed December 2012).

[10] `www.atlassian.com/software/confluence` (accessed December 2012).

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 an article to your wiki. From then on, the original article and the clone have different lifecycles, although users 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 the personal view on top of it. Edition and sharing are conducted within a personal realm: sharing through friends in Facebook or followers in Twitter, and edition through a transparent local repository.

5.6 Conclusions

This chapter spur on the use of WikiLayer as an augmentation facility targeted to wikipedians, although equally valid for users of any MediaWiki powered wiki. 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 engage wiki users: layer design is along wiki concepts (e.g., section, wikitext), layer syntax resorts to wiki templates, and layer management is achieved through wiki-like templates. 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.

However, some premises need yet to be demonstrated. Thus, the next follow-on is to conduct validation experiments to check the following hypothesis: (*i*) readers using layers are more inclined to become editors, (*ii*) wikipedians look at layer-enhanced wikis as appropriate hubs to collect web-based material, and (*iii*) layer-based editing surveillance leads to more frequent monitoring and hence play the advantage of the quality of the monitored articles.

Parts of this chapter have already been published:

- Oscar Díaz, Cristobal Arellano, Gorka Puente, "Wikipedia Customization through Web Augmentation Techniques". In *8th International Symposium on Wikis and Open Collaboration* (*WikiSym'12*), Linz, Austria, 2012. Acceptance rate 55%.

Chapter 6

Conclusions

"Real knowledge is to know the extent of one's ignorance."

– *Confucius*

6.1 Overview

Corporate wikis have become commonplace for collaborative knowledge formation and sharing. However, the very same wiki nature hinders its success if the main wiki design principles [Cun06] are not kept: initial wiki deployment should be *simple*; the wiki *organic* growth should not impede wiki maintenance and management; and individuals should be able to *share* their personal knowledge. This dissertation undertakes the challenge of providing support for users in following these principles.

This chapter reviews the main results of this work, assesses its limitations, and suggests work for future research.

6.2 Results

This dissertation developed the content of the research into four main chapters, whose contributions are detailed next:

- *Chapter* 3 introduces the notion of "Wiki Scaffolding" as a means for corporate strategies to permeate wiki construction. "Wiki Scaffolding" is realized as mind mapping to preserve wikis' openness. The result is WSL, a visual DSL on top of FreeMind. Thus, non-technical communities are enabled to facilitate the alignment of the wiki with organizational practices, promoting management engagement, enhancing the visibility of the wiki's practices, and promoting employee participation through direction setting.

- *Chapter* 4 paves the road for "Wiki Refactoring" to knowledge workers (i.e., wiki contributors). This chapter permits the layman to easily and reliably express refactoring processes. This vision is realized into WikiWhirl, a DSL built on top of FreeMind. Hence, wiki users import wikis as mind maps, perform refactoring operations as reshaping of mind map nodes, and commit these changes back to the wiki. In addition, WikiWhirl preserves authorship and readership during the whole process, improving the productivity, global understandability and automatic following of the refactoring good practices.

- *Chapter* 5 encourages wiki users to enhance wikis with their personal knowledge or perspective. In so doing, this chapter presents WikiLayer as a customization and private edition tool. 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 wiki users: layer design is along wiki concepts (e.g., section, wikitext), layer syntax resorts to wiki templates, and layer management is achieved through wiki-like templates.

- *Appendix* A describes another main contribution: Schemol, a DSL for harvesting models out of databases. Schemol originated as part of our endeavours to manipulate wiki content in terms of models rather than tuples. Indeed, WikiWhirl rests on Schemol for model extraction. This motivates the inclusion of Schemol as part of this dissertation. Nevertheless, its scope goes beyond wikis, and this explains it is not a chapter but an appendix. Specifically, this appendix focuses on Schemol to extract models from wiki databases and its special features for Web2.0 specifics.

- The rest of the appendixes are not contributions as such but supporting material.

6.3 Publications

Parts of the work explained in this thesis have been already presented and discussed in distinct peer-reviewed forums. The author has co-authored several publications, which have been backed by different communities:

Information Systems

- Gorka Puente, Oscar Díaz, Maider Azanza "Refactoring Affordances in Corporate Wikis: A Case for the Use of Mind Maps", In *Enterprise Information Systems* journal, 2013. **JCR**, Impact factor 3.684. **Under review**.

- Oscar Díaz, Gorka Puente. "Wiki Scaffolding: Aligning Wikis with the Corporate Strategy". In *Information Systems* journal, 2012. **JCR**, Impact factor 1.595 [DP12].

- Gorka Puente, Oscar Díaz, "Wiki Refactoring as Mind Map Reshaping". In *24th International Conference on Advanced Information Systems Engineering* (*CAiSE'12*), Gdansk, Poland, 2012. Acceptance rate 14% [PD12]

- Oscar Díaz, Gorka Puente. "A DSL for Corporate Wiki Initialization". In *23rd International Conference on Advanced Information Systems Engineering* (*CAiSE'11*), London, UK, 2011. Acceptance rate 13%. **Best paper award** [DP11a].

Model Driven Engineering

- Oscar Díaz, Gorka Puente, Javier Luis Cánovas Izquierdo, Jesús García Molina, "Harvesting Models from Web 2.0 Databases". In *Software and Systems Modeling* journal, (*SoSyM*), 2011. **JCR**, Impact factor 1.533 [DPCIGM11].

Wikis and Open Collaboration

- Oscar Díaz, Cristobal Arellano, Gorka Puente, "Wikipedia Customization through Web Augmentation Techniques". In *8th International Symposium on Wikis and Open Collaboration* (*WikiSym'12*), Linz, Austria, 2012. Acceptance rate 55% [DAP12].

- Oscar Díaz, Gorka Puente, "Wiki Scaffolding: Helping Organizations to Set Up Wikis". In *7th International Symposium on Wikis and Open Collaboration* (*WikiSym'11*), Mountain View, California, USA, 2011. Acceptance rate 42% [DP11b].

- Oscar Díaz, Gorka Puente, Cristóbal Arellano, "Wiki Refactoring: an Assisted Approach Based on Ballots". In *7th International Symposium on Wikis and Open Collaboration* (*WikiSym'11*), Mountain View, California, USA, 2011. Acceptance rate 42% [DPA11].

- Oscar Díaz, Gorka Puente, "Model-Aware Wiki Analysis Tools: the Case of HistoryFlow". In *6th International Symposium on Wikis and Open Collaboration* (*WikiSym'10*), Gdansk, Poland, 2010. Acceptance rate 39% [DP10].

Others

- Oscar Díaz, Gorka Puente, "Integrando la Wiki dentro de la Empresa", In *1st Congreso de Empresa 2.0 y Social Business (e20biz'12)*, Sevilla, Spain, 2012.

- Oscar Díaz, Gorka Puente, "Wiki Scaffolding: Framing Wiki Contributions along the Organization Concerns", Invited post in the blog *Follow the Crowd*. 2011.

- Javier Luis Cánovas Izquierdo, Oscar Díaz, Gorka Puente, Jesús García Molina, "ScheMoL: Un lenguaje específico del dominio". Demo tool. In *XVI Jornadas de Ingeniería del Software y Bases de Datos (JISBD'11)*, A Coruña, Spain, 2011

- Oscar Díaz, Gorka Puente, "A DSL for Corporate Wiki Initialization". Demo Tool. In *XVI Jornadas de Ingeniería del Software y Bases de Datos (JISBD'11)*, A Coruña, Spain, 2011

- Gorka Puente, "Wiki Reengineering". Invited talk. University of Malaga. 2010.

- Gorka Puente, Oscar Díaz, "Wiki Reengineering". In *3nd Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE'09)*, Braga, Portugal, 2009.

6.4 Research Visits

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 two research groups at the forefront of research in MDE, for one week each time. First, the *ModelUm Research Group* at the University of Murcia, headed by Prof. Jesús García Molina, and later, the *Atenea Systems Modeling Group* led by

Prof. Antonio Vallecillo. After those experiences, the author performed a longer research visit from April to June of 2011 to the *Intelligent Web and Information Systems Research Group* (*IWIS*) under the supervision of Peter Dolog. This dissertation has been certainly influenced for those research visits, which fostered new and unexpected insights, discussions and perspectives.

6.5 Assessment and Future Research

The work presented in this dissertation introduces, motivates and proposes approaches for the different stages a wiki may undergo: wiki initialization, wiki refactoring and wiki customization. However, an objective assessment exposes some limitations of this work, which motivates areas of extension and future improvement.

Wiki Initialization

- *Suitability*. According to ISO9126 [ISO], suitability is a functionality characteristic that refers to the appropriateness (to specification) of the functions of the software. WSL constructs are based on a literature survey about the use of wikis in companies. Nonetheless, the use of corporate wikis is at its inception. It is likely that social conventions and incentives will emerge and evolve to guide contributors, resolve disputes and help manage wikis. Hence, WSL constructs will need to be extended, as well as its functionality characteristics. As an example, consider a new feature that was not considered at WSL inception but properly suggested during a WSL demonstration: the automatic introduction of users from the company directory service (e.g., LDAP).

- *Validation*. Additional evidences are needed to claim "Wiki Scaffolding" succeed on better aligning wikis to corporate strategies as well as on engaging users. At the moment of writing this thesis, WSL is being deployed in a company to collect evidences and feedback about the advantages brought by the scaffolding.

- *Evolution*. The WSL engine raises evolution concerns. WSL might be affected by (*i*) changes in MediaWiki (or its extensions), (*ii*) changes in the underlying database schema (what impacts on the MOFScript transformation), and (*iii*) changes in the FreeMind metamodel (what impacts on the XSLT transformation). This challenges WSL maintenance. However, both FreeMind and MediaWiki are stable platforms backed by thousands of installations. In addition, wikis can be upgraded once deployed. Remember that WSL is used just to initialize the wiki. Once the scaffolding is deployed, users can upgrade the wiki to the newest version. From this moment on, the wiki will certainly evolve, this was the primary reason of the following work, "Wiki Refactoring".

- *Scalability*. Large wiki projects require large scaffoldings. This can lead to cluttered WSL maps. Fortunately, FreeMind offers view-like mechanisms that permit to filter map nodes based on content and relationships. Another option would be to study the possibility of initializing wikis in phases, with the consequent reduction in the mind maps size.

- *Learnability*. This usability characteristic refers to the learning effort required to use WSL. This has been a main concern during WSL design. We strive to make the draw of a "Wiki Scaffolding" as easy as possible. However, the presumption that is easy to use has still to be proven by the wiki community.

Wiki Refactoring

- *Suitability*. As future work, new empirical evidences should be collected about the usefulness of WikiWhirl, and to extend WikiWhirl to other wiki engines. Specifically, to Confluence and

DokuWiki, as the most used wiki engines, together with MediaWiki. Support for DokuWiki would represent an additional challenge since it stores wiki content in files as opposed to a database.

- *Validation*. Refactoring is inherent to wikis' organic growth. Although traditional refactoring techniques can be interpreted in wiki terms, the lack of appropriate tests for refactoring safeness makes the intervention of the users necessary. Based on this observation, proofs of community acceptance of the refactoring should be collected, and with that feedback, improve the WikiWhirl functionality. In addition, more evaluations are needed, specifically, with a larger group within a corporate setting. Our intuition leads us to think that productivity and the preservation of the authorship and readership will be more appreciated in organizations.

- *Scalability*. In this case, there might be also problems when coping with large wikis. The average corporate wikis have an estimate of up to 1,500 pages (i.e., nodes) [SB09]. In these numbers, FreeMind still works properly, but the human eye starts to have difficulties to effectively distinguish refactoring opportunities. Even though, FreeMind has mechanisms that help in this situation (e.g., filters, node folding, scripts, etc.), WikiWhirl could provide a feature to allow working only with partial views of the wiki e.g., only representing the descendant hierarchy of the XML category.

- *Learnability*. WikiWhirl has fewer constructors than WSL, which makes it easier to learn. Nevertheless, it still needs to be configured with initial database parameters. Although we spent time to reduce the configuration effort, it still requires further work.

Wiki Customization

- *Learnability*. WikiLayer design is addressed to wiki users with the intention of reducing the learning effort. WikiLayer is completely hidden when in the wiki (e.g., Wikipedia), giving the impression that the additions are in fact part of the wiki page. In addition, the creation of a wikinote is not but the same process of editing a wiki template. However, this premise needs to be validated by the wiki users to check the real learning effort required.

- *Validation.* WikiLayer still needs to be complemented with user validation test and prove the next conjectures: (*i*) WikiLayer boosts readers to step further and become editors, (*ii*) augmented wikis represent a useful mechanism to collect web-based personal knowledge for users, and (*iii*) the use of layers increases the supervision of articles what leads to regular monitoring, therefore enhancing their maintenance.

6.6 Conclusions

This dissertation has shown how to cope with the accidental complexity that lies with wikis: wiki initialization requires technical expertise, wiki refactoring implies a global view and in-depth wiki knowledge and wiki customization was not even an option. The different solutions to solve each challenge have the practical and engineering approach that characterizes the thesis carried out within Onekin.

Nevertheless, while distinct experiences provide evidences of the benefits when combining wikis and mind maps, the presented approaches have still to demonstrate that really pay off for real companies. The author is eager to assess the presented ideas in the harshest setting: the real market, where customers will provide the acid test. This will certainly imply moving from prototypes to products and from users to customers as the target audience.

Appendix A

Harvesting Models from Wiki Content[1]

"If people never did silly things, nothing intelligent would ever get done."

– *Ludwig Wittgenstein.*

A.1 Overview

Data rather than functionality are the sources of competitive advantage for Web2.0 applications such as wikis, blogs and social networking websites. This valuable information might need to be capitalized by third-party applications or be subject to migration or data analysis. MDE can be used for these purposes. However, MDE first requires obtaining models from the wiki/blog/website database (a.k.a. model harvesting).

This chapter introduces Schemol, a DSL tailored for extracting models out of databases which considers Web2.0 specifics. Schemol and some examples are available to download at `www.onekin.org/schemol`. Specifically, the aim of this apendix is to provide a brief on how Schemol

[1] Parts of this chapter have been previously presented [DPCIGM11].

extracts models out of wikis (for a complete description of Schemol refer to [DPCIGM11]). First, it is shown the original motivation for Schemol (Section A.2). Then an example is used to give a brief on Schemol (Section A.3). After that, the chapter moves to the application of Schemol to wikis (Section A.4). Related work (Section A.5) and conclusions (Section A.6) finish the chapter.

A.2 Web2.0 as Motivation of Schemol

Web2.0 applications revolve around user-provided data [ORe05]. Wikis, blogs, and tagging sites are the major exponents of this kind of applications (hereafter referred to as Web2.0 applications). Understanding this data then becomes key to capitalize and promote further user participation, hence, helping Web2.0 applications to thrive. In wikis, all related activities form a specific information space within the organization. This information space consists of different networks or perspectives [Car03]: social perspective (i.e., who knows who), knowledge perspective (i.e., who knows what), information perspective (i.e., what refers to what), and temporal perspective (i.e., what was done before). Broadly, these studies pose a hypothesis based on a model for the Web2.0 application at hand; next, collect the data for these applications, and finally, analyze the data so-collected to validate the hypothesis. The model for the Web2.0 application tends to be a mathematical model (e.g., a graph), which is represented as a data structure, and the process of populating this data structure tends to involve tedious programming. Such effort has three main drawbacks:

- First, it makes domain experts (e.g., frequently coming from social sciences) struggle with involved code, hence deviating efforts from analyzing the data as such.

- Second, it hinders evolution: the common cycle *set hypothesis / check data / reset hypothesis* commonly involves changes in the model which in turn, implies laboriously changing the data extracting programs.

- Third, it jeopardizes interoperability. Most analysis and visualization tools focus on large wikis such as Wikipedia where efficient handling of large bulk of data is key. This forces a tight coupling between the tool and the underlying wiki engine (e.g., MediaWiki) so that the terms of analysis (e.g., nodes, edges) tend to be coupled to how these notions are realized in the wiki (e.g., wiki pages, wiki users). Hence, it is not uncommon for tools to bind domain concepts to wiki notions (on an efficiency basis). For example, if the tool binds nodes to wiki articles then, analysts are prevented from using that tool to study other wiki concepts where nodes could have stood for other notions (e.g., wiki users, revisions, etc.). This is unfortunate, since the very same data could be studied from different perspectives or using different visualization metaphors.

We advocate for making analysis tools model aware. That is, tools are characterized in terms of their abstract analysis models (e.g., a graph model, a contributor model, a collaboration model). How this analysis model is then map into wiki implementation terms is left to the users who, as the domain experts, can better assess which is the right granularity to conduct the analysis. Unfortunately, this first requires obtaining models from the wiki database (a.k.a. model harvesting). This can be achieved through SQL scripts embedded in a program. However, this approach requires of powerful extractive techniques that permit the obtention of models out of wikis. Hence, the first question to answer is how wikis store the content. As already commented in the background, the most common repositories of wiki content are the relational Database Management Systems (DBMS) (e.g., MediaWiki uses MySQL or *PostgreSQL*). In this regards, there are two main differences between wiki databases and traditional databases, namely:

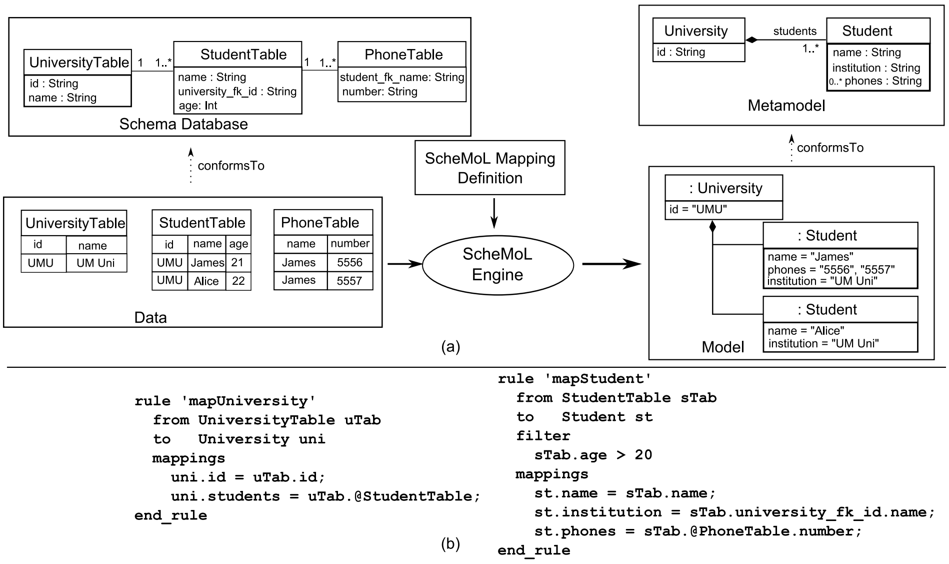
1. Wikis do not have their own database schema. That is, all wikis using the MediaWiki engine are stored using the very same schema. Of course, this is not the case for traditional database applications where, let's say, Oracle, does not restrict the tables of your database. Therefore, table names offer little help in automating the extractive process.

2. Data tend to be annotated. Annotation is about attaching additional information (i.e., metadata) about an existing piece of data. Unlike "traditional" databases, wiki data are likely to be annotated with HTML tags, CSS classes, RDFa annotations or templates. Annotations can provide valuable clues about hidden entities. For instance, you can use templates to structure the input/rendering of addresses in your wiki. Address fields (e.g., street, zip code, etc.) become template parameters. These parameters provide valuable clues about hidden entities/attributes that might need to be surfaced when conducting an analysis based on contribution patterns based on the zip code. The problem is that templates (as well as any other annotation) are stored together with the data as opaque strings, and transparent to the DBMS.

By contrast, a DSL can hide these *how* concerns, leaving the designer to focus on the *what*, i.e., the mapping of database schemas to model classes. With this idea was conceived Schemol. Schemol is a domain-specific language tailored for extracting models out of databases. Next section gives a brief on Schemol.

A.3 A Brief on Schemol

Schemol strives to mimic current model-to-model transformation languages such as *ATL* [JABK08] or *RubyTL* [CMT06]. In these languages, mappings are specified as correspondences between elements of the source and target metamodels. Likewise, Schemol describes model



```
rule 'mapUniversity'
  from UniversityTable uTab
  to   University uni
  mappings
    uni.id = uTab.id;
    uni.students = uTab.@StudentTable;
end_rule
```

```
rule 'mapStudent'
  from StudentTable sTab
  to   Student st
  filter
    sTab.age > 20
  mappings
    st.name = sTab.name;
    st.institution = sTab.university_fk_id.name;
    st.phones = sTab.@PhoneTable.number;
end_rule
```

Figure A.1: Model harvesting in Schemol

harvesting as a mapping between elements of the source database schema, and elements of the target metamodel.

Figure A.1a shows a simplified example. The database schema defines the *UniversityTable*, *StudentTable* (*university_fk_id* is foreign key) and the *PhoneTable* (*student_fk_name* is foreign key). The target metamodel captures universities (*University* metaclass) and students (*Student* metaclass). The transformation has two inputs: (*i*) the database itself, and (*ii*) the Schemol transformation (i.e., a set of transformation rules).

A transformation rule specifies the mapping between a database table and a class of the target metamodel. This is achieved through four clauses (see Figure A.1b):

1. the *from* part, which specifies the source table together with a variable (e.g., *uTab*) that will hold the tuple of this table at the time the rule is applied,

2. the *to* part, which specifies the target element metaclass as well as a variable (e.g., *uni*) to hold the instance being generated at enactment

time,

3. the *filter* part (optional), which includes a condition over the source element. The rule will only be triggered if the condition is satisfied (e.g., *sTab.age > 20* filters students over 20 years old),

4. the *mapping* part, which contains a set of *bindings* to set the attributes of the target element (e.g., *uni.id = uTab.id*).

A *binding* establishes the relationship between a source table and a target metamodel element (i.e., the '=' operator). The left-hand side must be an attribute of the target element metaclass. The right-hand side can be a literal value, a query or an expression. The rules in Figure A.1 shows different scenarios.

Rule '*mapStudent*'. The rule states that for each *StudentTable* tuple, a *Student* model element is to be generated. The bindings indicate how the distinct *Student* attributes are obtained from table columns. Two scenarios are possible:

- If the right-hand side of the binding is a literal value then, the value is directly assigned to the attribute specified in the left-hand side. Example: *"st.name = sTab.name"* readily fills the *name* attribute from the namesake column.

- If the right-hand side of the binding is a query then, the query is executed. To this end, Schemol interprets foreign keys as "object references", hence, amenable to be navigated through. Navigation can be forward or backward. This is one of the highlights of Schemol in comparison with ATL or RubyTL. The expression "*fromId.colName1. colName2*" denotes a forward navigation. The query retrieves the value of *colName2,* provided *colName1* is a foreign key. Otherwise, an error is raised. By contrast, the expression *"fromId.@tableName"* expresses a backward navigation. This query retrieves those tuples at *tableName* that refers to the *fromId* tuple (i.e., they have a foreign key from *tableName* table to the table that contains the*fromId* tuple).

For instance, in the expression "*st.phones = sTab.@PhoneTable. number*" the query "*sTab.@PhoneTable.number*" obtains the *set* of phones at *PhoneTable* for the student held by the *sTab* variable. Note that distinct tables can be collapsed into a single class (e.g., *StudentTable* and *PhoneTable* feed the class *Student*). Here, recovering the value for phones requires table joins.

Rule '*mapUniversity*'. This rule creates a *University* model element for each tuple of the *UniversityTable* table. In this case, a new scenario is possible:

- The left-hand side attribute is a reference, and the query returns a tuple set then, the engine returns the model elements of the tuple counterparts. If not yet created, returned model elements are built on the fly by triggering the appropriate rules. Example: *"uni.students = uTab.@studentTable"*. Here, the binding is not a mere assignment but Schemol should turn the backward navigation (i.e., *uTab. @studentTable*) into its reference counterpart value, so that *students* stand for a *model* association.

The following section shows the specificities of Schemol for wikis.

A.4 Schemol for Wikis

As previously mentioned, Web2.0 databases tend to have annotated content. Although transparent to the database, annotations might need to be surfaced in the target metamodel. This section first makes the case by introducing common approaches to self-descriptive content in Web2.0. Next, we introduce new functions in the Schemol query language to surface these annotation-based elements.

As for wikis, semantic wikis have been proposed that *"provide the ability to capture or identify information about the data within pages, and the relationships between pages, in ways that can be queried or exported like a database"* [SBBK08]. *Semantic MediaWiki* is an extension to MediaWiki, which allows for annotating semantic data within wiki pages [Sem]. Although not so sophisticated as semantic wikis, wiki templates also illustrate this idea of self-descriptiveness. In some cases, wiki pages or page sections exhibit some common structure/rendering that can be explicitly captured through wiki templates. The template defines the common content that can be parameterized through the so-called *template parameters*. A wiki page can then refer to this template, providing the actual values for the parameters. An example follows:

> *{{Acknowledgments |project*=TIN2008-06507-C02-01*/TIN*
> *|PhD student=Gorka Puente}}*

A page with this annotation would use the "*Acknowledgments*" templates, which is parameterized along "*project*" and "*PhD student*" parameters. What you really get when this page is rendered could look like:

> *We would like to acknowledge the support of the project TIN2008-06507-C02-01/TIN. Gorka Puente has a doctoral grant from the Spanish Ministry of Science & Education.*

Commonly cited advantages of wiki templates include: enforcing a uniform layout, ensuring that all available relevant information is provided, lack of information is made explicit, and easing comparison of wiki pages. This list can now be extended to include self-descriptiveness, since Schemol provides a function to exploit the wiki templates parameters. The function *parameter(templateName, parameterName)*, which applied to a string, returns the content of the wiki template parameter *parameterName* from the template *templateName*. This permits model elements (or attributes of model elements) to be obtained from wiki templates.

Case Study: Wiki Articles as Use Cases

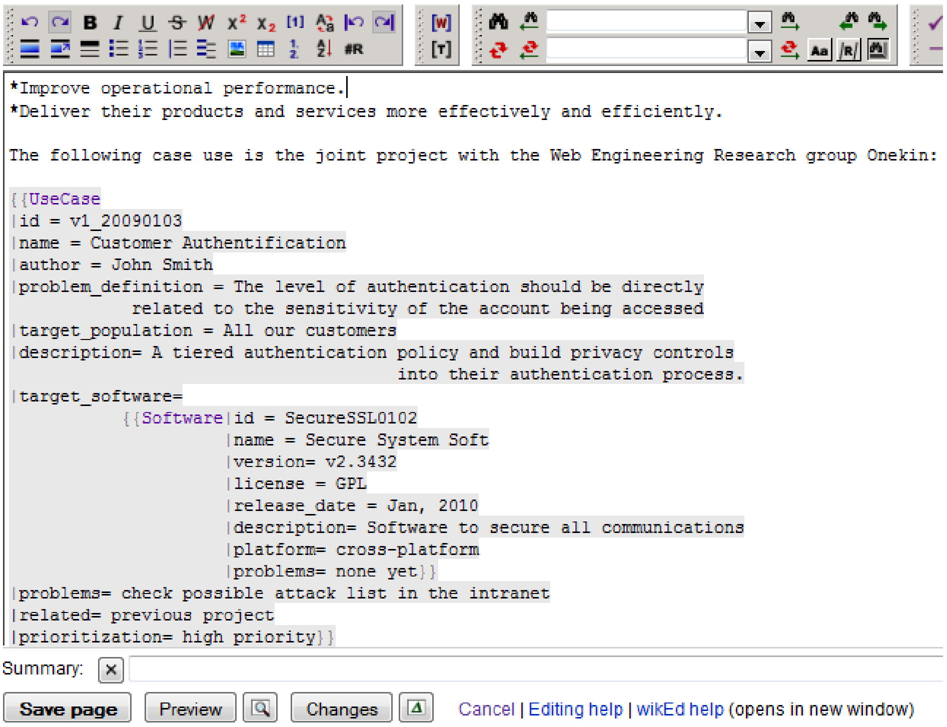
Wikis have been proposed to collect software requirements from stakeholders [Lou06]. Stakeholders become wiki users whereas use cases (a common mechanism for capturing system requirements) are realized as wiki articles. In this scenario, Schemol can serve to extract models from wiki articles that are then liable to be transformed to other models (e.g., being enriched with additional information through model composition), or to code (e.g., using model-to-text transformation languages). The latter can be useful to help wikis to interoperate with other tools. For instance, wikis are used to collectively create use cases which are later used to feed editors using XMI.

It comes as no surprise that wiki templates are available to ensure that all relevant information is provided. This case study incorporates the template defined by the W3C at [w3c]. This template (see Figure A.2) captures use cases in terms of impact (e.g., target audience or improvement of a series of aspects), tools expected to implement these recommendations, etc.

Source database schema. A partial view of MediaWiki database schema is depicted in Figure A.3 (left part). The use case shown in Figure A.2 is stored as textual content (i.e., a string) of the attribute *old_text* at the *text* table.

Target metamodel. A *UseCase* metamodel is introduced (Figure A.3, right part): a *Company* is an aggregate of *Projects* and *Employees*. A *Project* comprises distinct *UseCases* each one of which is implemented by some *Software*.

Transformation. It begins (see Figure A.4) by creating a *Company* element where *itsEmployees* and *itsProjects* are obtained from the *user* and *page* tables, respectively (lines 2-9). Each *user* tuple gives rise to an *Employee* class (lines 11-16). By contrast, only pages whose category is *"Projects"* (see filter at line 22) will be turned into a *Project* class. For each project page, the transformation looks at its hyperlinks (collected



```
*Improve operational performance.
*Deliver their products and services more effectively and efficiently.

The following case use is the joint project with the Web Engineering Research group Onekin:

{{UseCase
|id = v1_20090103
|name = Customer Authentification
|author = John Smith
|problem_definition = The level of authentication should be directly
            related to the sensitivity of the account being accessed
|target_population = All our customers
|description= A tiered authentication policy and build privacy controls
                                        into their authentication process.
|target_software=
            {{Software|id = SecureSSL0102
                      |name = Secure System Soft
                      |version= v2.3432
                      |license = GPL
                      |release_date = Jan, 2010
                      |description= Software to secure all communications
                      |platform= cross-platform
                      |problems= none yet}}
|problems= check possible attack list in the intranet
|related= previous project
|prioritization= high priority}}
```

Figure A.2: Editing a wiki article. Notice the use of two templates, *UseCase* and *Software*, for capturing use cases and software descriptions, respectively.

in the *mypagelinks* table, see line 25) which hold foreign keys to related wiki articles. For those related articles categorized as *"UseCases"* (line 40), a *usecase* model is obtained. However, wiki articles are kept as mere strings held into the *old_text* column. For *usecase* articles, we know this string is framed by a wiki template. On this premise, we use the function provided by Schemol *parameter(templateName, parameterName)*. This function makes us perceive wiki templates as "nested tables", and template parameters as table columns. Lines 42-49 illustrate how template parameters *name*, *description* and *project_leader* are dug out to fill their model property counterparts.

Interestingly, template parameters can use other templates to facilitate their specification. The *"target_software"* parameter is a case in point.

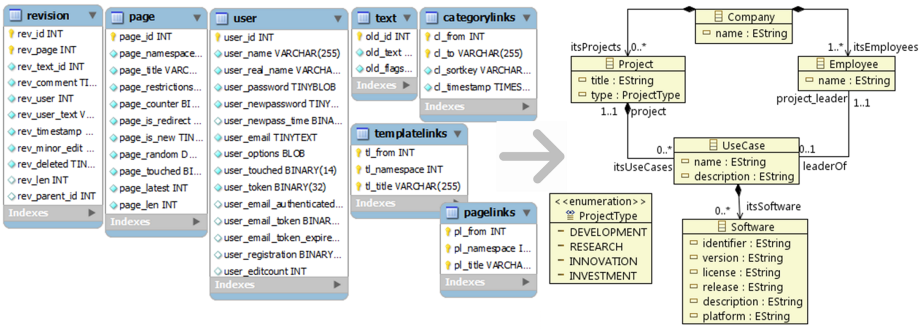
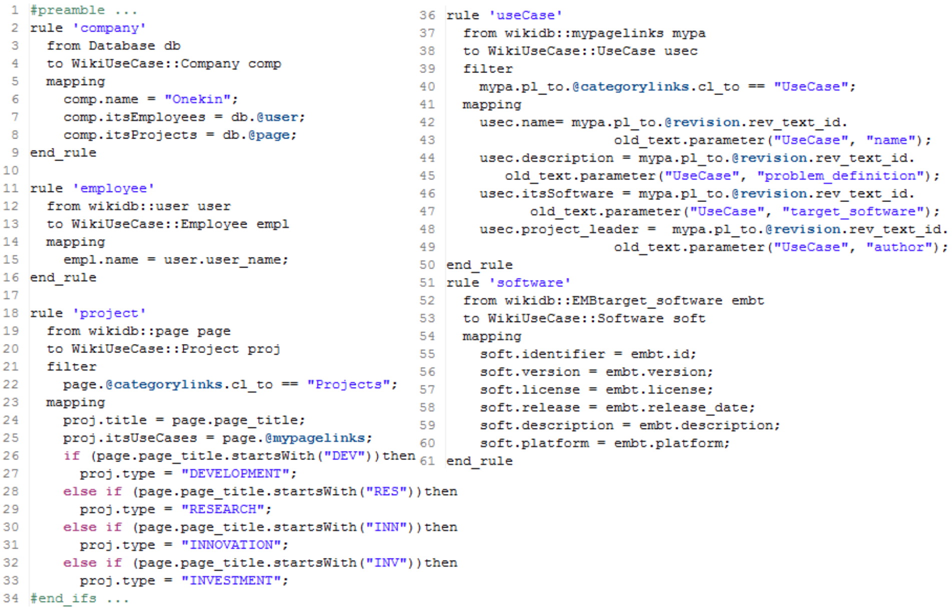


Figure A.3: MediaWiki database schema (partial view) and *UseCase* metamodel.

This parameter does not hold an "atomic" value but a *UseCase* template. In this case, the expression "*old_text.parameter("UseCase", "target_software")*" (lines 46-47) does not yield a string but a tuple. Better said, since a project can involve distinct software packages, the previous expression can potentially retrieve a set of tuples. Lines 46-47 result into a *model* association (i.e., *itsSoftware*) between a use case model and its software packages. According to the binding semantics, this assignment leads to the triggering of the '*software*' rule (lines 51-61). The '*software*' rule is worth looking at. It handles the transformation of the template *target_software* which is embedded as part of the *old_text* column. Template instances are conceived as tuples of "nested tables". To refer to this nested table, Schemol creates a table name after the name of the template plus the prefix *EMB* (e.g., *EMBtarget_software).* This term can then be used in a rule's *FROM* (line 52). Similar functions are available to handle other hidden annotations in Web2.0 resources.

This example accounts for 75 LOC. The JDBC counterpart involves around 400 LOC. As for other DSLs, the gains are not only on development but also on maintenance and understanding.



```
#preamble ...
rule 'company'
  from Database db
  to WikiUseCase::Company comp
  mapping
    comp.name = "Onekin";
    comp.itsEmployees = db.@user;
    comp.itsProjects = db.@page;
end_rule

rule 'employee'
  from wikidb::user user
  to WikiUseCase::Employee empl
  mapping
    empl.name = user.user_name;
end_rule

rule 'project'
  from wikidb::page page
  to WikiUseCase::Project proj
  filter
    page.@categorylinks.cl_to == "Projects";
  mapping
    proj.title = page.page_title;
    proj.itsUseCases = page.@mypagelinks;
    if (page.page_title.startsWith("DEV"))then
      proj.type = "DEVELOPMENT";
    else if (page.page_title.startsWith("RES"))then
      proj.type = "RESEARCH";
    else if (page.page_title.startsWith("INN"))then
      proj.type = "INNOVATION";
    else if (page.page_title.startsWith("INV"))then
      proj.type = "INVESTMENT";
#end_ifs ...
```

```
rule 'useCase'
  from wikidb::mypagelinks mypa
  to WikiUseCase::UseCase usec
  filter
    mypa.pl_to.@categorylinks.cl_to == "UseCase";
  mapping
    usec.name= mypa.pl_to.@revision.rev_text_id.
                    old_text.parameter("UseCase", "name");
    usec.description = mypa.pl_to.@revision.rev_text_id.
       old_text.parameter("UseCase", "problem_definition");
    usec.itsSoftware = mypa.pl_to.@revision.rev_text_id.
          old_text.parameter("UseCase", "target_software");
    usec.project_leader =  mypa.pl_to.@revision.rev_text_id.
                    old_text.parameter("UseCase", "author");
end_rule
rule 'software'
  from wikidb::EMBtarget_software embt
  to WikiUseCase::Software soft
  mapping
    soft.identifier = embt.id;
    soft.version = embt.version;
    soft.license = embt.license;
    soft.release = embt.release_date;
    soft.description = embt.description;
    soft.platform = embt.platform;
end_rule
```

Figure A.4: Schemol transformation for the MediaWiki case study.

A.5 Related Work

The interest in model harvesting should be sought in the increasing upheaval on software modernization, specifically when MDE techniques are used. OMG's Architecture Driven Modernization (ADM) initiative [ADM] aims to facilitate the interoperability between modernization tools by defining a set of standard metamodels for representing information involved in a software modernization process. In fact, the metamodel provided by ADM to represent software artefacts, named KDM, includes persistent data as one of the assets to be captured. In this scenario, model harvesting is a key enabler as the first step to abstract for technological platforms, and building effective bridges between TSs.

MoDisco (Model Discovery) is an extensible framework for model-driven reverse engineering, supported as an Eclipse Generative Modeling Technology (GMT) component [BBJ$^+$10]. Its objective is to facilitate the development of tools ("discoverers" in MoDisco terminology) to obtainmodels from legacy systems during modernization efforts. XML and

Java discoverers are available. Schemol could then become an additional MoDisco discoverer for relational data.

A.6 Conclusions

As MDE becomes a mainstream, model harvesting gains prominence as a key enabler to bridge other technical spaces to modelware. This chapter introduced Schemol for harvesting wiki models out databases DSL for model harvesting out of databases. This approach allowed WikiWhirl to extract models through model transformations. Hence, improving the maintenance and evolution of the tools: (i) changes on database schemas and (ii) modifications on the target models (e.g., FreeMind model) only impact on the transformations. As far as we are aware of, this work is the first that provides this perspective.

As future work, there are several challenging threads to pursue. First, we plan to improve the interoperability of Schemol by facilitating drivers for other DBMS and metamodel languages. Also, we would like to conduct experiments on the usability of Schemol. Finally, another interesting direction is to tap on existing HTML extractive methods rather than building our own in Schemol.

Parts of this chapter have already been published:

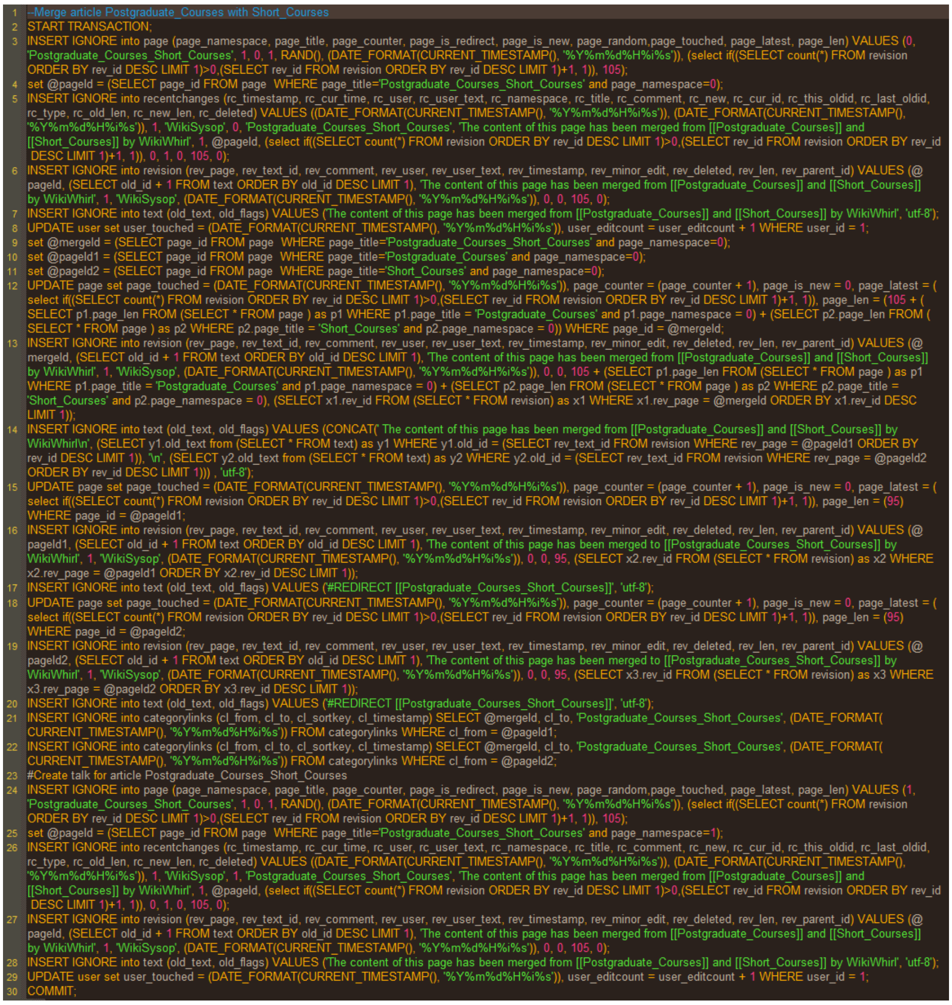
- Oscar Díaz, Gorka Puente, Javier Luis Cánovas Izquierdo, Jesús García Molina, "Harvesting Models from Web 2.0 Databases". In *Software and Systems Modeling* journal, (*SoSyM*), 2011. JCR, Impact factor 1.533.

Appendix B

MySQL Script for the Merge Operation

"All parts should go together without forcing. You must
remember that the parts you are reassembling were
disassembled by you. Therefore, if you can't get
them together again, there must be a reason.
By all means, do not use a hammer."
– IBM maintenance (1925).

Refactoring operations are transactions over the wiki database, specifically over the MediaWiki database. WikiWhirl automatically generates MySQL statements to modify the database with the refactoring changes. This appendix shows the MySQL script generated for the merge operation, which is based on the operational semantics.



```
--Merge article Postgraduate_Courses with Short_Courses
START TRANSACTION;
INSERT IGNORE into page (page_namespace, page_title, page_counter, page_is_redirect, page_is_new, page_random,page_touched, page_latest, page_len) VALUES (0,
'Postgraduate_Courses_Short_Courses', 1, 0, 1, RAND(), (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), (select if((SELECT count(*) FROM revision
ORDER BY rev_id DESC LIMIT 1)>0,(SELECT rev_id FROM revision ORDER BY rev_id DESC LIMIT 1)+1, 1)), 105);
set @pageId = (SELECT page_id FROM page  WHERE page_title='Postgraduate_Courses_Short_Courses' and page_namespace=0);
INSERT IGNORE into recentchanges (rc_timestamp, rc_cur_time, rc_user, rc_user_text, rc_namespace, rc_title, rc_comment, rc_new, rc_cur_id, rc_this_oldid, rc_last_oldid,
rc_type, rc_old_len, rc_new_len, rc_deleted) VALUES ((DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), (DATE_FORMAT(CURRENT_TIMESTAMP(),
'%Y%m%d%H%i%s')), 1, 'WikiSysop', 0, 'Postgraduate_Courses_Short_Courses', 'The content of this page has been merged from [[Postgraduate_Courses]] and
[[Short_Courses]] by WikiWhirl', 1, @pageId, (select if((SELECT count(*) FROM revision ORDER BY rev_id DESC LIMIT 1)>0,(SELECT rev_id FROM revision ORDER BY rev_id
 DESC LIMIT 1)+1, 1)), 0, 1, 0, 105, 0);
INSERT IGNORE into revision (rev_page, rev_text_id, rev_comment, rev_user, rev_user_text, rev_timestamp, rev_minor_edit, rev_deleted, rev_len, rev_parent_id) VALUES (@
pageId, (SELECT old_id + 1 FROM text ORDER BY old_id DESC LIMIT 1), 'The content of this page has been merged from [[Postgraduate_Courses]] and [[Short_Courses]]
by WikiWhirl', 1, 'WikiSysop', (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), 0, 0, 105, 0);
INSERT IGNORE into text (old_text, old_flags) VALUES ('The content of this page has been merged from [[Postgraduate_Courses]] and [[Short_Courses]] by WikiWhirl', 'utf-8');
UPDATE user set user_touched = (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), user_editcount = user_editcount + 1 WHERE user_id = 1;
set @mergeId = (SELECT page_id FROM page  WHERE page_title='Postgraduate_Courses_Short_Courses' and page_namespace=0);
set @pageId1 = (SELECT page_id FROM page  WHERE page_title='Postgraduate_Courses' and page_namespace=0);
set @pageId2 = (SELECT page_id FROM page  WHERE page_title='Short_Courses' and page_namespace=0);
UPDATE page set page_touched = (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), page_counter = (page_counter + 1), page_is_new = 0, page_latest = (
select if((SELECT count(*) FROM revision ORDER BY rev_id DESC LIMIT 1)>0,(SELECT rev_id FROM revision ORDER BY rev_id DESC LIMIT 1)+1, 1)), page_len = (105 + (
SELECT p1.page_len FROM (SELECT * FROM page ) as p1 WHERE p1.page_title = 'Postgraduate_Courses' and p1.page_namespace = 0) + (SELECT p2.page_len FROM (
SELECT * FROM page ) as p2 WHERE p2.page_title = 'Short_Courses' and p2.page_namespace = 0)) WHERE page_id = @mergeId;
INSERT IGNORE into revision (rev_page, rev_text_id, rev_comment, rev_user, rev_user_text, rev_timestamp, rev_minor_edit, rev_deleted, rev_len, rev_parent_id) VALUES (@
mergeId, (SELECT old_id + 1 FROM text ORDER BY old_id DESC LIMIT 1), 'The content of this page has been merged from [[Postgraduate_Courses]] and [[Short_Courses]]
by WikiWhirl', 1, 'WikiSysop', (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), 0, 0, 105 + (SELECT p1.page_len FROM (SELECT * FROM page ) as p1
WHERE p1.page_title = 'Postgraduate_Courses' and p1.page_namespace = 0) + (SELECT p2.page_len FROM (SELECT * FROM page ) as p2 WHERE p2.page_title =
'Short_Courses' and p2.page_namespace = 0), (SELECT x1.rev_id FROM (SELECT * FROM revision) as x1 WHERE x1.rev_page = @mergeId ORDER BY x1.rev_id DESC
LIMIT 1));
INSERT IGNORE into text (old_text, old_flags) VALUES (CONCAT(' The content of this page has been merged from [[Postgraduate_Courses]] and [[Short_Courses]] by
WikiWhirl\n', (SELECT y1.old_text from (SELECT * FROM text) as y1 WHERE y1.old_id = (SELECT rev_text_id FROM revision WHERE rev_page = @pageId1 ORDER BY
rev_id DESC LIMIT 1)), '\n', (SELECT y2.old_text from (SELECT * FROM text) as y2 WHERE y2.old_id = (SELECT rev_text_id FROM revision WHERE rev_page = @pageId2
ORDER BY rev_id DESC LIMIT 1))) , 'utf-8');
UPDATE page set page_touched = (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), page_counter = (page_counter + 1), page_is_new = 0, page_latest = (
select if((SELECT count(*) FROM revision ORDER BY rev_id DESC LIMIT 1)>0,(SELECT rev_id FROM revision ORDER BY rev_id DESC LIMIT 1)+1, 1)), page_len = (95)
WHERE page_id = @pageId1;
INSERT IGNORE into revision (rev_page, rev_text_id, rev_comment, rev_user, rev_user_text, rev_timestamp, rev_minor_edit, rev_deleted, rev_len, rev_parent_id) VALUES (@
pageId1, (SELECT old_id + 1 FROM text ORDER BY old_id DESC LIMIT 1), 'The content of this page has been merged to [[Postgraduate_Courses_Short_Courses]] by
WikiWhirl', 1, 'WikiSysop', (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), 0, 0, 95, (SELECT x2.rev_id FROM (SELECT * FROM revision) as x2 WHERE
x2.rev_page = @pageId1 ORDER BY x2.rev_id DESC LIMIT 1));
INSERT IGNORE into text (old_text, old_flags) VALUES ('#REDIRECT [[Postgraduate_Courses_Short_Courses]]', 'utf-8');
UPDATE page set page_touched = (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), page_counter = (page_counter + 1), page_is_new = 0, page_latest = (
select if((SELECT count(*) FROM revision ORDER BY rev_id DESC LIMIT 1)>0,(SELECT rev_id FROM revision ORDER BY rev_id DESC LIMIT 1)+1, 1)), page_len = (95)
WHERE page_id = @pageId2;
INSERT IGNORE into revision (rev_page, rev_text_id, rev_comment, rev_user, rev_user_text, rev_timestamp, rev_minor_edit, rev_deleted, rev_len, rev_parent_id) VALUES (@
pageId2, (SELECT old_id + 1 FROM text ORDER BY old_id DESC LIMIT 1), 'The content of this page has been merged to [[Postgraduate_Courses_Short_Courses]] by
WikiWhirl', 1, 'WikiSysop', (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), 0, 0, 95, (SELECT x3.rev_id FROM (SELECT * FROM revision) as x3 WHERE
x3.rev_page = @pageId2 ORDER BY x3.rev_id DESC LIMIT 1));
INSERT IGNORE into text (old_text, old_flags) VALUES ('#REDIRECT [[Postgraduate_Courses_Short_Courses]]', 'utf-8');
INSERT IGNORE into categorylinks (cl_from, cl_to, cl_sortkey, cl_timestamp) SELECT @mergeId, cl_to, 'Postgraduate_Courses_Short_Courses', (DATE_FORMAT(
CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')) FROM categorylinks WHERE cl_from = @pageId1;
INSERT IGNORE into categorylinks (cl_from, cl_to, cl_sortkey, cl_timestamp) SELECT @mergeId, cl_to, 'Postgraduate_Courses_Short_Courses', (DATE_FORMAT(
CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')) FROM categorylinks WHERE cl_from = @pageId2;
#Create talk for article Postgraduate_Courses_Short_Courses
INSERT IGNORE into page (page_namespace, page_title, page_counter, page_is_redirect, page_is_new, page_random,page_touched, page_latest, page_len) VALUES (1,
'Postgraduate_Courses_Short_Courses', 1, 0, 1, RAND(), (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), (select if((SELECT count(*) FROM revision
ORDER BY rev_id DESC LIMIT 1)>0,(SELECT rev_id FROM revision ORDER BY rev_id DESC LIMIT 1)+1, 1)), 105);
set @pageId = (SELECT page_id FROM page  WHERE page_title='Postgraduate_Courses_Short_Courses' and page_namespace=1);
INSERT IGNORE into recentchanges (rc_timestamp, rc_cur_time, rc_user, rc_user_text, rc_namespace, rc_title, rc_comment, rc_new, rc_cur_id, rc_this_oldid, rc_last_oldid,
rc_type, rc_old_len, rc_new_len, rc_deleted) VALUES ((DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), (DATE_FORMAT(CURRENT_TIMESTAMP(),
'%Y%m%d%H%i%s')), 1, 'WikiSysop', 1, 'Postgraduate_Courses_Short_Courses', 'The content of this page has been merged from [[Postgraduate_Courses]] and
[[Short_Courses]] by WikiWhirl', 1, @pageId, (select if((SELECT count(*) FROM revision ORDER BY rev_id DESC LIMIT 1)>0,(SELECT rev_id FROM revision ORDER BY rev_id
 DESC LIMIT 1)+1, 1)), 0, 1, 0, 105, 0);
INSERT IGNORE into revision (rev_page, rev_text_id, rev_comment, rev_user, rev_user_text, rev_timestamp, rev_minor_edit, rev_deleted, rev_len, rev_parent_id) VALUES (@
pageId, (SELECT old_id + 1 FROM text ORDER BY old_id DESC LIMIT 1), 'The content of this page has been merged from [[Postgraduate_Courses]] and [[Short_Courses]]
by WikiWhirl', 1, 'WikiSysop', (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), 0, 0, 105, 0);
INSERT IGNORE into text (old_text, old_flags) VALUES ('The content of this page has been merged from [[Postgraduate_Courses]] and [[Short_Courses]] by WikiWhirl', 'utf-8');
UPDATE user set user_touched = (DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%s')), user_editcount = user_editcount + 1 WHERE user_id = 1;
COMMIT;
```

Figure B.1: *MySQL* script automatically generated by WikiWhirl for the merge operation.

Appendix C

MediaWiki Background Questionnaire

"Judge a man by his questions rather than by his answers."
– *François Marie Arouet 'Voltaire'.*

We are performing a study to evaluate use of WikiWhirl, a tool to support wiki initialization and restructuration of Wikis, which has been developed by the Onekin group of the University of the Basque Country (UPV/EHU). In this first questionnaire we want to evaluate your previous MediaWiki knowledge in order to create the experimental groups[1]. All data will be stored anonymously. Thank you for your collaboration and the honesty of your answers.

1. Write the first three letters of your mother's name followed by the last four figures of your national ID number. Keeping your data anonymous, this code will be used to create homogeneous experimental groups. On the day of the experiment this code will be used again (e.g., Mary, ID 12345678F –> Code MAR5678).

[1] As all wikis participants collaborate in are built using MediaWiki and in order to avoid confusion, in the rest of the questionnaire questions refer to wikis in general.

Basic Concepts

Decide if the statements are true or false

1. Only the original author can modify a wiki

   (a) True

   (b) False

2. HTML knowledge is required to create or edit articles of a wiki

   (a) True

   (b) False

Wiki Knowledge

1. I know how to edit an article in a wiki
2. I know how to create an article in a wiki
3. I know how to delete an article in a wiki
4. I know how to create a category in a wiki
5. To rename a category, it is enough to click the corresponding button.
6. To change an article's category (categorize), it is enough to edit it and to change categoryName in [[Category:categoryName]]

Wiki Use

We want to know the frequency with which you perform the following tasks.

1. I add content to existing articles

   (a) Never

   (b) Between 1 and 5 times a WEEK

   (c) Between 6 and 10 times a WEEK

   (d) More than 10 times a WEEK

2. I create new articles

   (a) Never

   (b) Between 1 and 5 times a YEAR

   (c) Between 6 and 10 times a YEAR

   (d) More than 10 times a YEAR

3. I create new categories

   (a) Never

   (b) Between 1 and 5 times a YEAR

   (c) Between 6 and 10 times a YEAR

   (d) More than 10 times a YEAR

4. I reorganize existing article an category groups (e.g., assign articles to new categories, change article names, etc.)

   (a) Never

   (b) Between 1 and 5 times a YEAR

   (c) Between 6 and 10 times a YEAR

   (d) More than 10 times a YEAR

Others

1. Describe your relationship with Wikipedia

   (a) I consult it

(b) I consult it and I have an account

(c) I am an active user

(d) None of the above

Appendix D

Global Understandability Questionnaire

> "Research is to see what everybody else has seen,
> and to think what nobody else has thought."
> – *Albert Szent-Gyorgyi.*

In this questionnaire we want to evaluate how easy it is to understand the structure and content of the wiki. All data will be stored anonymously. Thank you for your collaboration.

1. Write the first three letters of your mother's name followed by the last four figures of your national ID number. Keeping your data anonymous, this code will be used to create homogeneous experimental groups. On the day of the experiment this code will be used again (e.g., Mary, ID 12345678F –> Code MAR5678).

WikiVet Questions

1. Could you name two parent categories of Blood_Changes?

(a) A category can only have one parent category, in this case Clinical_Pathology.

(b) WikiBlood and Clinical_Pathology.

(c) General Pathology and WikiEpi.

2. Mycoplasmas are characterized as Infectious_Agents?

   (a) Yes.

   (b) No, they are characterized as a pathology (WikiPath).

   (c) No, they are characterized as a virus (Viruses).

3. Which statement indicates MORE PRECISELY the relation between articles Anaplasmosis and Coagulation_Tests?

   (a) Both are characterized as general pathologies.

   (b) Both are characterized as pathologies (WikiPath).

   (c) Both are characterized as degenerative pathologies (Degenerations).

4. Fungi are characterized as...

   (a) Mycoses

   (b) Infectious_Agents

   (c) Bacteria

5. How many articles are characterized as epidemiology (WikiEpi)?

   (a) 1

   (b) 3

   (c) 4

6. In total, how many articles and categories are characterized as pathologies (WikiPath)?

(a) 2
(b) 9
(c) 10

7. How many categories are defined in the WikiVet wiki?

    (a) 5
    (b) 24
    (c) 25

8. With the data available in the wiki, would it be correct to characterize the Anaplasmosis as a Clinical_Pathology?

    (a) No, because it has nothing to do with pathologies.
    (b) No, because even if it characterized as a pathology, it is characterized as a General_Pathology.
    (c) Yes, because it is related to Clinical_Pathology.

9. With the data available in the wiki, what do you think the Short_Courses article is about?

    (a) About short courses in epidemiology (WikiEpi).
    (b) About short courses in education.
    (c) About short courses in veterinary (Wikivet).

10. With the data available in the wiki, what have cells, la anaemia and haematology changes in common?

    (a) They are clinical pathologies.
    (b) They are all related to blood.
    (c) They characterize immunology and pressure.

11. With the data available in the wiki, do you believe that an haematology change can indicate a clinical pathology?

    (a) No, an haematology change does not indicate a clinical pathology.

    (b) Yes, both concepts are related in the wiki.

    (c) No, both conceps are not related in the wiki.

12. With the data available in the wiki, do you believe that anaplasmosis is caused by a parasite?

    (a) No, it is caused by a bacteria.

    (b) Yes, both concepts are related in the wiki.

    (c) No, both conceps are not related in the wiki.

13. With the data available in the wiki, do you believe that parasites can be...?

    (a) Fungy and bacteria.

    (b) Mycoses y mycoplasmas.

    (c) None of the above.

14. With the data available in the wiki, do you believe that paracetamol is prescribed in...?

    (a) Degenerative pathologies.

    (b) Clinical pathologies.

    (c) No answer can be deduced from the wiki.

Appendix E

Final Questionnaire

“Knowing is not enough, you must apply;
willing is not enough, you must do.”
– Bruce Lee.

We are performing a study to evaluate use of WikiWhirl, a tool to support wiki initialization and restructuration of Wikis, which has been developed by the Onekin group of the University of the Basque Country (UPV/EHU). This last questionnaire has two short sections. All data will be stored anonymously. Thank you for your collaboration and the honesty of your answers.

General

1. Write the first three letters of your mother’s name followed by the last four figures of your national ID number. Keeping your data anonymous, this code will be used to create homogeneous experimental groups. On the day of the experiment this code will be used again (e.g., Mary, ID 12345678F –> Code MAR5678).

2. Gender

   (a) Female

   (b) Male

3. Age

4. Rate your programming background (1-none, 5-expert)

5. Rate your Web engineering background (1-none, 5-expert)

6. Rate your Web 2.0 background (1-none, 5-expert)

7. Rate your collaborative system background (1-none, 5-expert)

8. Rate your mind mapping background (1-none, 5-expert)

Execution

In this section the performed tasks and the invested time are collected. For each task, we ask you to write whether you have finished its parts and how long it took. Remember that it is normal not to have finished some of the tasks in the allocated time.

FIRST TASK: Understand Content

1. I have finished the questionnaire on how undestandable the content and structure of the wiki is.

2. If you have finished the task, write down the time it took (minutes).

SECOND TASK: Refactor the Wiki Structure

1. I have finished the CREATE operations in the handout.

2. If you have finished the CREATE operations, write down the time it took (minutes).

3. I have finished the CATEGORIZE operations in the handout.

4. If you have finished the CATEGORIZE operations, write down the time it took (minutes).I have finished the CREATE operations in the handout.
5. If you have finished the UNCATEGORIZE operations, write down the time it took (minutes).I have finished the CREATE operations in the handout.
6. If you have finished the UNCATEGORIZE operations, write down the time it took (minutes).
7. I have finished the RENAME operations in the handout.
8. If you have finished the RENAME operations, write down the time it took (minutes).
9. I have finished the DROP operations in the handout.
10. If you have finished the DROP operations, write down the time it took (minutes).

THIRD TASK: Refactor the Wiki Content

1. I have finished the SPLIT operations in the handout.
2. If you have finished the SPLIT operations, write down the time it took (minutes).
3. I have finished the MERGE operations in the handout.
4. If you have finished the MERGE operations, write down the time it took (minutes).
5. I have finished the MOVE operations in the handout.
6. If you have finished the MOVE operations, write down the time it took (minutes).

Bibliography

[AC10] Judd Antin and Coye Cheshire. Readers are not Free-Riders: Reading as a Form of Participation on Wikipedia. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, CSCW '10, pages 127–130. ACM, 2010.

[AdAMV$^+$11] B. Thomas Adler, Luca de Alfaro, Santiago Moisés Mola-Velasco, Paolo Rosso, and Andrew G. West. Wikipedia Vandalism Detection: Combining Natural Language, Metadata, and Reputation Features. In *Proceedings of the 12th international conference on Computational linguistics and intelligent text processing - Volume Part II*, CICLing'11, pages 277–288. Springer-Verlag, 2011.

[ADM] Architecture-Driven Modernization (ADM). Online; `http://adm.omg.org` [accessed July 2012].

[ADM09] Ademar Aguiar, Uri Dekel, and Paulo Merson. Wikis for Software Engineering. In *Proceedings of the 31st International Conference on Software Engineering, ICSE Companion*, Wikis4SE'09, pages 480–481. IEEE Computer Society, 2009.

[Ame09] David Ameller. Considering Non-Functional Requirements in Model-Driven Engineering. Master's thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 2009.

[ASR$^+$10] O. Arazy, E. Stroulia, S. Ruecker, C. Arias, C. Fiorentino, V. Ganev, and T. Yau. Recognizing Contributions in Wikis: Authorship Categories, Algorithms, and Visualizations. *Journal of the American Society for Information Science and Technology (JASIST)*, pages 1166–1179, 2010.

[ath] ATHENA (MDI) Framework. Online; `www.modelbased.net/mdi/index.html`. [accessed July 2012].

[AVG10] AVG. AVG LinkScanner - How it Works, 2010. Online; `http://linkscanner.avg.com/ww.sals-how-it-works.html` [accessed July 2012].

[BAGB11] Ankica Barisic, Vasco Amaral, Miguel Goulão, and Bruno Barroca. Quality in Use of Domain-Specific Languages: a Case Study. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU'11, pages 65–72. ACM, 2011.

[Bas92] Victor R. Basili. Software Modeling and Measurement: the Goal/Question/Metric Paradigm. Technical report, College Park, MD, USA, 1992.

[Bat06] Don S. Batory. Multilevel Models in Model-Driven Engineering, Product Lines, and Metaprogramming. *IBM Systems Journal*, 45(3):527–540, 2006.

[BBJ$^+$10] Gabriel Barbier, Hugo Bruneliere, Frédéric Jouault, Yves Lennon, and Frédéric Madiot. Modisco, A Model-Driven Platform to Support Real Legacy Modernization

Uses Cases. In *Information Systems Transformation: Architecture-Driven Modernization Case Studies*. Elsevier Science, 2010.

[Béz04] Jean Bézivin. In Search of a Basic Principle for Model-Driven Engineering. *UPGRADE, The European Journal for the Informatics Professional, Special Issue on UML and Model Engineering*, 5(2):21–24, 2004.

[BG06] Michel Buffa and Fabien Gandon. SweetWiki: Semantic Web Enabled Technologies in Wiki. In *Proceedings of the 2th International Symposium on Wikis and Open Collaboration*, WikiSym '06, pages 69–78. ACM, 2006.

[BG10] Tony Buzan and Chris Griffiths. *Mind Maps for Business*. BBC active, 2010.

[BK05] Jean Bezivin and Ivan Kurtev. Model-based Technology Integration with the Technical Space Concept. In *Proceedings of the Metainformatics Symposium*. Springer-Verlag, 2005.

[BLW05] Paul Baker, Shiou Loh, and Frank Weil. Model-Driven Engineering in a Large Industrial Context - Motorola Case Study. In *8th International Conference on Model Driven Engineering Languages and Systems*, volume 3713 of *MoDELS'05*, pages 476–491. Springer, 2005.

[Bou99] Niels Olof Bouvin. Unifying Strategies for Web Augmentation. In *Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots: returning to our diverse roots*, HYPERTEXT '99, pages 91–100. ACM, 1999.

[BQBB10] Gillian Brown, Megan Quentin-Baxter, and Zoe Belshaw. WikiVet: Building a Community of Practice to Support a Self-sustaining Wiki for Veterinary Education. *International Journal of Web Based Communities (IJWBC)*, 6(2):183–196, 2010.

[BSV10] Jean Bézivin, Richard Mark Soley, and Antonio Vallecillo. Editorial to the Proceedings of the First International Workshop on Model-Driven Interoperability. In *Proceedings of the First International Workshop on Model-Driven Interoperability*, MDI '10, pages 1–2, New York, NY, USA, 2010. ACM.

[Béz04] Jean Bézivin. In Search of a Basic Principle for Model Driven Engineering. *Novatica Upgrade*, V(2):21–24, 2004.

[Car03] Kathleen M. Carley. *Dynamic Network Analysis*, pages 133–145. Dynamic Social Network Modeling and Analysis:. Committee on Human Factors and National Research Council, 2003.

[Car07] Dan Carlin. Corporate Wikis Go Viral, 2007. Online; `www.businessweek.com/technology/content/mar2007/tc20070312_476504.htm` [accessed July 2012].

[CH06] Krzysztof Czarnecki and Simon Helsen. Feature-Based Survey of Model Transformation Approaches. *IBM Systems Journal*, 45(3):621–646, 2006.

[CL12] Michael Cariaso and Greg Lennon. SNPedia: a Wiki Supporting Personal Genome Annotation, Interpretation and Analysis. *Nucleic Acids Research*, 40(Database-Issue):1308–1312, 2012.

[CMT06] Jesús Sánchez Cuadrado, Jesús García Molina, and Marcos Menárguez Tortosa. RubyTL: A Practical, Extensible Transformation Language. In *Proceedings of the Second European conference on Model Driven Architecture: foundations and Applications*, ECMDA-FA'06. Springer-Verlag, 2006.

[Coh88] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 2 edition, 1988.

[Col09] Melissa Cole. Using Wiki Technology to Support Student Engagement: Lessons from the Trenches. *Computers & Education*, 52(1):141–146, 2009.

[Cun02] Ward Cunningham. What Is Wiki, 2002. Online; `www.wiki.org/wiki.cgi?WhatIsWiki` [accessed July 2012].

[Cun06] Ward Cunningham. Design principles of wiki: how can so little do so much? In *Proceedings of the 2006 international symposium on Wikis*, WikiSym '06, pages 13–14. ACM, 2006.

[Cun11] Ward Cunningham. Smallest Federated Wiki, 2011. Online; `http://c2.com/cgi/wiki?SmallestFederatedWiki` [accessed July 2012].

[Cza05] Krzysztof Czarnecki. Overview of Generative Software Development. *Unconventional Programming Paradigms*, 3566(Unconventional Programming Paradigms):326–341, 2005.

[DAP12] Oscar Díaz, Cristobal Arellano, and Gorka Puente. Wikipedia Customization through Web Augmentation Techniques. In *Proceedings of the 8th International Symposium on Wikis and Open Collaboration*, WikiSym '12. ACM, 2012.

[DP10] Oscar Díaz and Gorka Puente. Model-Aware Wiki Analysis Tools: the Case of HistoryFlow. In *Proceedings of the 6th International Symposium on Wikis and Open Collaboration*, WikiSym '10. ACM, 2010.

[DP11a] Oscar Díaz and Gorka Puente. A DSL for Corporate Wiki Initialization. In *Proceedings of the 23th International Conference on Advanced Information Systems Engineering*, CAiSE'11, pages 237–251. Springer, 2011.

[DP11b] Oscar Díaz and Gorka Puente. Wiki Scaffolding: Helping Organizations to Set Up Wikis. In *Proceedings of the 7th International Symposium on Wikis and Open Collaboration*, WikiSym '11, pages 154–162. ACM, 2011.

[DP12] Oscar Díaz and Gorka Puente. Wiki Scaffolding: Aligning Wikis with the Corporate Strategy. *Information Systems journal*, 37(8):737–752, 2012.

[DPA11] Oscar Díaz, Gorka Puente, and Cristóbal Arellano. Wiki Refactoring: an Assisted Approach based on Ballots. In *Proceedings of the 7th International Symposium on Wikis and Open Collaboration*, WikiSym '11, pages 195–196. ACM, 2011.

[DPCIGM11] Oscar Díaz, Gorka Puente, Javier Luis Cánovas Izquierdo, and Jesús García Molina. Harvesting Models from Web 2.0 Databases. *Software and Systems Modeling, SoSyM*, pages 1–20, 2011. 10.1007/s10270-011-0194-z.

[DS08] Catalina Danis and David Singer. A Wiki instance in the Enterprise: Opportunities, Concerns and Reality. In *Proceedings of the 2008 ACM conference on Computer* *supported cooperative work*, CSCW ’08, pages 495–504. ACM, 2008.

[Due08] Gunter Dueck. Bluepedia. *Informatik Spektrum*, 31(3):262–269, 2008.

[EGHW08] Anja Ebersbach, Markus Glaser, Richard Heigl, and Alexander Warta. *Wiki: Web Collaboration*. Springer-Verlag New York Inc, 2008.

[EMF] Eclipse Modeling Framework. Online; `www.eclipse.org/modeling/emf` [accessed July 2012].

[Fil06] Robert E. Filman. Taking Back the Web. *IEEE Internet Computing*, 10:3–5, 2006.

[Fin09] Finding ROI. Measuring Intranet Investments. Technical report, Prescient Digital Media, 2009.

[Fow99] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[Fre] FreeMind, Mind Mapping Software. Online; `http://freemind.sourceforge.net` [accessed July 2012].

[Fre10] Chuck Frey. Mind Mapping Software User Survey, 2010. Online at `http://mindmappingsoftwareblog.com/2010-survey-form` [accessed December 2012].

[GKSK11] Chitrabharathi Ganapathy, Jeon-Hyung Kang, Erin Shaw, and Jihie Kim. Classification Techniques for Assessing Student Collaboration in Shared Wiki Spaces. In *Proceedings of the 15th international conference on Artificial intelligence in education*, AIED’11, pages 456–458. Springer-Verlag, 2011.

[GP10] Jonathan Grudin and Erika Shehan Poole. Wikis at Work: Success Factors and Challenges for Sustainability of Enterprise Wikis. In *Proceedings of the 6th International Symposium on Wikis and Open Collaboration*, WikiSym '10, pages 5:1–5:8. ACM, 2010.

[Gro09] Future Melbourne Reference Group. Future Melbourne Wiki: Post Implementation Review. Technical report, City of Melbourne, 2009.

[HDW10] Lester J. Holtzblatt, Laurie E. Damianos, and Daniel Weiss. Factors Impeding Wiki Use in the Enterprise: a Case Study. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems, Extended Abstracts Volume*, CHI'10, pages 4661–4676. ACM, 2010.

[HF12] R. Stuart Geiger Heather Ford. "Writing Up Rather than Writing Down": Becoming Wikipedia Literate. In *Proceedings of the 8th International Symposium on Wikis and Open Collaboration*, WikiSym '12. ACM, 2012.

[HHGC10] Christian Hirsch, John G. Hosking, John C. Grundy, and Tim Chaffe. ThinkFree: Using a Visual Wiki for IT Knowledge Management in a Tertiary Institution. In *Int. Sym. Wikis*, WikiSym '10, pages 7:1–7:10. ACM, 2010.

[HMPR04] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[HSB07] Martin Hepp, Katharina Siorpaes, and Daniel Bachlechner. Harvesting Wiki Consensus: Using Wikipedia Entries as Vocabulary for Knowledge Management. *IEEE Internet Computing*, 11(5):54–65, 2007.

[HWRK11] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical Assessment of MDE in Industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 471–480. ACM, 2011.

[HWS07] Wu-Yuin Hwang, Chin-Yu Wang, and Mike Sharples. A Study of Multimedia Annotation of Web-Based Materials. *Computers & Education*, 48(4):680–699, 2007.

[IM10] Javier Luis Cánovas Izquierdo and Jesús García Molina. An Architecture-Driven Modernization Tool for Calculating Metrics. *IEEE Software*, 27(4):37–43, 2010.

[ISO] ISO/IEC 9126-1:2001 Software engineering – Product quality – Part 1: Quality model.

[JABK08] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. ATL: A Model Transformation Tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008.

[JML11] Sara Javanmardi, David W. McDonald, and Cristina Videira Lopes. Vandalism Detection in Wikipedia: a High-Performing, Feature-Rich Model and its Reduction through Lasso. In *Proceedings of the 7th International Symposium on Wikis and Open Collaboration*, WikiSym '11, pages 82–90. ACM, 2011.

[Joh92] Peter Johnson. *Human Computer Interaction: Psychology, Task Analysis, and Software Engineering*. McGraw-Hill, 1992.

[JP05] Andreas Jedlitschka and Dietmar Pfahl. Reporting Guidelines for Controlled Experiments in Software

Engineering. In *Proceedings of the International Symposium on Empirical Software Engineering*, ISESE'05, pages 95–104, 2005.

[JPBB07] Guy Doumeingts Jean-Pierre Bourey, Reyes Grangel and Arne J. Berre. Report on Model Driven Interoperability. Technical report, INTEROP, Interoperability Research for Networked Enterprises Applications and Software, 2007.

[KBA02] Ivan Kurtev, Jean Bézivin, and Mehmet Aksit. Technological Spaces: An Initial Appraisal. In *Proceedings of the International Symposium on Distributed Objects and Applications*, CoopIS, DOA'02 Federated Conferences, Industrial Track, 2002.

[KCH+90] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, 1990.

[KMC11] Joachim Kimmerle, Johannes Moskaliuk, and Ulrike Cress. Using Wikis for Learning and Knowledge Building: Results of an Experimental Study. *Educational Technology & Society*, 14(4):138–148, 2011.

[Lam04] Briam Lamb. Taking a Walk on the Wiki Side, 2004. Online; `http://campustechnology.com/articles/2004/04/taking-a-walk-on-the-wiki-side.aspx` [accessed July 2012].

[LB10] Hyunkyung Lee and Curtis Bonk. The Use of Wikis for Collaboration in Corporations: Perceptions and Implications for Future Research. In *World Conference on*

E-Learning in Corporate, Government, Healthcare, and Higher Education, 2010.

[LDP+11] Ioanna Lykourentzou, Younes Djaghloul, Katerina Papadaki, Foteini Dagka, and Thibaud Latour. Planning for a Successful Corporate Wiki. In *Proceedings of the Digital Enterprise and Information Systems - International Conference*, DEIS 2011, pages 425–439. Springer, 2011.

[LDP+12] Ioanna Lykourentzou, Foteini Dagka, Katerina Papadaki, Giorgos Lepouras, and Costas Vassilakis. Wikis in Enterprise Settings: a Survey. *Enterprise Information Systems*, 6(1):1–53, 2012.

[LE07] Ralph Lengler and Martin J. Eppler. Towards a Periodic Table of Visualization Methods for Management. In *International Conference on Graphics and Visualization in Engineering*, GVE'07, pages 1–6, 2007.

[LOO+12] Cliff Lampe, Jonathan Obar, Elif Ozkaya, Paul Zube, and Alcides Velasquez. Classroom Wikipedia Participation Effects on Future Intentions to Contribute. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 403–406. ACM, 2012.

[Lou06] Panagiotis Louridas. Using Wikis in Software Development. *IEEE Software*, 23(2):88–91, 2006.

[Mad08] Stewart Mader. *Wikipatterns : A Practical Guide to Improving Productivity and Collaboration in your Organization.* John Wiley & Sons Inc, 2008.

[McF05] Nigel McFarlane. Fixing Web Sites with Greasemonkey. *Linux Journal*, 138:1, 2005.

[McH12] Roger McHaney. The Web 2.0 Mandate for a Transition from Webmaster to Wiki Master. In *Open-Source Technologies for Maximizing the Creation, Deployment, and Use of Digital Resources and Information*. Shalin Hai-Jew, 2012.

[MD08a] Parastoo Mohagheghi and Vegard Dehlen. Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In *Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications*, ECMDA-FA '08, pages 432–443. Springer-Verlag, 2008.

[MD08b] Parastoo Mohagheghi and Vegard Dehlen. Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In *4th European Conference on Model Driven Architecture - Foundations and Applications*, volume 5095 of *ECMDA-FA'08*, pages 432–443. Springer, 2008.

[mem03] OMG members. MDA Guide Version 1.0.1. Technical report, OMG, 2003. Online; `http://www.omg.org/cgi-bin/doc?omg/03-06-01` [accessed 3-Feb-12].

[Men] Bank Identification Number (BIN) Sponsorship. Online; `http://goo.gl/VNL4X`. [accessed July 2012].

[MF12] Mostafa Mesgari and Samer Faraj. Technology Affordances: The Case of Wikipedia. In *Proceedings of the 18th Americas Conference on Information Systems*, AMCIS'12, 2012.

[MFBC10] Pierre-Alain Muller, Frédéric Fondement, Benoît Baudry, and Benoît Combemale. Modeling Modeling Modeling. *Software and Systems Modeling*, pages 1–13, 2010. 10.1007/s10270-010-0172-x.

[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.

[MWY06] Ann Majchrzak, Christian Wagner, and Dave Yates. Corporate Wiki Users: Results of a Survey. In *Proceedings of the 2006 international symposium on Wikis*, WikiSym '06, pages 99–104. ACM, 2006.

[Nor02] Donald A. Norman. *The Design of Everyday Things*. Basic Books, 2002.

[Nov09] Beth Simone Noveck. *Wiki Government: How Technology Can Make Government Better, Democracy Stronger, and Citizens more Powerful.* Brookings institution press, 2009.

[NPO] Wikipedia's Neutral Point of View Guidelines. Online; `http://en.wikipedia.org/wiki/Wikipedia:Neutral_point_of_view` [accessed July 2012].

[NT95] I. Nonaka and H. Takeuchi. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation.* Oxford University Press, 1995.

[Nyf09] Felix Nyffenegger. WikiMindMap, 2009. Online at `www.wikimindmap.org` [accessed December 2012].

[Oli07] Antoni Olivé. *Conceptual Modeling of Information Systems*. Springer, 2007.

[OMG] OMG. OMG Model Driven Architecture (MDA). Online; `www.omg.org/mda/.` [accessed July 2012].

[OMG02] OMG. Meta Object Facility (MOF) Specification - Version 1.4. Adopted Specification, April 2002.

Online; `http://www.omg.org/spec/MOF/1.4/PDF/` [accessed June 2012]].

[ORe05] Tim OReilly. What Is Web 2.0, 2005. Online; `http://oreilly.com/pub/a/web2/archive/what-is-web-20.html?page=1` [accessed July 2012].

[OVBD06] Eyal Oren, Max Völkel, John G. Breslin, and Stefan Decker. Semantic Wikis for Personal Knowledge Management. In *Proceedings of the 17th international conference on Database and Expert Systems Applications*, DEXA'06, pages 509–518. Springer-Verlag, 2006.

[Pat08] Susanne Patig. A Practical Guide to Testing the Understandability of Notations. In *Proceedings of the 5th Asia-Pacific Conference on Conceptual Modelling*, volume 79 of *APCCM'08*, pages 49–58. Australian Computer Society, 2008.

[PD12] Gorka Puente and Oscar Díaz. Wiki Refactoring as Mind Map Reshaping. In *Proceedings of the 24th International Conference on Advanced Information Systems Engineering*, CAiSE'12. Springer, 2012.

[PDA13] Gorka Puente, Oscar Díaz, and Maider Azanza. Refactoring Affordances in Corporate Wikis: A Case for the Use of Mind Maps. *Information Systems journal*, 2013. Under review.

[Phu09] Ammy Jiranida Phuwanartnurak. Did you Put it on the Wiki?: Information Sharing through Wikis in Interdisciplinary Design Collaboration. In *Proceedings of the 27th Annual International Conference on Design* *of Communication*, SIGDOC'09, pages 273–280. ACM, 2009.

[Ram06] Murali Raman. Wiki Technology as A "Free" Collaborative Tool within an Organizational Setting. *IS Management*, 23:59–66, 2006.

[RFD10] Martin Rosenfeld, Alejandro Fernández, and Alicia Díaz. Semantic Wiki Refactoring. A Strategy to Assist Semantic Wiki Evolution. In *5th Workshop on Semantic Wikis Linking Data and People*, SemWiki2010, 2010.

[RGvD06] Thijs Reus, Hans Geers, and Arie van Deursen. Harvesting Software Systems for MDA-Based Reengineering. In *Proceedings of the 2nd European Conference on Model-Driven Architecture - Foundations and Applications*, ECMDA-FA'06, pages 213–225. Springer-Verlag New York, Inc., 2006.

[RRO05] Murali Raman, Terry Ryan, and Lorne Olfman. Designing Knowledge Management Systems for Teaching and Learning with Wiki Technology. *Journal of Information Systems Education*, 16(3):311–321, 2005.

[Sar05] Edward P. Sarafino. *Research Methods: Using Processes & Procedures of Science to Understand Behavior*. Pearson, 2005.

[SB09] Klaus Stein and Steffen Blaschke. Corporate Wikis: A Comparative Analysis of Structures and Dynamics. In *Fifth Conference Professional Knowledge Management: Experiences and Visions*, Wissensmanagement'09, pages 77–86. GI, 2009.

[SBBK08] Sebastian Schaffert, François Bry, Joachim Baumeister, and Malte Kiesel. Semantic Wikis. *IEEE Software*, 2008.

[Sch06] Douglas C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, 2006.

[SDJ$^+$08] Yu Sun, Zekai Demirezen, Frédéric Jouault, Robert Tairas, and Jeff Gray. A Model Engineering Approach to Tool Interoperability. In *Proceedings of the 2008 Software Language Engineering*, SLE'08, pages 178–187. Springer-Verlag, 2008.

[Sei03] Ed Seidewitz. What Models Mean. *IEEE Software*, 20(5):26–32, 2003.

[Sel03] Bran Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20(5):19–25, 2003.

[Sel07] Bran Selic. From Model-Driven Development to Model-Driven Engineering. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, ECRTS'07, page 3. IEEE Computer Society, 2007.

[Sem] Semantic MediaWiki SMW. Online; `http://semantic-mediawiki.org/wiki/Semantic_MediaWikis` [accessed July 2012].

[SHH$^+$05] Dag I. K. Sjoberg, Jo E. Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, Nils-Kristian Liborg, and Anette C. Rekdal. A Survey of Controlled Experiments in Software Engineering. *IEEE Transactions on Software Engineering*, 31(9):733–753, September 2005.

[SK06] Yuyan Su and James Klein. Effects of Navigation Tools and Computer Confidence on Performance and Attitudes in a Hypermedia Learning Environment. *Journal of Educational Multimedia and Hypermedia*, 15(1):87–106, 2006.

[Sky05] Skype. Skype button in Internet Explorer or Firefox toolbar, 2005. Online; `www.skype.com/intl/en/support/user-guides/toolbar` [accessed July 2012].

[SMB08] Ragnhild Van Der Straeten, Tom Mens, and Stefan Van Baelen. Challenges in Model-Driven Software Engineering. In *Models in Software Engineering*, MoDELS'08 Workshops, pages 35–47. Springer-Verlag, 2008.

[Spe93] Paul E. Spector. Research Designs. *Experimental Design and Methods*, pages 1–72, 1993.

[ST09] Alexander Stocker and Klaus Tochtermann. Exploring the Value of Enterprise Wikis - A Multiple-Case Study. In *Proceedings of the 1st International Conference on Knowledge Management and Information Sharing*, KMIS'09, pages 5–12. Springer, 2009.

[ST11] Alexander Stocker and Klaus Tochtermann. Enterprise Wikis - Types of Use, Benefits and Obstacles: A Multiple-Case Study. *Communications in Computer and Information Science*, 128(4):297–309, 2011.

[TLEC11] Wei-Tek Tsai, Wu Li, Jay Elston, and Yinong Chen. Collaborative Learning Using Wiki Web Sites for Computer Science Undergraduate Education: A Case Study. *IEEE Trans. Education*, 54(1):114–124, 2011.

[TMC08] Sacip Toker, James L. Moseley, and Ann T. Chow. Is There a Wiki in Your Future?: Applications for Education, Instructional Design, and General Use. *Educational Technology Magazine*, page 6, 2008.

[TPP09] Franck Tétard, Erkki Patokorpi, and Kristian Packalén. Using Wikis to Support Constructivist Learning: A Case Study in University Education Settings. In *Proceedings of the 42nd Hawaii International International Conference on Systems Science*, HICSS '09, pages 1–10. IEEE Computer Society, 2009.

[TS10] Brendan Tansey and Eleni Stroulia. Annoki: a MediaWiki-based Collaboration Platform. In *Proceedings of the 1st Workshop on Web 2.0 for Software Engineering*, Web2SE '10, pages 31–36. ACM, 2010.

[TSMDM09] Diego Torres, Hala Skaf-Molli, Alicia Díaz, and Pascal Molli. Supporting Personal Semantic Annotations in P2P Semantic Wikis. In *Proceedings of the 20th International Conference on Database and Expert Systems Applications*, DEXA '09, pages 317–331. Springer-Verlag, 2009.

[UK07] Adam John Ullman and Judy Kay. WikiNavMap: a Visualisation to Supplement Team-based Wikis. In *CHI Extended Abstracts*, CHI EA'07, pages 2711–2716. ACM, 2007.

[UN10] William M. Ulrich and Philip H. Newcomb. *Information Systems Transformation: Architecture-Driven Modernization Case Studies*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[Val08] Antonio Vallecillo. A Journey through the Secret Life of Models. In Uwe Ašmann, Jean Bézivin, Richard Paige, Bernhard Rumpe, and Douglas C. Schmidt, editors, *Perspectives Workshop: Model Engineering of Complex Systems (MECS)*, number 08331 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[Val10] Antonio Vallecillo. On the Combination of Domain Specific Modeling Languages. In *Proceedings of the 6th European Conference on Modelling Foundations and Applications*, ECMFA'10, pages 305–320. Springer, 2010.

[Vat10] Ravi K. Vatrapu. Explaining Culture: an Outline of a Theory of Socio-Technical Interactions. In *Proceedings of the 3rd international conference on Intercultural collaboration*, ICIC '10, pages 111–120. ACM, 2010.

[vDKV00] Arie van Deursen, Paul Klint, and Joost Visser. Domain-Specific Languages: An Annotated Bibliography. *SIGPLAN Notices*, 35(6):26–36, 2000.

[Vis07] Eelco Visser. WebDSL: A Case Study in Domain-Specific Language Engineering. In *GTTSE*, GTTSE'07, pages 291–373. Springer-Verlag, 2007.

[VWD04] Fernanda B. Viégas, Martin Wattenberg, and Kushal Dave. Studying Cooperation and Conflict between Authors with History Flow Visualizations. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '04, pages 575–582. ACM, 2004.

[w3c] Use Case Wiki Template, W3C. Online; `www.w3.org/egov/wiki/Use_Case` [accessed July 2012].

[War11] Toby Ward. The Social Intranet Study, 2011.

[Wik11] Wikimedia Foundation Public Policy Initiative, 2011. Online; `http://outreach.wikimedia.org/wiki/Public_Policy_Initiative` [accessed July 2012].

[WLS$^+$07] Hai H. Wang, Yuan-Fang Li, Jing Sun, Hongyu Zhang, and Jeff Z. Pan. Verifying feature models using OWL. *J. Web Sem.*, 5(2):117–129, 2007.

[YA12] M. Lisa Yeo and Ofer Arazy. What Makes Corporate Wikis Work? Wiki Affordances and Their Suitability for Corporate Knowledge Work. In *Seventh International Conference on Design Science Research in Information Systems and Technology*, DESRIST'12, pages 174–190, 2012.

[YMY$^+$08] Gil Yehuda, Kyle McNabb, G. Oliver Young, Sara Burnes, and Zachary Reiss-Davis. Forrester TechRadar For I&KM Pros: Enterprise Web 2.0 For Collaboration, 2008.

[ZKK12] H. Zhu, R. Kraut, and A. Kittur. Organizing without Formal Organization: Group Identification, Goal Setting and Social Modeling in Directing Online Production. In *Proceedings of the 2012 ACM conference on Computer supported cooperative work*, CSCW'12. ACM, 2012.

Acknowledgments

Let me start by adapting the famous quote of Henry Ford "Coming together was the beginning, keeping together has been progress, and working together is success." Joining Onekin was only the beginning, and during this time I have met many people that have really contributed to get this thesis ready. That is why, with the following words, I try to thank everyone that has somehow participated in it.

First and foremost, I would really like to thank my supervisor Prof. Oscar Díaz for his invaluable support, patience and advice. His ideas and way of thinking have really inspired me, and will certainly mark my professional future. I would like to express my gratitude to Raúl Miñón, who encouraged me to contact Oscar and, above all, for our friendship.

Oscar is the leader of the Onekin research group at the University of the Basque Country (UPV/EHU). This group eagerly promotes collaboration, and the success of a member implies the effort of others. Particularly, from those who have been there from the very beginning, Cristóbal Arellano was always ready to help in any aspect from Java coding or iPhone jailbreaks to red tape; Sandy Pérez provided first-hand news ranging from Apache projects or continuous delivery to beer fairs; and Maider Azanza contributed with fruitful discussions about MDE, life and cooking, and then she goes and reviews this thesis!. All you, hope our paths cross again. I would like to show my appreciation to Jon Iturrioz who has been an understanding mate; and Arantza Irastorza who volunteered (really!) for reviewing this dissertation, putting a great effort on it. Of course, I am glad for having spent time with the remaining members of Onekin:

Luis M. Alonso, Iker Azpeitia, Jokin García, Felipe Ibáñez, Felipe Martín, Itziar Otaduy, Iñaki Paz and Iñigo Aldalur. We shared many moments, discussions and coffe breaks.

This thesis was economically supported by the Spanish Ministry of Education and Science, which permitted me to work full time on my research and also to perform a research visit to the IWIS research group at the Aalborg University (Denmark). I would like to thank Peter Dolog, leader of the IWIS research group, for giving me the opportunity to work with them and dedicating their time to me. My gratitude to Fred Durao and Ricardo Lage for welcoming me to Aalborg.

One of the best things this PhD has given to me is the opportunity to meet, work and even discuss with renowned internationally experts. Specially, I would like to thank Jesús Garcia Molina, who leads the ModelUM research group at the University of Murcia, Javier Luis Cánovas Izquierdo and Jesús Sánchez Cuadrado. All of them have patiently helped and explained me the nitty-gritty details of MDE, although it had a hefty price, they eat a lot. Antonio Vallecillo, leader of the Atenea Systems Modeling Group, who taught me about transformations chains, and that it rains in Málaga, a lot. That also had a hefty price, I eat a lot.

I would also like to thank the external reviewers, Ademar Aguiar and Christian Wagner, and the members of the dissertation committee, Antoni Olivé, Antonio Vallecillo, Ioanna Lykourentzou, Oscar Pastor and José Miguel Blanco, for their invaluable work.

To all my friends, Unai, Vanesa, Buba, Juantxo, Ana, Fernando, Silvia, Birujo, Bea for all the good times, and because a dinner with you is worth much.

To all my relatives but specially to my parents, Javi and Atxen, and my brother Jon. They brought me up, and during all my life they supported me both economically and morally, always encouraging me to improve myself.

With special affection to Esme. She is always there, and she will be. Never stop smiling.

Epilogue

Oscar Wilde once said: “*Education is an admirable thing. But it is well to remember from time to time that nothing that is worth knowing can be taught*”. Throughout these years, I have learned things that go way beyond these chapters. Now I face a new stage where I can apply what I learned about hard work, passion, and, above all, perseverance.

Vita

Gorka Puente García was born in Vitoria-Gasteiz, Spain on May 18th, 1983, son of Francisco Javier Puente Prieto (father) and María Ascensión García Ozaeta (mother). Gorka is brother of Jon Puente García and fiancé of Esmeralda Ramírez Sánchez. Gorka obtained the Bachelor of Science (BSc in 2006) and the Master of Science (MSc in 2009) in Computer Science at the University of the Basque Country (UPV/EHU). Inbetween, during 2007 he enjoyed a grant for an intership and spent eight months in Edinburgh (Scotland).

puente.gorka@gmail.com

This dissertation was typed by the author.