Design and Implementation of Wiki Content Transformations and Refactorings

Hannes Dohrn
Friedrich-Alexander-University
Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen, Germany
+49 9131 85 27621
hannes.dohrn@fau.de

Dirk Riehle
Friedrich-Alexander-University
Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen, Germany
+49 9131 85 27621
dirk@riehle.org

ABSTRACT

The organic growth of wikis requires constant attention by contributors who are willing to patrol the wiki and improve its content structure. However, most wikis still only offer textual editing and even wikis which offer WYSIWYG editing do not assist the user in restructuring the wiki. Therefore, "gardening" a wiki is a tedious and error-prone task. One of the main obstacles to assisted restructuring of wikis is the underlying content model which prohibits automatic transformations of the content. Most wikis use either a purely textual representation of content or rely on the representational HTML format. To allow rigorous definitions of transformations we use and extend a Wiki Object Model. With the Wiki Object Model installed we present a catalog of transformations and refactorings that helps users to easily and consistently evolve the content and structure of a wiki. Furthermore we propose XSLT as language for transformation specification and provide working examples of selected transformations to demonstrate that the Wiki Object Model and the transformation framework are well designed. We believe that our contribution significantly simplifies wiki "gardening" by introducing the means of effortless restructuring of articles and groups of articles. It furthermore provides an easily extensible foundation for wiki content transformations.

Categories and Subject Descriptors

H.4 [**Information Systems**]: Information Systems Applications; I.7 [**Computing Methodologies**]: Document and Text Processing; D.2 [**Software**]: Software Engineering

General Terms

Design, Languages

Keywords

Wiki, Wiki Markup, WM, Wiki Object Model, WOM, Transformation, Refactoring, XML, XSLT, Sweble

WikiSym '13, August 05 - 07 2013, Hong Kong, China

1. INTRODUCTION

The first wiki was created in 1995 by Ward Cunningham[1]. In the following years the wiki idea was widely adopted and today there are more than a hundred wiki engines available[2] which offer a wide variety of features. One central abstraction in the original wiki idea is the use of a "dirt simple" [14] wiki markup language (WML) to author articles. This wiki markup is then rendered as hypertext markup (HTM) by the wiki engine for viewing in a browser.

Since then wikis have evolved many new features, however, two features are still painfully lacking in most engine implementations: (a) a powerful abstraction of the content stored in articles that facilitates visual editing, content transformation, querying, data exchange and storage and (b) automated transformations that aid authors and maintainers in applying modifications consistently within one article or over multiple articles in a wiki.

We believe that these shortcomings are both related to the above-mentioned central abstraction of using a markup language to represent content. While wikis evolved so did their markup languages. Today most wiki engines are faced with the fact that working with the content in their wikis is extremely difficult due to a format that does not lend itself to processing by a computer program. Since computers are barely able to access the wealth of information stored in WMLs, features like visual editing or automated content transformation are only slowly entering the world of wikis.

In [7] we have demonstrated that it is possible to implement proper parsers for today's WMLs. We did so by tackling one of the most notorious WML dialects, namely the wiki markup used by MediaWiki. In the work at hand we shift the focus on a richer abstraction of the content that has thus become possible and transformations of wiki content enabled by this abstraction.

At this point we also like to clarify our usage of the terms refactoring and transformation. The term *refactoring* was originally used to describe behavior preserving transformations of program code written in languages like Smalltalk, C++, or Java [15]. Fowler defines "Refactoring [as] the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure." [9]

The term has also found its way into literature about wikis [6, 16, 17] to describe transformations which for example split the content of one article into two articles. However, in the

[1] `http://c2.com/cgi/wiki?WikiHistory`
[2] `http://www.wikimatrix.org`

context of a wiki it is not the behavior of a software system that is of interest but the body of knowledge that a wiki imparts to the reader.

We therefore propose to use the term *refactoring* for a transformation in which the internal representation of articles is altered in such a way that the rendered pages of the wiki, which are presented to the user, will not have changed in any way. Examples for refactorings are transformations of programming-language constructs embedded in articles or pretty-printing of the internal representation.

This article makes the following contributions:

- We propose a framework for defining and performing transformations on wiki articles.
- We present a catalog with basic transformations and refactorings applicable to most wiki markup languages.

The remainder of this paper is organized as follows. In section 2 we will discuss the state of the art in wiki engine technology and transformations in the wiki. In section 3 we introduce the model over which we define and perform transformations. In section 4 we present a catalog of transformations and discuss selected transformations in detail. Finally, section 5 will discuss the results of our work and section 6 provides an outlook on future research in this field.

2. STATE OF THE ART

2.1 Representation of article content

Article content is usually stored as wiki markup which is a purely textual representation of data that, if analyzed properly, exhibits a rich structure. The work presented in this paper is based on our findings in [7], where we show how to implement a parser that can accurately convert wiki markup into an abstract syntax tree (AST) without loosing the original formatting of the WM.

While we were the first to present a fully-functional parser for a complicated WML, the realization that wiki content should be represented by well-structured data has been around for some time. We believe that two schools of thought can be identified.

The first school of thought addresses the front-end of wikis directly by advocating a standardized markup language called WikiCreole [18] that was supposed to be shared by all wiki engines. An important point of their work was that the language was well-founded by a formal grammar [11] and an XML format [12] which in turn would have enabled wiki developers to easily generate an AST from WM.

The second school of thought attacks at the back-end and is mostly championed by developers of semantic wikis. Unlike unifying the front-end of wikis by demanding a common WML for all wikis, standardizing the back-end of wikis does not interfere with the user experience of editing in a wiki.

Völkel et al. were among the first to present a structured data format, called *Wiki Interchange Format* (WIF) [25], to enable processing and interchange of data between wikis. Schaffert et al. describe the design principles of the semantic wiki *IkeWiki* [19] which stores pages in an XML format that is also called *wiki interchange format* (WIF). The DBpedia project is a community effort with the goal of extracting information from Wikipedia and providing that information as structured data [1].

Buffa et al. presented the semantic wiki *SweetWiki* [2] in 2006 with XHTML as storage format, motivating the choice with the difficulties of implementing a WYSIWYG editor for a WML. In 2011 they presented *SweetWiki* again [3] introducing a *wiki object model* which they describe as "an ontology of the wiki itself" that allows the system to perform queries about the wiki from within articles.

All work on interchange formats and wiki object models proposes the use of a higher-level representation of wiki content but does not provide a definition. The only exception we are aware of is our own work in which we followed the second school of thought by defining and implementing a Wiki Object Model [8].

2.2 Refactoring in the Wiki

Not only the software that drives a wiki evolves, the content stored in a wiki's articles also evolves. When articles grow beyond a certain size, it can be better to split the article into smaller ones. If two authors start two articles with different names but which talk about the same subject, such articles should be merged. There are many more cases in which changing the content structure of a wiki becomes necessary.

On huge wikis such as Wikipedia[3] the continuous degradation of content structure causes considerable maintenance work[4]. Unfortunately, in many cases programs are not yet able to fix such issues automatically. Whether to split a section and how to split a section can only be decided by a human. However, no matter how involved the task, in the end one or multiple articles will have to be altered.

In software engineering a discipline has emerged that tackles these same problems in the domain of program design. The individual steps in the process of improving the design of software are called refactorings. Fowler defines "Refactoring [as] the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure." [9] It is still the human who has to decide what and how to refactor, but in software engineering it is the machine that helps applying the changes.

To the wiki community the term refactoring is not new. A search on the web[5,6] shows that the term refactoring is occasionally used to describe the process of changing the structure of a wiki article without changing the meaning it conveys. But in the case of wikis most of the time it is still a human who does all the work. From deciding that a refactoring has become necessary to working out the details of how to refactor and finally performing the legwork of actually carrying out the necessary changes.

As long as a refactoring only affects one or two individual articles at a time it is still feasible. However, if a refactoring affects multiple articles maintainers need computer support. In Wikipedia so-called bots [10] can perform such support tasks. Bots are programs which crawl the wiki from time to time in search for articles which they have to alter according to their assigned task. To this end maintainers file a bot request and a bot operator has to program the bot to perform the requested task the next time it runs. While better than performing the work manually it is still hard work. Moreover bots and people who can program bots are not available to

[3] http://www.wikipedia.org/

[4] http://en.wikipedia.org/wiki/Wikipedia:Maintenance

[5] http://c2.com/cgi/wiki?WikiRefactoring

[6] http://c2.com/cgi/wiki?RefactoringWikiPages

every wiki community.

Rosenfeld et al. [17] and Puente et al. [16] have introduced refactorings similar to those found in software engineering to wikis. Rosenfeld et al. perform refactorings in semantic [20] wikis. Consequently the presented 7 refactorings operate on semantic meta-data that is mostly stored in well-defined data structures outside the actual articles. Puente et al. demonstrate the benefits of using a mind map as user interface for applying refactorings. They present 8 refactorings that operate on meta-data (categories) and from whole articles down to section level.

2.3 Specification of Refactorings and Refactoring Data Structures

In order to specify refactorings and transformations we take a look at the work done in the field of refactorings for programming-languages. We will later reuse these techniques to describe transformations in the context of wikis.

When Opdyke wrote his thesis on refactoring [15] he used prose and mathematical expressions to describe the individual refactorings. The descriptions consisted of a set of operations (prose) and a set of preconditions (mathematical expressions) that had to be fulfilled if the refactoring was to be applied. To specify complex refactorings, Opdyke first introduced a set of simple, basic refactorings that were later used by the complex refactorings.

3. GENERALIZED WIKI OBJECT MODEL

3.1 From Wiki Markup to an Object Model

Transformations can be defined over wiki markup as well and this is the method most programmers choose that have to transform wiki articles. However, this method has disadvantages. Two disadvantages stem directly from the fact that WMLs are usually not formally defined: (i) This makes it impossible to specify transformations formally and (ii) it is not possible to specify transformations for wikis in general since one always has to focus on one specific WML.

The fact that there is no formal specification also explains why for most WMLs parsers that could produce a higher-level representation are not available. Without a higher-level representation transformations have to be implemented using complex textual search and replace techniques down to algorithms which examine the WM character by character. Moreover, with no parser at hand pure text transformations are error-prone since some syntactic constructs are hard to distinguish in WMLs.

As solution we propose to generate a higher-level representation from the WM first and then perform the transformation on the higher-level representation as illustrated in figure 1 [4]. One way of obtaining such a representation is the transformation from WM to an abstract syntax tree (AST) by a parser. While implementing parsers for WMLs is certainly not trivial we have shown in [7] that it is possible and, if the parser is driven by a formal grammar, one formally defines the respective WML at the same time.

We also want to specify transformations for wikis in general, not only for one specific syntax. Parsers on the other hand usually generate an abstract syntax tree that, as the name suggests, is still tied to syntactic idiosyncrasies of the language. We solve this issue by further abstracting to a generalized wiki object model. Such a model will represent features commonly found in WMLs in an easy, accessible and standardized way while the model fully preserves WML-specific information at same time. The details of this data structure are explained in section 3.2.

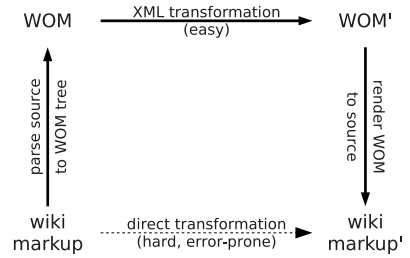

Figure 1: Changing the domain simplifies the implementation of transformations.

Figure 1 illustrates the use of the WOM. Instead of taking the difficult path by directly transforming from WM to WM′, one can first change the domain. Once in the domain of the WOM transformations are easy and safe (they can still be semantically wrong though). We argue that changing the domain twice and performing the transformation over the WOM has more advantages than disadvantages over trying to directly transform wiki markup.

3.2 Generalizing the WOM 1.0

In order to be able to discuss and rigorously define transformations we have to establish a data model. The transformations are applied to instances of this model. While we mainly experiment with MediaWiki's WML we want to show that the model as well as the transformations defined over this model are applicable to all wikis which follow the original design principle of using WM as a front-end to HTM.

To become independent of MediaWiki's idiosyncrasies we take the Wiki Object Model as defined in [8] and generalize it. Listing 1 shows a short article in MediaWiki's WM serialized as WOM XML and as rendered page to give the reader an idea of the structure and elements of an article.

The WOM defines a document as a tree-like structure, similar to the Document Object Model (DOM) of HTML [22]. The nodes of this structure are corresponding to those elements commonly found in WMLs. For a detailed listing and explanation of all available nodes, please refer to [8].

To make it possible to represent arbitrary wiki markup the following new tags were added:

Arbitrary elements A generalized WOM cannot support every feature of every possible wiki engine. The `e` tag is used to encode arbitrary elements in a WOM document like a TikiWiki *simple box*[7] or a MediaWiki *transclusion*[8]. Both elements are not commonly available in different wiki engines and therefore have no equivalent in the WOM.

Substitution Arbitrary elements are only meaningful to those wiki engines that exported the article as WOM document (and therefore support the respective feature). To preserve as much information as possible for other engines who do not support such features the

[7] `http://tiki.org/WikiSyntax`

[8] `https://www.mediawiki.org/wiki/Transclusion`

```
The '''Sweble''' Wikitext parser is an
[[Open-source software|open-source tool]]
to [[Parsing|parse]] the [[Wikitext]]
[[markup language]] used by [[MediaWiki]],
the software behind [[Wikipedia]].

== The current state of parsing ==
* list item 1
* list item 2

== How Sweble works ==
another section

[[Category:Parsing]]
```

The **Sweble** Wikitext parser is an open-source tool to parse the Wikitext markup language used by MediaWiki, the software behind Wikipedia.

The current state of parsing

- list item 1
- list item 2

How Sweble works

another section

```
<page title="Sweble" version="1.0">
  <body>
    <p>
      <text>The </text>
      <b><text>Sweble</text></b>
      <text> Wikitext parser is an </text>
      <intlink target="Open-source_software">
        <title><text>open-source tool</text></title>
      </intlink>
      <text> to </text>
      <intlink target="Parsing">
        <title><text>parse</text></title>
      </intlink>
      <text> the </text>
      <intlink target="Wikitext"/>
      <text> </text>
      <intlink target="Markup_language"/>
      <text> used by </text>
      <intlink target="MediaWiki"/>
      <text>, the software behind </text>
      <intlink target="Wikipedia"/>
      <text>. </text>
    </p>
    <section level="2">
      <heading>
        <text> The current state of parsing </text>
      </heading>
      <body>
        <ul>
          <li><text> list item 1</text></li>
          <li><text> list item 2</text></li>
        </ul>
      </body>
    </section>
    <section level="2">
      <heading>
        <text> How Sweble works </text>
      </heading>
      <body>
        <p><text>another section</text></p>
        <p><intlink target="Category:Parsing"/></p>
      </body>
    </section>
  </body>
</page>
```

Listing 1: A severely reduced version of the article "Sweble" from the English Wikipedia. In the top left corner the WM of the article is shown. To the right of the WM the rendering of the page is shown. Below a simplified version (collapsed whitespace, no RTD) of the WOM XML for this article is printed.

substitution tag `subst` is introduced. It contains two child elements, the replacement tag `repl` and the `for` tag. The replacement tag contains an alternative representation of the unsupported tag. The `for` tag contains the `e` tags which will allow the exporting wiki to restore the original element.

Round-Trip Data WMLs, like programming languages, offer certain degrees of freedom to the author concerning the formatting of the WM or code. Furthermore, every wiki engine has its own markup syntax and some wiki engines offer alternative syntaxes to express the same thing. If a WOM document only stores semantic information neither the exact WM formatting nor the original WM syntax can be restored. To resolve this problem `rtd` tags are interspersed in the WOM document and text nodes must be wrapped in `text` tags to be considered when restoring the original markup in allusion to [5].

To illustrate how those three kinds of elements work together consider the following WM (MediaWiki): "`[[Dog]]s`". The WOM XML produced by our implementation is printed in listing 2. The WM denotes an internal link pointing to another article called *Dog*. However, the author wants to use the plural. MediaWiki syntax allows authors to append or prepend text directly to the link to the effect that the pre- or postfix will be rendered as part of the link. Without such a feature an author would have to repeat the whole word in its plural form as link title (e.g. "`[[Dog|Dogs]]`").

```
...
  <subst><repl>
      <intlink target="Dog">
        <title><text>Dogs</text></title>
      </intlink>
    </repl><for>
      <e postfix="s" target="Dog" type="intlink">
        <rtd>[[Dog]]s</rtd>
      </e>
  </for></subst>
...
```

Listing 2: Example of how the `e`, `subst`, and `rtd` tags work.

Most wiki engines do not support link pre- or postfixes which is why the generalized WOM doesn't support them. As remedy our implementation uses the `e` tag to preserve the use of a link postfix. Since the `e` tag is meaningless to other wikis the `subst` tag and its child tags are used to provide an alternative representation. Note that the alternative representation should imitate the original meaning as well as possible, however, this is not always possible. In this example the original meaning is fully preserved but syntactic subtleties are lost.

Finally, `rtd` and `text` tags are used to also preserve the original WM. By concatenating all text nodes returned by the following simple XPath [23] expression the original WM can be restored:

```
//text[not(ancestor::repl)]|
//rtd[not(ancestor::repl)]
```

This expression selects all `text` and `rtd` nodes which are not descendants of a `repl` tag. RTD information in `repl` tags is excluded to prevent duplication with the RTD information in the `for` tags.

4. TRANSFORMATIONS

4.1 Applying a Transformation

The content of a wiki is not exclusively defined by its articles. Especially semantic wikis maintain rich graphs of meta-data that are not stored within the article source itself but in RDF triple stores or relational databases [19, 21].

Another kind of information that is stored outside of the article source is what Sint et al. [21] call management data. It comprises authorship information, edit messages and time-stamps, access restrictions, etc. but also redundant meta-data that is usually computed from the articles and is needed by the engine for better performance, queries or analytics. Examples are inter page link indices or full-text search indices.

When talking about transformations we only consider the transformation of information that is commonly found in articles. We furthermore assume that once the transformation has been applied the management meta-data will be updated by the respective wiki engine.

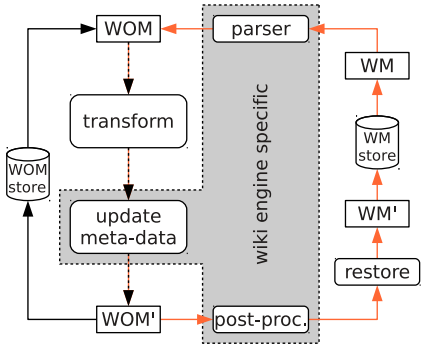

Figure 2: The transformation pipeline

The whole process of applying a transformation is illustrated in the left half of figure 2. It starts by retrieving all affected articles as WOM documents from storage to which the transformation is then applied. At this stage, the resulting WOM′ has undergone a complete semantic transformation. Articles can now be rendered from the transformed WOM′ documents into a presentation format like HTML and the user will see the result of the transformation.

Before the whole operation is finished the management meta-data required by most wikis has to be updated. This also happens when an author manually changes an article through an editor front-end and thus does not introduce a new processing step to a wiki.

4.2 Restoring Wiki Markup

In the above walkthrough we assumed that the wiki engine stores its articles as WOM documents. At the time of writing this is not the case in most wikis and a transformation operation becomes more involved if a wiki offers its users to textually edit articles in a WML. To better understand the process we take a look at the right half of figure 2.

Before the transformation can be applied the retrieved WM has to be converted to a WOM document. Afterwards processing proceeds as discussed above until the management meta-data has been updated. At this point the transformed WOM′ document has to be converted back to WM′. As shown in section 3.2 this is a straightforward operation. However, since the transformation is a generic process which does not know about the syntactic intricacies of the underlying WML the `rtd` tags cannot be transformed correctly.

This is problematic if elements of the document are rewritten or new elements are added. When for example text is formatted as bold a generic transformation cannot know what bold WM looks like for a specific wiki engine.

To fix these issues another engine-specific work step, called post-processing, has to be introduced. After updating the management meta-data and before converting the WOM′ document back into WM′, all elements which were newly added or altered as well as all neighbors of altered, removed or added elements have to be inspected and if necessary RTD information has to be added or updated.

4.3 Elements in the WOM

Before we provide a catalog of basic transformations we have to identify the elements of wiki articles to which transformations can be applied. We will not go into detail here since most of an article's elements are borrowed from the HTML 4 [22]. Instead we will operate on major structural elements, elements that are not found in HTML 4 and categories of elements. For clarity we will write elements in angle brackets (e.g. `<section>`) and groups of elements in curly brackets (e.g. {formatting}). We identified the following categories and elements relevant to the transformations presented in this paper:

{list}:	`<ul>`, `<ol>`, `<dl>`
{list item}:	{definition list item}, `<li>`
{definition list item}:	`<dt>`, `<dd>`
{table cell}:	`<th>`, `<td>`
{content block}:	`<section>`, `<table>`, `<blockquote>`, `<pre>`, `<p>`, {list}
{running text}:	{formatting}, `<intlink>` (inter/intra wiki link), `<extlink>` (link to URL), `<text>`
{formatting}:	HTML layout tags (` `, etc.), HTML formatting tags (`<span>`, `<b>`, `<small>`, etc.), HTML semantic formatting tags (`<abbr>`, `<code>`, etc.)

4.4 A Catalog of Transformations

4.4.1 *Fundamental Transformations*

While most transformations are simple to implement and describe others can be complex. To simplify the implementation of complex transformations we introduce *fundamental transformations* which can be combined into more complex transformations. They are not disclosed to the user. We will give one example of such a transformation.

wrap-in-p: *A block of {running text} is not necessarily wrapped in a paragraph. This is often the case if {running text} is located in a {list item} or a {table cell} element. If these elements are dissolved it might become necessary to wrap the {running text} in a paragraph instead.*

The `wrap-in-p` transformation will be used in an example in section 4.5.1.

4.4.2 Creation and Removal Transformations

In order to populate an article elements can be created and also removed. Some elements are created explicitly (like a <section>), others implicitly (e.g. the <heading> and <body> element that are automatically created when a section is created). When creating elements in an article the elements are inserted into the WOM document without RTD information attached. Post-processing will see to it that newly added elements are properly formatted.

When removing elements from an article whole subtrees are removed. This means that not only the element is removed but all its children and children's children are also removed. Such an operation can remove elements that are referenced elsewhere. It is the duty of removal transformations to walk the subtree that is marked for removal and look for elements that are referenced elsewhere. If such elements should be found the user has to be informed to decide how to proceed.

4.4.3 Generic Transformations

Generic transformations apply to many different elements. Table 1 gives an overview of the transformations presented in this section and the elements they apply to. These transformations are offered to users and sometimes rely on other transformations to complete their task.

dissolve (unwrap): *Pulls the children of an element in front of the element and then removes the element.*

convert-into: *Converts one kind of element into another kind of element.*

merge-with: *Merges two adjacent elements of the same kind.*

move-to (drag&drop): *Select an item and move it to another location in the article.*

move-up-down: *Two adjacent elements swap positions moving one of them down and the other one up.*

promote-demote: *Promote or demote an element on a certain level to a higher/lower level.*

pretty-print: *Reformat the WM stored in the RTD information of the WOM document.*

rename: *Rename a resource and update all references to that resource.*

move-to-article: *Select an item and move it to a location within another article.*

4.4.4 Specialized Transformations

The transformations presented in this section are specific to one type of element only.

Transformations for <table>s

move-row-column *Move a row or column of table cells to a different location in the table.*

join-split-cells *Join two adjacent cells into one cell spanning both cells or split a spanning cell into individual cells.*

Transformations for {running text}

wrap-selection *Wrap a selection of {running text} into a {formatting} element.*

clear-formatting *Remove all {formatting} from a selection of {running text}.*

Transformations for <extlink>s

replace-url *Search a URL in all articles and replace that URL with a different URL.*

Transformations for {media}

replace-caption *Search for all {media} elements that refer to a specific resource and replace their captions with the given caption.*

4.4.5 Wiki Transformations

Wiki transformations affect the wiki's top-level elements, the articles and categories. Puente et al. [16] describe the following transformations on *articles*: **rename-article**, **merge-articles**, **split-articles**, **add-article-to-category**, and **remove-article-from-category**. Rosenfeld et al. [17] describe the following transformations on *categories*: **remove-category**, **merge-categories**, and **change-subcategory-relationship**. Puente et al. [16] describe the following transformations on *categories*: **split-category** and **rename-category**.

4.5 Specifying and Implementing Transformations

The WOM can have many different representations. The WOM XML representation was already used in this work and lends itself well to data exchange, storage in document oriented databases but also for discussing the WOM. Another representation are data structures in programming languages. To work with the WOM in a program we've presented a set of Java interfaces in [8]. When implemented, the WOM is represented by a graph of objects. This representation is useful for directly manipulating a document, is easily accessible in a program and very efficient in that manipulations and queries are fast as are conversions into other formats.

In order to describe and discuss transformations we propose the use of the *eXtensible Stylesheet Language: Transformations* or in short XSLT 2.0 [24]. We decided to use XSLT for the following reasons:

- XSLT is a high-level document transformation language [13].
- XSLT is a "lingua franca" for transformations on the web. Bindings to libraries which can perform XSLT transformations are available for many major programming languages.
- XSLT is a side-effect free language [13]. This simplifies reasoning about the effects of transformations.

Whether transformations are implemented using XSLT or whether to use an in-memory object model which is manipulated programmatically remains the choice of the developer. XSLT has the advantage of not being bound to a specific programming language. It can also be exposed to the user who can provide his own transformation scripts at run-time. In contrast, we believe that the main argument for specifying transformations programmatically is performance.

4.5.1 Example 1: Dissolving a List

As a first example we show how a list is dissolved. The wiki markup of the article we want to transform is given in listing 3. The article consists only of an unordered list with 3 items. The first item contains plain text, then an explicit paragraph and then plain text with {formatting} markup. The next item contains a nested list and the last item only contains plain text.

The `dissolve` transformation will dissolve the list and the list items but not the content of the list items. The content will be wrapped in paragraphs and those paragraphs will then replace the list and its items. When dissolving simple {formatting} markup one can simply replace the formatting element with its content, thus "unwrapping" the content and removing the formatting.

Table 1: Matrix listing basic transformations and the elements to which they apply.

	dissolve (unwrap)	convert into	merge with	move to/ (drag&drop)	move up/down	promote/ demote	pretty print	rename	move to article‡
<section>	✓§	–	–[1]	✓§	✓§	✓§	✓	✓[2,†,§]	✓†
<p>	–	✓[3]	✓[4]	✓	✓	–	✓	–	✓
<table>	–	–	–	✓	✓	–	✓	–	✓
,	✓	✓[5]	✓[6]	✓	✓	–	✓	–	✓
<dl>	✓	–	✓[7]	✓	✓	–	✓	–	✓
	✓	–	✓[8]	✓	✓	✓	✓	–	✓
<dt>, <dd>	✓	–	✓[9]	✓	✓	✓	✓	–	✓
<hr>	–	–	–	✓	✓	–	✓	–	✓
<intlink>	✓	–	–	✓	–	–	✓	✓[10,$]	✓
<extlink>	✓	–	–	✓	–	–	✓	–	✓
{media}	–	✓[11]	–	✓	–	–	✓	✓[10,$]	✓
{running text}	–	–	–	✓	–	–	✓	–	✓
<category>	–	–	–	–	–	✓†	✓	✓†,$	–

[1] use dissolve to merge into previous section
[2] rename <heading>
[3] transform into /<li>, /<li>, <section>/<heading>
[4] merge with <p>
[5] transform into <ul>,
[6] merge with <ul>,
[7] merge with <dl>
[8] merge with
[9] merge with <dt>, <dd>
[10] rename link *target*
[11] transform into <intlink>, <extlink>
† Mentioned by Puente et al. in [16]
§ Since sections can be referenced the transformation may affect multiple articles.
‡ Affects two articles. More if a referencable element was moved.
$ References have to be updated.

Dissolving a list is more involved since the individual items have to be converted into paragraphs which is not straightforward. To simplify the definition of more complex transformations we extract fundamental transformation steps as individual transformations. In case of the `dissolve` transformation we extracted the `wrap-in-p` transformation.

The Transformation Script

In line 9 of listing 3 the element we want to dissolve is given as parameter *e*. Usually the parameter would be injected from outside the XSLT but for demonstration purposes we set the parameter in the declaration using an XPath expression to select the list. The `dissolve` transformation is coded in lines 11 – 27 as modified identity transformation that copies all elements it encounters in the source document into the result document.

Templates in XSLT are a way to produce output. The modified identity transformation template in line 11 matches any element of the input and is therefore invoked when the XSLT processor starts work by trying to invoke a template for the root node `/`. If the current node is not *e* the `otherwise` case in line 20 will be instantiated and the template will copy the current node (not its children) to the result tree. Then templates are applied recursively to the children of the current node which will in turn call the template again to now copy the children.

This continues until the current element is *e*. Instead of copying the list it applies the `wrap-in-p` template to the first child of each of its list items in line 15 and 17. By applying the template `wrap-in-p` the list will effectively be replaced by whatever nodes the invoked template instantiates.

Lines 29 – 43 define helper functions which we will not elaborate. Notice that this is a feature of XSLT 2.0. The transformation works with XSLT 1.* as well but the script would be more verbose to compensate the lack of expressiveness.

The remainder of the script defines the fundamental transformation `wrap-in-p`. It is the task of the `wrap-in-p` transformation to find sequences of non-{block content} elements and wrap them into paragraph elements. {Block content} elements are copied to the result tree unaltered. The transformation is called on the first child node of a list item and searches among its siblings until it finds a {block content} element (stored in `$first-block`)

If there is a {block content} element we try to wrap the node sequence from the current node (inclusive) to the first {block content} element (exclusive) (stored in `$head`) in a paragraph element. To prevent generating empty paragraphs we only wrap a non-empty sequence. It is important to **exclude** RTD information from the sequence. The siblings the script currently operates on are children of the list items. Any RTD information encountered here is therefore specific to the list and its items. Since we want to remove those, we also have to remove their RTD information.

After wrapping any non-{block content} elements to the result tree the {block content} element `$first-block` is copied to the result tree and processing continues by recursively invoking the `wrap-in-p` template on the sibling that comes after the {block content} element `$first-block`.

If the `wrap-in-p` template does not find a {block content} element from its current position the remaining nodes are wrapped and written to the result tree in lines 68 – 77. Again we have to test for an empty sequence and make sure that no list RTD information is leaked to the result tree.

Post-processing

After applying the transformation as described above the resulting WOM′ document (see listing 3.5) can be rendered and the user will see the intended result. However, if the WOM′ document would be transformed back into wiki markup at this point, the WM′ (see listing 3.3) will **not** correspond to the WOM′ document. This can be automatically tested by parsing the WM′ into a WOM″ document again and comparing both WOM′ and WOM″.

The reason for the discrepancy lies in the RTD information.

Listing 3.1 – The transformation script:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
    xmlns="http://sweble.org/schema/wom"
    xmlns:loc="local stuff"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:wom="http://sweble.org/schema/wom"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:param name="e" select="/wom:page/wom:body/wom:ul" />

  <xsl:template match="*">
    <xsl:choose>
      <xsl:when test=".=$e">
        <!-- . = the list -->
        <xsl:for-each select="./*/*[1]">
          <!-- . = the first child of the first list item -->
          <xsl:call-template name="wrap-in-p" />
        </xsl:for-each>
      </xsl:when>
      <xsl:otherwise>
        <xsl:copy>
          <xsl:copy-of select="@*"/>
          <xsl:apply-templates />
        </xsl:copy>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <!-- Test if $e is a block element -->
  <xsl:function name="loc:is-block" as="xs:boolean">
    <!-- ... --></xsl:function>

  <!-- Test if $e is a RTD element -->
  <xsl:function name="loc:is-rtd" as="xs:boolean">
    <!-- ... --></xsl:function>

  <!-- Return the node sequence "$e|$e/following-sibling::*" -->
  <xsl:function name="loc:following-sibling-incl">
    <!-- ... --></xsl:function>

  <!-- Return the sequence of all siblings between [$from, $to) -->
  <xsl:function name="loc:from-to-excl">
    <!-- ... --></xsl:function>

  <xsl:template name="wrap-in-p"><!-- Find the first block
      element, starting with the current node itself. -->
    <xsl:variable name="first-block"
      select="loc:following-sibling-incl(.)[loc:is-block(.)][1]" />
    <xsl:choose>

<xsl:when test="$first-block"><!-- There's a block element up ahead -->
  <xsl:variable name="head"
    select="loc:from-to-excl(.,$first-block)[not(loc:is-rtd(.))]" />
  <xsl:if test="$head"> <!-- We found non-block elements
      to wrap in front of the block element -->
    <p>
      <xsl:apply-templates select="$head" />
    </p>
  </xsl:if>
```

Listing 3.1 – The transformation script (continued):

```
  <!-- copy the block element as is -->
  <xsl:apply-templates select="$first-block" />
  <!-- Continue wrapping after the block element -->
  <xsl:for-each select="$first-block/following-sibling::*[1]">
    <xsl:call-template name="wrap-in-p" />
  </xsl:for-each>
</xsl:when>

<xsl:otherwise> <!-- There are no more block
    elements, wrap the remaining elements -->
  <xsl:variable name="remainder"
    select="loc:following-sibling-incl(.)[not(loc:is-rtd(.))]" />
  <xsl:if test="$remainder"> <!-- But only if there are any -->
    <p>
      <xsl:apply-templates select="$remainder" />
    </p>
  </xsl:if>
</xsl:otherwise>

    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

Listing 3.2 – The wiki markup *before transformation*:

```
* Item 1.1 <p>Item 1.2</p> Item <b>
    1.3</b>
** Item 2.1
** Item 2.2
* Item 3
```

Listing 3.3 – The wiki markup *after transformation*:

```
 Item 1.1 <p>Item 1.2</p> Item <b>
     1.3</b>** Item 2.1
** Item 2.2
 Item 3
```

Listing 3.4 – The wiki markup *after post-processing*:

```
Item 1.1 <p>Item 1.2</p>

Item <b>1.3</b>
* Item 2.1
* Item 2.2

Item 3
```

Listing 3.5 – The WOM′ XML *after transformation*:

```
...
  <p><text> Item 1.1 </text></p>
  <p>
    <rtd>&lt;p&gt;</rtd>
    <text>Item 1.2</text>
    <rtd>&lt;/p&gt;</rtd>
  </p><p>
    <text> Item </text>
    <b><rtd>&lt;b&gt;</rtd>
      <text>1.3</text>
      <rtd>&lt;/b&gt;</rtd></b>
  </p>
  <ul>
    <li>
    <rtd>**</rtd>
    <text> Item 2.1</text>
    <rtd>
</rtd>
    </li><li>
    <rtd>**</rtd>
    <text> Item 2.2</text>
    <rtd>
</rtd>
    </li>
  </ul>
  <p><text> Item 3</text></p>
...
```

While we did strip the RTD information for the list item nodes we did not update RTD information for newly added nodes, namely the paragraph nodes and the nested list items. This is done in *post-processing* by adding the necessary newlines in front of and after the paragraphs and by updating the list item prefixes (the "*" characters at the beginning of a line). Only now will the recovered wiki markup correspond to the intention of the transformation (see listing 3.4).

4.5.2 Example 2: Moving a Paragraph to another Article

Dissolving a list is a transformation local to one article. Moving a section from one article to another affects two articles, however, an XSLT script can only transform one document at a time. To overcome this limitation we introduce another element to the WOM: the <articles> node. Every time more than one article is affected the articles are stored in one document as children of the <articles> node. This way both articles are addressable by the XSLT script using XPath expressions.

4.5.3 Example 3: Renaming an Article

Renaming a resource like a category or an article affects every article in which the resource is referenced. One way to apply such a transformation is to put all articles into an <articles> node and thus apply the transformation to the whole wiki. Every article that does not reference the resource is copied unaltered, those who do contain a reference will be transformed.

However, this is impractical even for small wikis and impossible for big wikis since it requires too much processing power and memory resources. Hence, the set of affected articles has to be determined using an index or a similar data structure a priori. Such an index is typically part of the redundant meta-data of a wiki engine and for the purpose of this example we assume that the wiki engine does provide a means of quickly identifying all affected articles.

Once the set of affected articles is determined they are transformed separately. This is possible since the `rename-article` transformation can work with a single article at a time. Processing each affected article separately ensures that the memory resources of the machine on which the transformation is performed are not exhausted. Below the XSLT script for the `rename-article` transformation is printed (only the part that addresses internal links).

```
...
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()" />
    </xsl:copy>
  </xsl:template>

  <xsl:template match="wom:intlink[@target=$search]">
    <intlink>
      <xsl:attribute name="target">
        <xsl:value-of select="$replace" />
      </xsl:attribute>
      <xsl:copy-of select="@*[name()␣!=␣'target']" />
      <xsl:apply-templates
        select="node()[name()␣!=␣'rtd']" />
      <xsl:if test="not(./wom:title)">
        <title>
          <text><xsl:value-of select="@target" /></text>
        </title>
      </xsl:if>
    </intlink>
  </xsl:template>
...
```

5. RESULTS AND LIMITATIONS

By implementing various transformations from the list of examples presented in sections 4.4.1 – 4.4.5 we could confirm that the separation into a semantical transformation and the post-processing to rectify the RTD information is well chosen. The XSLT script for renaming an article in section 4.5.3 demonstrates this. The script focuses on the target of an internal link and how the title of an internal link changes when the target changes. The developer of a transformation does not have to deal with syntactic intricacies of the underlying WM. The question of how to for example map prefix and postfix of a MediaWiki internal link to an altered title is left to the wiki engine specific post-processing step.

Our tests show further that the duration of a transformation[9,10] is dominated by the conversion of WM to a WOM document[11] ($\mu \approx 27\text{ms}, M \approx 8\text{ms}, \sigma \approx 90\text{ms}$) and the XSLT transformation[12] ($\mu \approx 13\text{ms}, M \approx 4\text{ms}, \sigma \approx 44\text{ms}$). Performing a transformation programmatically (in Java in our case) is considerably faster ($\mu < 1\text{ms}, M < 1\text{ms}, \sigma < 1\text{ms}$) since the document is altered directly and is not completely recreated. If articles are stored as WOM documents in the data store and the transformation is implemented programmatically the process is dominated by the time it takes to serialize and de-serialize the articles from the data store. Finally, post-processing and conversion from the transformed WOM document to WM adds little to the overall transformation time (each have $\mu < 1\text{ms}, M < 1\text{ms}$ and $\sigma < 4\text{ms}$).

A limitation to our approach is the fact that we are restricted to the content of an article. Especially semantic wikis contain rich semi-structured data which is stored as meta-data outside the articles and is thus not accessible to a transformation.

Another limitation is the complexity of XSLT. Even though a wiki maintainer can focus exclusively on the semantics of a transformation learning XSLT is an entry barrier.

Finally, while renaming an article that is referenced in 1000 other articles takes less than two minutes on our test machine, there are articles and templates in the English Wikipedia[10] that are referenced in more than 10000 other articles. In such cases our unoptimized implementation would need up to 20 minutes which may not be acceptable and would demand further social protocol to coordinate a big transformation.

6. CONCLUSION AND FUTURE WORK

The wiki community has recognized the importance of assisting authors and maintainers with "gardening" their wikis. Recent work in this area has introduced methods from software engineering, namely refactorings, to wikis. However, so far refactorings for wikis focus either on meta-data or cannot alter the structure of articles below section level. We believe that these limitations stem from the fact that content in articles is barely accessible to computers. By introducing a generalized Wiki Object Model and by demonstrating how to rigorously specify and apply transformations we have overcome these limitations.

[9] Measured on an Intel Core 2 Quad Q9550 with 8 GB RAM, single threaded. The article *Hypothalamus* was renamed and 622 articles were processed. μ denotes the mean, M the median and σ the standard deviation over all processed articles. Articles are UTF-8 encoded with an average size of 30kB.

[10] We use a dump from the English Wikipedia dated 07/03/2012.

[11] Using the Sweble parser for MediaWiki's WM

[12] Using the *Saxon-HE 9.4* XSLT processor

Future work will focus on improving and extending the methods we have presented in this article. The transformations proposed here only give a cursory overview. We seek to extend the list of transformations and to add detailed descriptions to each transformation.

Another place for improvement is the scope of transformations. The methods we presented apply to information stored within articles. Information stored in meta-data and other data structures of the wiki is excluded. Future research will try to lift this restriction, thus allowing us to transform all kinds of information stored in wikis using the same transformation mechanism.

In conclusion, we believe that using XSLT to rigorously specify and discuss transformations is a good choice for experts. However, while XSLT scripts enable authors to specify transformations themselves without modifying the wiki software, the XSLT language is too complex for the layman. Future research will explore ways of empowering authors to specify transformations with simplified, more comprehensible tools.

7. REFERENCES

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: a nucleus for a web of open data. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] Michel Buffa and Fabien Gandon. Sweetwiki: semantic web enabled technologies in wiki. In *Proceedings of the 2006 international symposium on Wikis*, WikiSym '06, pages 69–78, New York, NY, USA, 2006. ACM.

[3] Michel Buffa, Fabien Gandon, Peter Sander, Catherine Faron, and Guillaume Ereteo. Sweetwiki: A semantic wiki. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1), 2011.

[4] Michael L Collard and Jonathan I Maletic. Document-oriented source code transformation using xml. In *Proceedings of 1st International Workshop on Software Evolution Transformation (SET'04), Delft, The Netherlands, Nov*, volume 9, pages 11–14, 2004.

[5] Michael L. Collard, Jonathan I. Maletic, and Andrian Marcus. Supporting document and data views of source code. In *Proceedings of the 2002 ACM symposium on Document engineering*, DocEng '02, pages 34–41, New York, NY, USA, 2002. ACM.

[6] 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, New York, NY, USA, 2011. ACM.

[7] Hannes Dohrn and Dirk Riehle. Design and implementation of the sweble wikitext parser: unlocking the structured data of wikipedia. In *Proceedings of the 7th International Symposium on Wikis and Open Collaboration*, WikiSym '11, pages 72–81, New York, NY, USA, 2011. ACM.

[8] Hannes Dohrn and Dirk Riehle. Wom: An object model for wikitext. Technical report, Technical Report CS-2011-05, University of Erlangen, Dept. of Computer Science, 2011.

[9] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1 edition, July 1999.

[10] A. Halfaker and J. Riedl. Bots and cyborgs: Wikipedia's immune system. *Computer*, 45(3):79–82, March 2012.

[11] Martin Junghans, Dirk Riehle, Rama Gurram, Matthias Kaiser, Mário Lopes, and Umit Yalcinalp. A grammar for standardized wiki markup. In *Proceedings of the 4th International Symposium on Wikis*, WikiSym '08, pages 21:1–21:8, New York, NY, USA, 2008. ACM.

[12] Martin Junghans, Dirk Riehle, and Umit Yalcinalp. An xml interchange format for wiki creole 1.0. *SIGWEB Newsl.*, 2007(Winter), December 2007.

[13] Michael Kay. *XSLT Programmer's Reference*. Wrox Press Ltd., Birmingham, UK, UK, 2nd edition, 2001.

[14] Bo Leuf and Ward Cunningham. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley Professional, April 2001.

[15] William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois, Urbana-Champaign, IL, USA, 1992.

[16] 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, pages 646–661, Berlin, Heidelberg, 2012. Springer-Verlag.

[17] M. Rosenfeld, A. Fernández, and A. Díaz. Semantic wiki refactoring. a strategy to assist semantic wiki evolution. In *Proceedings of the Fifth Workshop on Semantic Wikis (SemWiki 2010), co-located with 7th European Semantic Web Conference, ESWC*, 2010.

[18] Christoph Sauer, Chuck Smith, and Tomas Benz. Wikicreole: a common wiki markup. In *Proceedings of the 2007 international symposium on Wikis*, WikiSym '07, pages 131–142, New York, NY, USA, 2007. ACM.

[19] Sebastian Schaffert. Ikewiki: A semantic wiki for collaborative knowledge management. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE'06. 15th IEEE International Workshops on*, pages 388–396. IEEE, 2006.

[20] Sebastian Schaffert, François Bry, Joachim Baumeister, and Malte Kiesel. Semantic wikis. *software, IEEE*, 25(4):8–11, 2008.

[21] Rolf Sint, Stephanie Stroka, Sebastian Schaffert, and Roland Ferstl. Combining unstructured, fully structured and semi-structured information in semantic wikis. In Christoph Lange, Sebastian Schaffert, Hala Skaf-Molli, and Max Völkel, editors, *SemWiki*, volume 464 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[22] The World Wide Web Consortium. HTML 4.01 Specification. `http://www.w3.org/TR/html401`, December 1999.

[23] The World Wide Web Consortium. XML Path Language (XPath) Version 1.0. `http://www.w3.org/TR/xpath`, November 1999.

[24] The World Wide Web Consortium. XSL Transformations (XSLT) Version 2.0. `http://www.w3.org/TR/xslt20`, January 2007.

[25] Max Völkel and Eyal Oren. Towards a wiki interchange format (wif). In *Proceedings of the 1st Workshop on Semantic Wikis, Budva, Montenegro*, 2006.