

# Towards a Wiki Interchange Format (WIF)

## Opening Semantic Wiki Content and Metadata

Max Völkel<sup>1</sup> and Eyal Oren<sup>2</sup>

<sup>1</sup> Forschungszentrum Informatik, Karlsruhe, Germany,  
voelkel@fzi.de, <http://xam.de>

<sup>2</sup> DERI Galway, Ireland  
eyal.oren@deri.org, <http://eyaloren.org>

**Abstract.** Wikis are increasingly being used in world-wide, intranet and even in personal settings. Unfortunately, current wikis are data islands: people can read and edit them, but machines can only send around text strings without structure. Wiki migration, publishing from one wiki to another one and free choice of syntax hold back broader wiki usage.

We define a wiki interchange format (WIF) that allows data exchange between wikis and between related tools. Different from other approaches, we also tackle page content and semantic annotations. The linking from formal annotations to parts of a structured text is analysed and described.

## 1 Introduction

Wikis are increasingly being used in world-wide, intranet and even in personal settings. There are over 130 wiki engines listed in the *C2 Wiki*<sup>3</sup>. Google statistics reveal that *MediaWiki*<sup>4</sup> and *TWiki* are the most popular engines, with rising tendency. *MediaWiki* has  $314 \times 10^9$  hits, probably influenced by its usage on Wikipedia, one of the top 20 most used websites<sup>5</sup> in the world. *TWiki*<sup>6</sup> with  $30 \times 10^9$  hits is mostly run in company intranets.

Apart from these two specialised wiki engines, the many other wiki engines have relatively similar popularity<sup>7</sup>: PukiWiki (8,7 Mio), TikiWiki (7,1 Mio), MoinMoin (6 Mio), Atlassian Confluence (4,2 Mio, commercial), PhpWiki (4 Mio), PmWiki (3,9 Mio), Xwiki (3,3 Mio), JspWiki (1,8 Mio), SnipSnap (1,8 Mio), JotSpot (1,5 Mio), UseMod (0,7 Mio), and SocialText (0,7 Mio, commercial).

Unfortunately, the increasing usage of wikis leads to new problems, as many people have to use multiple wikis, e.g. a personal (local) wiki, the company

---

<sup>3</sup> <http://c2.com/cgi/wiki?WikiEngines>.

<sup>4</sup> <http://www.mediawiki.org>.

<sup>5</sup> according to Alexa.com, April 2006

<sup>6</sup> <http://www.twiki.org>.

<sup>7</sup> Based on Google queries since 2004 up to 28.03.2006. Google hit count can only be a rough indicator of wiki engine popularity

intranet wiki, an external collaboration wiki and the Wikipedia for background knowledge. But there is no interoperability between these wikis: despite their open-ness, current wikis are data islands.

Ideally, one could export some or all pages from one wiki and import them into another wiki, which uses a different wiki syntax and offers a different set of features. But the diversity of different wiki syntaxes makes exchanging structured content quite expensive: a converter has to be written for each wiki that is to be connected with other systems.<sup>8</sup>

Instead of writing conversion scripts between each pair of available wiki engines we propose the use of a wiki interchange format. This reduces the implementation costs roughly from  $n^2$  to  $n$  for  $n$  different wiki engines. An interchange format would also allow the independent creation of multiple wiki syntaxes and user interfaces.

### 1.1 Structure of this paper

First we analyse requirements (see Sec. 2). In Sec. 3, we elaborate on what constitutes the wiki data model and which is the right layer of abstraction for an interchange format. We also discuss interaction with wikis on the web. In Sec. 4, we make a proposal for a wiki interchange format (WIF) and a wiki archive format (WAF). We present the concept of a Wiki Mediation Server to handle runtime aspects of wiki migration. We report on implementation and experiences in Sec. 5 and review some related work in Sec. 6. Finally, in Sec. 7 we conclude and present future work.

## 2 Scenarios and Requirements

The following scenarios are currently not supported by existing wiki engines and would profit from a wiki interchange format (WIF):

**Exchanging wiki content.** Currently, the only way to copy content from one wiki to another one is through copy-and-paste and manual reformatting of the text. This is painful and unnecessary, since most wikis offer similar content formatting and structuring abilities, namely those found in HTML (as most wikis render their content as HTML). The rising popularity of personal wikis adds even more weight to the need of exchanging content between wikis.

**Wiki migration.** Sometimes, the complete content of one wiki should be migrated to another wiki, e.g. because another wiki engine should be used. This implies migrating all pages. Ideally, user accounts and page histories would also be migrated.

**Wiki syntax united.** Existing wikis use various different wiki syntaxes. Ideally, one could use the same favoured syntax on all wikis.

---

<sup>8</sup> Although some systems provide an HTML-import (e.g. JSPWiki [9] and TWiki), this does not fully solve the problem, as most other wikis can only export pages including navigational aids.

**Wiki archiving.** Creating a browse-able off-line version of a wikis content, e. g. to read on the train or as a backup.

**Wiki synchronising.** One of the big advantages of a wiki is the easy linking ability: users just have to remember the page title to create a link. External links require much more effort. Persons participating in multiple wikis have to remember many logins, navigation schemes and wiki syntaxes. Ideally, one could change a page in wiki *a* and have it automatically changed in wiki *b* as well.

From these scenarios and additional thinking, we can derive a number of requirements for a WIF:

**Round-tripping:** Full data round-tripping is the overall goal. Ideally, one could export all content of a wiki site into WIF and import it back again into another wiki installation without loss of data or structure. E. g.: If we have two wikis *a* and *b*, where *a* has a very rich and *b* a very little page structure model, the transition  $a \rightarrow b$  (i) would be useful for users of wiki *b*, even if a lot of structure within the pages would be lost due to the restricted page data model of wiki *b*. The mapping  $b \rightarrow a$  (ii) would also in general be beneficial for the users of wiki *a*. The problems arise with mappings like  $a \rightarrow b \rightarrow a$  (iii). Now the users of wiki *a* would be faced with a loss of structure which they used to have before the export and re-import of data. This is a problem that *cannot* be solved in general, due to the lower expressivity of wiki *b*. As the mappings (i) and (ii) are useful in practice for wiki migration, we have to find a format that is expressive enough.

**Easy to implement:** as we need a way to obtain the WIF for each different wiki engine, it is important to keep development costs low.

**Renderable:** WIF files should be renderable in standard browsers.

**Completeness:** If a wiki lists e. g. all pages in the category “employee” using some kind of plugin, a WIF should export this information in an appropriate format. Wikis with a similar powerful plugin system would profit from an interchange format that keeps a reference to the plugin used. Less capable wikis, however, would need the content generated by the plugin, as they do not have the power to generate the content themselves. Users migrating from a wiki with a particular feature to another wiki engine would rather profit from having dynamic content materialised into static content than not having access to that content at all.

**Compactness:** The single-page-WIF needs to encode all structural information of a wiki, e. g. nested lists, headlines, tables, nested paragraphs, emphasised or strongly emphasised words. But it does not need to encode font size or font color.

**Ease of Use:** It should not only be easy to generate single-page-WIF, it should also be easy to work with it.

## 2.1 Experiences in Wiki Migration

Recently, in one of the authors’ research groups, the intranet was migrated from SnipSnap wiki to MediaWiki – to use features of the Semantic MediaWiki ex-

tension [19]. As WIF was not ready at that time, one person manually converted the contents. As he migrated a wiki before, the basic task was not new to him. He worked on the stored text files and used a 213 lines long bash script, which in turn called 102 lines of *sed* and 29 and 34 lines of *awk* [4].

Reported problems where the unusual SnipSnap syntax - opening and closing tags are the same for plugins. Encoding was also an issue. Comments were migrated to MediaWiki discussion page sections. Finally, a generated XML file is imported using a MediaWiki XML import feature.

The approach took about 20 hours of work, would require significant refactoring before applicable to another target or source wiki engine, depends on having admin rights on the source and target wiki server and relies on specifics of the wiki engines, e. g. the XML import of MediaWiki.

### 3 Analysis and Discussion

In this section we argue that an interchange format has to exchange data on the level of the wiki data model and not on the wiki syntax level. Additionally, we show which parts of the wiki data model are relevant for a WIF.

One could try to standardise the wiki syntax, as many people suggested on diverse wiki and web pages [1]. A standard markup would indeed be of great benefit for novice users and data migration. But in reality, existing wikis are unlikely to change the established syntax rules. For new wiki engines, the MediaWiki syntax should be considered, as it is likely to be the most widely known syntax. Some new Semantic Wikis such as IkeWiki adopt this approach. On the other hand, innovation in wiki engines and wiki syntaxes is still high: new wiki engines and new wiki features, such as semantic annotations and plugins, need syntax extensions as well.

Therefore, standardising wiki syntax is not a feasible solution; instead we propose to define a standard format for the data model.

As we want to exchange the data structures of a wiki page, we abstract away from the ways these structures were created. E. g. a bulleted list is a structural concept, worth exporting, while the syntax that was used to create this structure (star or minus sign at the beginning of a line) is of less interest for other wikis or tools.

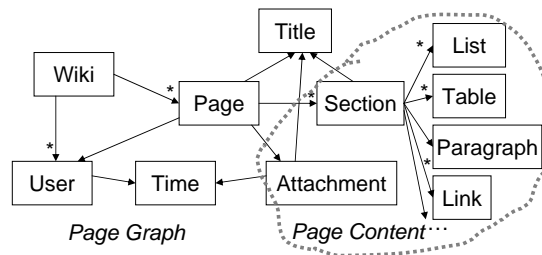
Unfortunately, most wiki engines have no formal data model published. Their data models are defined only by their implementation of wiki syntax and rendering. In order to obtain a *common* wiki data format, one would have to

- formalise existing wiki data models including page graph, page metadata and the structured page content, and
- identify the union of all formalised data models.

We looked into the data models of SnipSnap (used as the intranet of one of the authors' institute), MediaWiki (popular, see Sec. 1) and JspWiki (used by one of the authors as a personal desktop wiki).

### 3.1 Wiki data model

What is in a wiki? Several definitions are possible, we go for a definition from the end-user point of view. The end-user has two ways of interacting with wiki content. She can either *browse* (read-only) the hypertext version or *create and change* content of pages using wiki syntax. It is *the content written by users*, that has to be migrated to other wiki engines. Some important wiki features such as “backlinks” are inferrable from the user-entered content. Arguably, usage data such as “10 most accessed pages” or the “recently changed pages” contribute considerably to the wiki usability. This data cannot be manipulated directly by users through the web interface. So we have to ask ourselves:



**Table 1.** A high-level view on the wiki data model

What constitutes the data model of a wiki? The content of a wiki is defined by the page graph including page metadata and the actual content of pages. Figure 1 shows some of the most common elements of a wiki data model. Existing exchange formats tackle the page graph part (as XML dialects), but often simply embed the page content part as wiki syntax (e. g. MediaWiki, SnipSnap). This is maybe due the fact that most wiki engines have an internal representation of the page graph but not necessarily a detailed representation of a page given in wiki text. In fact, pages are often rendered by wikis using a cascade of text search and replace commands using carefully crafted regular expressions. To sum up, we distinguish three levels of data for wiki content:

**A wiki page** consisting of:

**Content structure:** e. g. headlines, nested lists, tables, or other elements that state a visual relation between content items.

**Content style:** e. g. font size, font color, bold, italic or other visual rendering of content items.

**Content semantics:** e. g. user authored content, backlinks, lists generated by a plugin or macro, embedding of other wiki pages, or a reference to a variable that display the number of pages in a wiki. The content semantics are invisible to e. g. an HTML processor.

**Metadata about a wiki page.** We have two kinds of metadata (at least in a Semantic Wiki context):

**Explicit metadata** as stated by semantic annotations or as defined in special user interface elements (i. e. access rights).

**Application metadata** such as last editor, creator of a page, previous version. This metadata cannot be changed directly by the user, only indirectly through application usage.

**Global wiki data** such as user accounts.

### 3.2 Semantic Wiki data model

Semantic Wikis such as SemperWiki [15, 14] add the notion of formal annotations to wikis, and we therefore extend the data model with annotations. We model an annotation consisting of three parts [16]:

1. A real world artefact that is annotated, such as a person, a book, or a web resource.
2. A formal identifier for the real-world artefact, such as an ISBN number or a URI [2].

The link from identifier to real-world artefact is outside the expressiveness of formal systems. In RDF, we face an additional problem: URIs are used both as concept identifiers and as locators (URL) of resources on the WWW. Given the URI of a web resource, what is the URI of the concept described on that web page?

In WIF, we solve the so-called “URI crisis” by using *mirror-URIs* [16]: for each web-locatable URI we construct a non-locatable URI by prefixing it with the URN-scheme ‘concept’ and URL-encoding characters as necessary. Such a mirror URI describes “the primary concept mentioned in the human-readable representation of the resource *a*”. E. g. `http://w3.org` represents the web page of the W3 consortium, while `urn:concept:http://w3.org` could denote the consortium itself. The exact meaning of the URN is up to a social process outside the scope of this paper. Nevertheless, the distinction between locatable web resources and concepts is crucial.

3. Formal statements about the real-world artefact, using the formal identifier as a placeholder. We assume formal annotations to be represented as RDF [12]. Note that the formal statements can also include information about provenance or scope of the annotation.

## 4 Design

In this section we describe the design of the wiki interchange format (WIF) for a single page and the wiki archive format (WAF) for a set of wiki pages.

### 4.1 WIF – A single-page wiki interchange format

We map wiki pages to directories on a storage medium or in a zip file. Each folder contains:

**index.html** - the unchanged html file corresponding to the wiki. Includes all navigation buttons and forms. This view helps to identify content visually. CSS files should be included. This file can be obtained with simple HTTP-GET tools, such as `WGET`.

**wiki.txt** - the source code in wiki syntax. This is a simple to implement fallback, if other steps in the conversion process produced wrong or suspicious output.

**wif.xhtml** - the wiki interchange format. In order to fulfill requirement “Renderable” (c.f. Sec. 2, WIF should re-use elements of HTML or XHTML. We decided to re-use XHTML elements, which are both open for machine processing, due to its XML nature as well as human viewable, using a standard browser. It is no problem to use elements not defined in XHTML: Browsers are requested to ignore unknown elements<sup>9</sup>.

WIF should be valid XML. This allows to use XSLT stylesheets to convert WIF back into a wiki syntax of choice. XSLT stylesheets can be run on all platforms, many programming languages, and even in some browsers.

The number of WIF-elements should be small, in order to facilitate further processing (req. “ease of use”). A study by Google shows<sup>10</sup>, that most HTML pages contain an average of only 19 different elements.

**index.rdf** - Annotations as RDF files. Meta-data, such as the distinction between wiki-link or external link is carried by using special XHTML attributes and values. This approach is inspired by Microformats<sup>11</sup>. More complex or user-given page meta-data should be stored in a linked RDF file.

In order to round-trip wiki specific features such as templates and macros, we represent them as annotations. The basic wiki page is exported as rendered. This fulfills requirement “Completeness” and ensures that another (even less capable) wiki, will show the page as the user knows it.

For macros and templates, annotations carry the additional hint that and how this part of the page was generated. Simple tools can only process the page as is, while smarter tools can exploit the knowledge of the annotations and e.g. update pages when a template is changed.

**Attachments** - Like emails, wiki pages can have arbitrary files attached to them. In order to store such files, we simply store them as files and link them from the WIF page. File creation date and other file related metadata can be encoded in the native file attributes.

Now we take a closer look at `wif.xhtml`. Which wiki structures are most important? We can distinguish three levels of formatting:

**Linking** is probably the most important element of a wiki. Wikis distinguish links to other wiki pages in the same wiki (wiki-links), links to pages in other wikis (interwiki-links) and links to any other web resource (external links).

---

<sup>9</sup> but process the content, for details see <http://www.w3.org/TR/xhtml1/#uaconf>

<sup>10</sup> <http://code.google.com/webstats/index.html>.

<sup>11</sup> <http://www.microformats.org>.

**Layout (inline-level)** is used to highlight parts of a text item. Basically, only bold and italic text can be used. For bold, HTML offers `<strong>` or `<b>`, for italic `<em>` or `<i>`. In single-page-WIF, we use only `<strong>` and `<em>`, to simplify its usage (req. Ease of Use). CSS layout is already separated from XHTML markup and is simply ignored.

**Structure (block-level)** is used to relate textual items on a page. Structural tags are paragraphs, headlines, lists and tables. Additionally, pages can contain horizontal lines and pre-formatted sections.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>...wiki page title ... </title>
  </head>
  <body>
    ... page content ...
  </body>
</html>
```

**Table 2.** The basic WIF page

A WIF-page thus has the structure shown in Fig. 2. WIF-Pages should always be UTF-8-encoded, to simplify further processing. The doctype should be XHTML 1.1. The `title`-element should be present and contain the wiki page title. The header may contain arbitrary additional elements.

To sum up, element tags used by WIF-processors in the WIF page body are: `a`, `dd`, `dl`, `dt`, `em`, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`, `hr`, `li`, `ol`, `pre`, `strong`, `table`, `td`, `tr` and `ul`. Other elements in the body of a WIF page may be ignored. The WIF page has to be a valid XHTML document, i.e. the usual XHTML element nesting rules apply.

We reserve some attributes and values to encode special wiki semantics: In links (`<a>`), we use the attribute `class` to specify the nature of a link: `wiki`, `interwiki` or `external`. This approach is inspired by microformats ideas<sup>12</sup>, and keeps WIF-pages valid XHTML and makes rendering in a browser easy. Note that an HTML-element may have multiple space-separated classes. As the title of a wiki page is often encoded, in order to make it a valid URL or filename, each link to an internal or external wiki page should also have a `title`-attribute with the wiki page title.

<sup>12</sup> <http://www.microformats.org>.

## 4.2 Linking pages with annotations

In order to keep the link between wiki pages (represented as XHTML) and their annotations (represented as RDF), we face a number of options. The problem is in principle the same as describing elements of one or more XML documents with RDF. Keeping the link is important, e. g. to record from which parts of an XHTML page an RDF statement was derived from.

**Embed XHTML elements in RDF:** The idea is to mimic the XML tag relations with RDF predicates, similar to the ideas in [11, 5]. Thus we would end up having e. g. a relation `:hasHead` which corresponds to the `<head>` tag. The content of XHTML tags would be stored as *individual plain RDF literals*. Thus granularity of content would be exactly at the level of tags, which is not sufficient.

**Embed RDF in XHTML:** A W3C proposal dubbed “RDF/A” specifies how to encode (even arbitrary) RDF as XHTML *attributes* (hence the “A” in the name). GRDDL<sup>13</sup> can extract the RDF back out of the XHTML file. RDF/A files are not valid XHTML files, as they introduce some changes to the document structure. Additionally, current RDF infrastructure cannot operate on RDF/A files.

**Embed Page:** One could embed the whole XML file as *one big XML literal* into an RDF graph. Then we lose interoperability with current XML infrastructure, e. g. a browser cannot even show the embedded pages.

**Link RDF file:** We leave the XHTML content as a separate file, and point to an RDF file from the `<head>`-element. In this RDF file, we can reference the XHTML page using the local filename (e. g. `HowToPrint.html`). Referencing parts of the XML file could be achieved by appending an XPointer [6] to the (relative) URI representing the XML document, e. g. `wif.xhtml#xpointer(/html/head/title)` could denote the title of the WIF page stored as `wif.xhtml`. This approach is used by semantic web annotation tools such as CREAM [7] and Annotea [10]. Even before XPointer was specified, people used fragment identifiers to denote parts of documents [8]. The relation between the URI representing the XML document fragment and the XML document must be stated explicitly, as current RDF infrastructure does not handle XPointer expressions, e. g. by a triple like adding `wif.xhtml#xpointer(/html/head/title) wif:partOf wif.xhtml`. With more formal definitions of the ‘concept’ namespace ID such statements could be inferred automatically. This approach is also used in CREAM [7] and Annotea [10].

However, there is a conceptual drawback: The web architecture and RDF speak about *resources*. An XML document is not a resource, but a *representation of a resource*. URIs are defined as *resource* identifiers. The representation returned by a server depends not only on the URI but also e. g. on the accept-header of the HTTP request. The semantics of the ‘concept’ namespace ID are thus defined explicitly only for XML content types. The XPointer specification uses the same approach.

<sup>13</sup> <http://www.w3.org/2004/01/rdxh/spec>.

For binary files, we propose a pragmatic approach: Each file called `filename.ext` can have an accompanying file `filename.rdf` which stores the meta-data of that file. Both files are included in the wiki archive format, which is described in the next section.

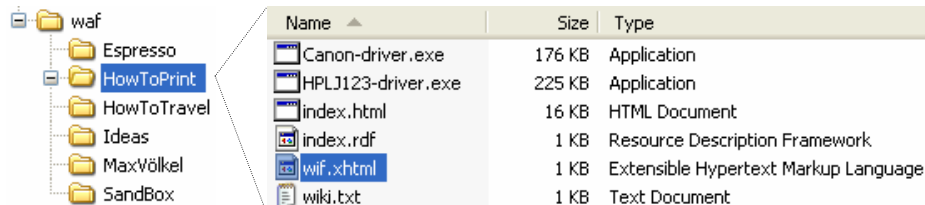


Fig. 1. WAF Example

### 4.3 WAF – The wiki archive format

Two use cases need a complete WAF, as opposed to a single-page-WIF:

**Wiki migration.** Here we have to handle the requirement to move a set of wiki pages at once. The easiest way to move multiple wiki pages across the network is probably to re-use an archive format such as the zip format. The same approach is taken in the Java world to handle a set of Java class files as a .jar file, which is a zip archive with a special structure. Another popular example of zip file usage is the open office format <sup>14</sup>, which stores a set of XML files that together constitute one document.

**Wiki archiving.** Using a zip file with subdirectories for each wiki page has the additional benefit that, if done right, this archive can be extracted and viewed off-line, using a standard browser. Hierarchical namespaces can be modeled as subfolders.

We propose to use a zip archive consisting a set of WIF files. For Semantic Wikis, we also include RDF files, linked from XHTML files. For pragmatic reasons, we decided not to include different versions of a wiki page in the interchange format. WIF files should be legal XHTML files and the links between the pages should point to each other (e.g. a wiki page *a* linking to a page called “HowToPrint” would contain the snippet ... `<a href="HowToPrint.html" title="HowToPrint" class="wiki"> HowToPrint </a>` ... All WIF files should have a file extension of “.html”. RDF files can have any extension (besides .html) as and should be linked from the XHTML files. Background knowledge, not related to a particular page should be stored as `index.rdf` in the root folder of the WAF zip file. An example of a WAF archive with six exported wiki pages is shown in 1.

<sup>14</sup> <http://www.oasis-open.org/committees/office/>.

#### 4.4 Wiki Mediation Server

At runtime, we need a component that provides translation from one wiki into WIF and from WIF back into wiki syntax. For migration, the component should also interact with wikis through their web interfaces, simulating a human editor. This idea is similar to WikiGateway [17], but WikiGateway does not address the problem of page structure translation. In order to provide WIF-translation services also for other tools, we use a service oriented architecture, as shown in Figure 2. This architecture allows a user independent of the wiki engines *source* and *target* to migrate wiki content. The functions offered by the wiki mediation server are:

**GetPage(*t*)** retrieves a WIF representation of a wiki page *p*, given its title *t*.

**PutPage(*p*,*t*)** converts a page *p* given in WIF into a wiki syntax of choice and stores it under a given page title *t*.

**GetRecentChanges(*u*)** gives a list of pages (with URLs) that have been changed after a given point in time, including a time stamp indicating the last change.

**GetIndex()** returns a list of all pages stored in the wiki. The list should contain URLs.

We show now how this design solves the scenarios given in Sec. 2 and then describe the design of the specific functions.

**Migrating wiki content:** In order to migrate a page with title *t* from wiki *a* to wiki *b*, we call `migrate(t,a,b)` which calls in turn `x = a.getPage(p)` and then `b.putPage(x)`.

**Wiki synchronising.** In order to synchronise the wiki *a* with wiki *b*, the wiki mediation server has regularly to invoke `a.GetRecentChanges`, for all pages that have been changed after the last invocation of this function. Then, for each page *p* with title *t* that has been changed, we call `migrate(t,a,b)`.

**Wiki syntax united.** In order to use an arbitrary wiki syntax *s* for a wiki *a*, we propose to use a proxy-wiki, which works as follows. For rendering a page with title *t*, we return `a.getPage(t)`. When a user wants to edit a page, we call `a.getPage(t)`, and convert the obtained WIF into syntax *c*. Upon page save, we convert the wiki text in syntax *c* back to WIF and put the result *r* using `a.putPage(r)` back into wiki *a*.

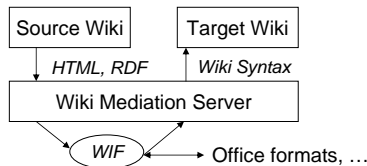


Fig. 2. Software Architecture for Remote Wiki Migration

## 5 Implementation and Experiences

A proof-of-concept implementation has been developed in Java made available as open source (LGPL) at <http://wif.ontoware.org>. Currently, it only exports a SnipSnap wiki – accessed via the web interface – into WAF. Support for JSPWiki import is being added. Output rendering for MediaWiki is available.

How do we obtain the data from a wiki? As each wiki has some kind of data persistence, the wiki administrator could access a wikis content on this layer. But each wiki has a different kind of persistence (e.g. text files, data bases, RDF), so writing adapters would require very different skills.

For open-source wiki engines, one could add a new export function to the code base. This would require to handle a large number of programming languages.

In order to migrate from abandoned wiki engines which are no longer maintained or where the user who wishes to migrate has not the necessary admin rights, we need another solution.

As almost every wiki has an HTML output, one could also try to get the content from there. Simulating a browser that clicks on “Edit this page” and other wiki core functions, one could get access to the same set of data as a user.

Current Wiki-APIs (XML-RPC based ones can be found e.g. in JspWiki<sup>15</sup>, Atlassian Confluence<sup>16</sup>, XWiki<sup>17</sup>) stop at the page graph level. A page can either be retrieved as raw text (wiki syntax) or rendered as HTML. The HTML version is often enriched by additionally inferred information, such as backlinks. Such information is harder to extract from HTML than from wiki syntax, e.g. the difference between wikilinks and external links can only be computed by comparing the base URL of a page with the link target. Nevertheless, Wiki engines use quite different syntaxes, but (almost) all wikis emit HTML in the user interface. Thus starting with the HTML syntax significantly eases development.

Semantic wikis offer the same ways of obtaining the data. Usually semantic data is made accessible via a link from the HTML version. Note that current semantic wikis do not retain the link which statement was generated from which part of the page.

As less then 1% of all web pages are valid (X)HTML [13], in the sense of passing the W3C validator<sup>18</sup>, we cannot expect wikis to emit only valid HTML. Fortunately, there are a number of software components available, that mimic the parser behaviour in browsers to transform ill-formed HTML into well-formed (X)HTML. As analysed in [18], the best performing component is CyberNEKO<sup>19</sup>. CyberNEKO transforms any ill-formed HTML input into well-formed XML, add missing opening and closing tags as necessary.

Individual pages are read with the *Jakarta Commons HttpClient* and post-processed with custom XSL transformations into WIF. HTTP Basic Authenti-

---

<sup>15</sup> <http://www.jspwiki.org>.

<sup>16</sup> <http://confluence.atlassian.com>.

<sup>17</sup> <http://www.xwiki.org>

<sup>18</sup> <http://validator.w3.org/>.

<sup>19</sup> <http://people.apache.org/~andyc/neko/doc/html/>.

cation is used to get into protected pages, login data has to be supplied by the user as URL parameters.

Although we lose some information (such as slight syntax variations), we can obtain most information even from the HTML data. We simply compare the `<base href>` of the page with the link targets. If the query string or last path segment (for `mod_rewrite`) matches the target, it is a link to the wiki. As this matching is done in the stylesheet, more complicated transformations are possible. It is e. g. possible to detect Wikipedias Inter-language-Links and mark them up as such in WIF, e. g. by adding a CSS class for the target language to the link class. We ignore linked CSS stylesheets and some presentational tags, in order to get a more concise WIF, fulfilling requirement “Compactness”.

The page index is read from the index page, which all wiki engines provide, and post-processed with XPath [3] expressions to get the actual page names.

Internally, the mediation server relies on Jetty<sup>20</sup>, as an embedded web server, and JRest<sup>21</sup>, as an automatic mapping from Java objects to RESTful servlets.

A demo server is currently set up, serving a WAF-archive for any SnipSnap wiki. A login is simulated in order to obtain the text-files in wiki syntax.

## 5.1 Experiences

No formal evaluation has been done. Instead, we review what we achieved.

Using WIF can dramatically lower the costs required to migrate wiki content. As in most integration problems using an intermediate format (WIF) reduces the number of translators needed from  $n^2$  to  $2n$ . The process of writing the translators from HTML to WIF can partially be automated (see [18]). As WIF consists of less elements than full XHTML, XSL stylesheets to convert WIF back into wiki syntax are easier to write.

Future wikis can use the Wiki Mediation Server to offer real-time wiki syntax of choice. To do this, they have to act as a proxy. When a user edits a page, the user entered text in syntax  $a$  is run through the parser of a wiki  $a$ , resulting in HTML. That HTML is converted first to WIF and then to wiki syntax  $b$ . This syntax  $b$  is then stored in wiki  $b$ .

First tests show that the transformation from SnipSnap’s HTML to clean XHTML and then via custom XSLT to WIF is indeed possible. Another XSLT was written to convert WIF to MediaWiki syntax.

Mapping wiki pages to folders leaves much freedom. The format is extensible, i. e. more files can be stored in the same directory, e. g. different versions. Complete static web sites can be generated from a WAF file, in fact, a WAF file *is* a static web site, in one zip file. This makes WAF also an ideal wiki backup format, with no need to keep the original server infrastructure alive. Alas some features (e. g. full-text search) get lost.

The problem with this approach is the reference management. To create a valid reference between static wiki pages, one has to link to `/PageName/index.html`, instead of just the page name.

<sup>20</sup> <http://www.mortbay.org>.

<sup>21</sup> <http://jrest.ontoware.org>.

## 6 Related Work

There are several proposals about wiki standardisation. The WikiModel<sup>22</sup>, addresses an in-memory model for wikis in Java, using a particular semantic model (pages with sections, allowing page inclusion). WikiModel includes a clever wiki syntax and parser design.

WikiWyg<sup>23</sup> is an approach to offer in-browser WYSIWYG editing of wiki content. To do this, WikiWyg uses Javascript to convert from DOM to wiki syntax. Oddmuse<sup>24</sup> has an integration with Emacs<sup>25</sup> which allows users to read and update pages with Emacs, a desktop-based text editor.

Similar to WikiGateway, an extension to JSPWiki written by Janne Jalkanen [9], allows to get and put pages. DavWiki exposes the wiki pages as text files in a WebDAV directory. Note that the syntax conversion problem is left untackled.

SweetWiki<sup>26</sup> uses RDF/A to encode semantic annotations in HTML pages. Annotations can only be on the page level, not allowing annotating parts of a page. IkeWiki offers a custom XML-based export format. It is similar in spirit to WIF, as it exports only the core structures. But different from WIF, IkeWikis export is not suitable for viewing in a browser.

## 7 Conclusion and Outlook

We have shown how a wiki interchange format should be designed. Although our model is still in a prototype stage, we have carved the path for the development of a true Wiki Interchange Format. A reference implementation is available.

A standardisation effort needs consensus. We hope to continue the discussion that started at the WikiSym 2005 as *Wiki Standardisation Request 3* on the wiki page [http://www.wikisym.org/wiki/index.php/WSR\\_3](http://www.wikisym.org/wiki/index.php/WSR_3). The next steps are a more precise requirements gathering and and RFC-style document.

*Acknowledgments:* This material is based upon works supported by the Science Foundation Ireland under Grants No. SFI/02/CE1/I131 and SFI/04/BR/CS0694. This research was partially supported by the European Commission under contract FP6-507482 (Knowledge Web) and FP6-027705 (Nepomuk). The expressed content is the view of the authors but not necessarily the view of the Knowledge Web Network of Excellence as a whole.

We thank Sebastian Gerke and Werner Thiemann for their help in creating the Wiki Exchange Format, and Dirk Riehle for initiating the BOF at the WikiSym 2005, where all this started. Thanks also to Mikhail Kotelnikov and Malte Kiesel for lengthy and fruitful discussions, and special thanks to Dirk Achenbach who went the painful way of manual wiki migration.

<sup>22</sup> <http://wikimodel.sourceforge.net/>.

<sup>23</sup> <http://www.wikiwyg.net>.

<sup>24</sup> <http://www.oddmuse.org>.

<sup>25</sup> <http://www.gnu.org/software/emacs/>.

<sup>26</sup> <http://wiki.ontoworld.org/wiki/SweetWiki>.

## References

1. M. Altheim. Inter wiki markup language (iwml), 03 2004.
2. T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (uri): Generic syntax. Rfc 3986, The Internet Society, Jan 2005.
3. J. Clark and S. DeRose. Xml path language (xpath) version 1.0. Technical report, W3C, Nov 1999.
4. D. Dougherty and A. Robbins. *sed & awk (2nd Edition)*. O'Reilly Media, Inc., March 1997.
5. M. Erdmann and R. Studer. How to structure and access xml documents with ontologies. *Data and Knowledge Engineering*, 2000.
6. P. Grosso, E. M. J. Marsh, and N. Walsh. Xpointer framework. W3c recommendation 25 march 2003, W3C, MAR 2003.
7. S. Handschuh and S. Staab. Cream - creating metadata for the semantic web. *Computer Networks*, 42:579–598, AUG 2003. Elsevier.
8. M. Hori, R. Mohan, H. Maruyama, and S. Singhal. Annotation of web content for transcoding. W3C Note, July 1999.
9. J. Jalkanen. Davwiki – the next step of wikircpinterfaces? In *Proceedings of Wikimania 2005 - The First International Wikimedia Conference*. Wikimedia Foundation, JUL 2005.
10. J. Kahan and M.-R. Koivunen. Annotea: an open RDF infrastructure for shared web annotations. In *Proceedings of the 10th International World Wide Web Conference*, pages 623–632, 2001.
11. M. Klein. *Interpreting XML via an RDF Schema*. IOS Press, Amsterdam, 2003.
12. G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. <http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>, February 2004.
13. M. L. Noga and M. Völkel. From web pages to web services with wal. In *NCWS 2003*, Växjö, Sweden, NOV 2003. Mathematical Modelling in Physics Engineering and Cognitive Science.
14. E. Oren. SemperWiki: a semantic personal wiki. In S. Decker, J. Park, D. Quan, and L. Sauermaun, editors, *The Semantic Desktop – Next Generation Information Management & Collaboration Infrastructure*, Galway, Ireland, 2005.
15. E. Oren, J. G. Breslin, and S. Decker. How semantics make better wikis. In *WWW (poster)*, 2006.
16. E. Oren, R. Delbru, K. Möller, M. Völkel, and S. Handschuh. Annotation and navigation in semantic wikis. In M. Völkel and S. Schaffert, editors, *Proceedings of the First Workshop on Semantic Wikis - From Wiki to Semantics at the ESWC 2006*, May 2006.
17. B. Shanks. Wikigateway: a library for interoperability and accelerated wiki development. In *WikiSym '05: Proceedings of the 2005 international symposium on Wikis*, pages 53–66, New York, NY, USA, 2005. ACM Press.
18. M. Völkel. Extraktion von XML aus HTML-seiten – das WYSIWYG-werkzeug d2c. Diplomarbeit, May 2003.
19. M. Völkel, M. Krötzsch, D. Vrandeic, H. Haller, and R. Studer. Semantic wikipedia. In *Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, May 23-26, 2006*, May 2006.