Towards a Wiki Interchange Format (WIF)

Opening Semantic Wiki Content and Metadata

Max Völkel[1] and Eyal Oren[2]

[1] Institute AIFB, Universität Karlsruhe, Germany
voelkel@aifb.uni-karlsruhe.de
[2] DERI Galway, Ireland
eyal.oren@deri.org

Abstract. Wikis tend to be used more and more in world-wide, intranet and increasingly even in personal settings. Current wikis are data islands. They are open for everyone to contribute, but closed for machines and automation.
In this paper we define a wiki interchange format (WIF) that allows data exchange between wikis and related tools. Different from other approaches, we also tackle the problem of page content and annotations. The linking from formal annotations to parts of a structured text is analysed and described.

1 Introduction

Wikis tend to be used more and more in world-wide, intranet and increasingly even in personal settings. We believe, the easy way of creating arbitrary structured and linked pages lead to wide-spread adoption of wikis.

Many wikis, many wiki syntaxes The increasing usage of wikis leads to new problems, as many people have to use multiple wikis, e. g. a company intranet wiki, a wiki for collaboration with an external partner and Wikipedia. Some persons additionally run a local wiki engine on their desktop.

The most popular wiki is Wikipedia, one of the top 30 most used websites[3] in the world. Wikipedia runs on MediaWiki[4] (314 Mrd), using MediaWiki syntax. Company intranets often run on TWiki[5] (30 Mrd). There are over 130 wiki engines listed in the C2 Wiki[6]. Google queries reveal that MediaWiki and TWiki are the most popular engines, with rising tendency. But despite the enormous growth of these two engines, many other wiki engines have rising Google hit counts as well[7]. The following wiki engines had high Google hit counts as

[3] according to Alexa

[4] http://www.mediawiki.org.

[5] http://www.twiki.org.

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

[7] Based on Google queries since 2004. Google hit count can only be a rough indicator of wiki engine popularity

of 28.03.2006 (sorted in descending order): 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), ZwiKi (0,9 Mio), UseMod (0,7 Mio), SocialText (0,7 Mio, commercial). 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.

Interoperability problem But despite their open-ness, current wikis are data islands. They are open for everyone to contribute, but closed for machines and automation. Of course, scripts that interact with a wiki over the web are easily written. But its hard for such scripts to do useful things with a page's *content*. Especially data integration and data migration suffer from an interoperability problem: It is not easily possible to export a page from one wiki and import it to another one. Ideally, one could export a wiki page from one wiki and import it into another wiki, using a different wiki syntax and offering a different set of features. Unfortunately, there exist many different wiki engines, all using a different syntax.

Reducing integration costs with an intermediate format 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. In particular we propose a *Semantic Wiki Exchange FormaT* (SWIFT) which abstracts away from the wiki syntax and HTML output. We argue that Semantic Web technologies are flexible and open enough for this task. Opening the data model of wiki engines would allow the independent creation of multiple wiki syntaxes and user interfaces. This paper tries to show how Semantic Web concepts can be beneficial for wiki development and explains some basic concepts.

Semantic Wikis Recently, wikis have been combined with Semantic Web technologies. Semantic Wikis allow users to annotate pages or parts of pages with formal metadata. A wiki page in a semantic wiki thus consists of the structured text, formal statements and the link from the formal statements to parts of the structured text. For data exchange, we face even more problems: We must export all three kinds of data – only this allows full round-tripping.

1.1 Structure of this paper

First we analyse requirements (see Sec. 2). In Sec. 3, we elaborate on what constitutes to 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 a hypothetical Wiki Mediation Server to handle runtiem aspect of wiki migration. We give a short evaluation inSec. 5 and review some relaated work in Sec. 6. Finally, in Section **??** we conclude on present future work.

2 Scenarios and Requirements

Now we briefly review scenarios, that demand for a wiki interchange format (WIF), as they are currently not supported by existing wiki engines:

Migrating wiki content. Currently, the only way to copy content from one wiki to another one is through copy & paste and careful manual reformatting of the text. This is especially painful, as most wikis offer the same *content formatting and structuring* abilities anyways, namely those found in HTML. This should be no surprise, as most wikis render their content as HTML anyways. 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. But for pragmatic reasons, we leave this out of the first version of WIF.

Wiki syntax united. One of the most often expressed problems[8] are the different wiki syntaxes. Ideally, one could abstract away the different syntaxes and use the most favoured syntax on all wikis.

Wiki archiving. Creating a browsable off-line version of a wikis conent, e. g. to read on the train or as a backup is also desired by some wiki users.

Wiki synchronising. E. g. synchronising a local wiki with a public or shared wiki. 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 access controlled team wikis have to remember many logins, navigation schemes and wiki syntaxes. Ideally, on 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: Most obviously, full data round-tripping is the overall goal. Ideally, one could export a complete wiki into WIF and import it back again into another wiki engine (or other tool) without any loss of data. If we have two wikis a and b, where a has a very rich and b a very poor model, the transition $a \to b$ (i) would be useful for users of wiki b, even if a lot of structure within the data would be lost due to the restricted data model of wiki b. The mapping $b \to a$ (ii) would also in general be beneficial for the users of wiki a. The problems arise with mappings like $a \to b \to a$ (iii). Now the users of wiki a would be faced with a loss in data structure richness which they used to have before the export and re-import of data. This is a problem that *cannot* be solved in general. As the mappings (i) and (ii) are useful in practice for wiki migration, we have to find a format that is expressive enough.

[8] personal communication with the authors

Easy to implement: as we need a way to obtain the WIF for each different wiki engine, it's 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 have not 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.

3 Analysis and Discussion

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

Interchange of wiki syntax or wiki structures? One could try to standardise the wiki syntax, as many people suggested on diverse wiki and web pages [1,2] . A standard markup would indeed be of great benefit for novice users and data migration (assuming data storage would also be standardised). 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[9].

As standardising wiki syntax is not a feasible solution for the wiki migration problem, we propose to define a standard format for the data model. We want to exchange the data structures of a wiki page, abstracting away form the syntax used to create these structures. 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. Additionally, we want to exchange all other relevant data of a wiki, besides the pages.

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

[9] Some new Semantic Wikis such as IkeWiki adopt this approach

– 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, MediaWiki and JspWiki.

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 form 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:

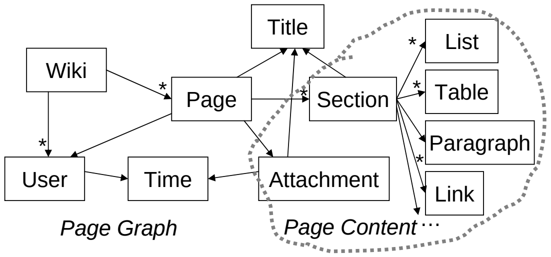

Fig. 1. A high-level view on the wiki data model

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. In some Semantic Wikis, such as SemWiki [5], users can make semantics annotations, that are not bound to a particular page. Rather, the annotations belong to the wiki as a whole.

Extending the data model with annotations. Semantic Wikis add the notion of formal annotations to wikis. We model an annotation consisting of three parts:

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.
3. Formal statements about the real-world artefact, using the formal identifier as a placeholder. Formal annotations are typically represented as RDF using RDFS or OWL as the representational ontology..

Note that the formal statements can also include information about provenance or scope of the annotation. The most difficult part for formal annotations is an agreement about (2): Which formal identifier stands for which real-world artefact? In RDF, these problems get even worse by the fact that URIs are used for formal identifers which are a superset of URLs, which are locations of real-world web resources, e. g. an HTML page. This problem is known as the URI "crisis". Fortunately, there is an elegant, pragmatic solution, because web browsers do not transmit the fragment identifier part to the web server. As long as we limit ourselves to annotate complete web resources and not parts of them, appending a "#" to a web resource URL a leads to a unique URI a' suitable for use in RDF. This URI a' then describes "the primary concept mentioned in the human-readable representation of the resource a". Technically, it would be possible to have web resources with an address which contains a '#', but we can simply ignore those hypothetical cases.

How do we obtain the data from a wiki? We have two or three options:

- 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. 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.

- 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.
- 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[10], Atlassian Confluence[11], XWiki[12]) 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. Some 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. The basic motivation is this: Wiki engines use quite different syntaxes, but (almost) all wikis emit HTML in the user interface.

4 Design

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

4.1 WIF – A single-page wiki interchange format

WIF as XHTML subset. In order to fulfill requirement "Renderable" (c. f. Sec. 2, WIF should be a subset of HTML or XHTML. We decide to use an XHTML subset, which is both open for machine processing, due to its XML nature as well as human viewable, using a standard browser. The subset should be small, in order to facilitate further processing (req. "ease of use"). A study by Google shows [13], that most HTML pages contain an average of only 19 different elements.

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[14]. More complex or user-given page metadata should be stored in a linked RDF file.

[10] http://www.jspwiki.org.

[11] http://confluence.atlassian.com.

[12] http://www.xwiki.org

[13] http://code.google.com/webstats/index.html.

[14] http://www.microformats.org.

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 pages. File creation date and other file related metadata can be encoded in the native file attributes.

Extending WIF for full round-tripping. In order to round-trip subtleties such as templates and macros, we model 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 maybe even re-run the macro or template code.

Obtaining WIF from HTML. In order to be easy to implement, we process the HTML output of a wiki. Although we loose some information (such as unimportant 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's a link to the wiki. We ignore CSS stylesheets and some presentational tags, in order to become a more concise WIF, fulfilling requirement ”Compactness”.

Which wiki structures are most important? We can distinguish two levels of formatting:

Linking (¡a¿) 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).

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).

Structure (block-level) is used to relate textual items on a page. Structural tags are paragraphs (¡p¿), headlines (¡h1¿ - ¡h6¿) lists (¡ol¿, ¡ul¿, ¡li¿, ¡dl¿, ¡dt¿, ¡dd¿) and tables (¡table¿, ¡tr¿, ¡td¿). Additionally, pages can contain horizontal lines (¡hr¿) and pre-formatted sections (¡pre¿, ¡code¿).

A WIF-page thus has the structure shown in Fig. 2. Note: 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, ul pre, strong, table, td, th, tr` and `img`. 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.

```
<?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>
```

Fig. 2. The basic WIF page

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 [15], 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.

Linking pages with annotations In order to keep the link between wiki pages (represented as XHTML) and their annotations (represented as RDF), we discuss a number of options:

Atomize XHTML: The idea is to mimic the WIF page model with RDF predicates. Thus we would end up having a relation `:hasHead` which relates a node of `rdf:type :head` with a node of `rdf:Type :body`. Instead of using blank nodes for the nodes, we suggest to use random unique URIs, as described in [6]. The content of XHTML tags would be stored as RDF literals using this schema:

```
rnd://12345 rdf:value "The content of an paragraph"
```

But, although RDF literals are now addressable, annotations are only possible on this (sometimes too coarse) level of granularity.

Embed Page: One could embed the whole WIF (which is legel XHTML) as an XML literal into an RDF graph. Again, we make the literal referencable by stating

```
rnd://12345 rdf:value "<?xml ... </html>"^^XMLLiteral
```

Referencing parts of the xml literal could be achieved by appending an XPath [16] to the URI (rnd://12345) representing the XML Literal, e. g. leading to

[15] `http://www.microformats.org`.

[16] `http://www.w3.org/TR/xpath`.

the full URI `rnd://12345#/html/head/title` to denote the titel of the WIF page.
The problem with this approach lies in the non-standardised usage of Xpath expressions in fragment identifiers. Thus RDF query languages cannot operate on the fragments of the XMLLiteral.

Link RDF file: We leave the XHTML file as a separate page, but add a link to the ¡head¿-element, and let it point to an RDF file. In this RDF file, we can reference the XHTML page using the local filename (e. g. HowToPrint.html) and parts of it by appending URI-encoded XPath expressions to it (e. g. HowToPrint.html

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). Although GRDDL[17] can extract the RDF back out of the XHTML file, this methods seems rather cumbersome, especially for large amounts of RDF.

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

4.2 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. Another popular example of zip file usage is the open office format [18], which stores a set of XML files that together constitute one document.

Wiki archiving. Using a zip file of WIF files has the additional benefit that, if done right, this archive can be extracted and viewed off-line, using a standard browser.

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 snipped `... <a href="HowToPrint.html" title="HowToPrint">HowToPrint</a> ...` All WIF files should have a file extension of ”`.html`”. RDF files can have any extension (besides .html) as and

[17] `http://www.w3.org/2004/01/rdxh/spec`.

[18] `http://www.oasis-open.org/committees/office/`.

should be linked from the XHTMl files. Background knowledge, not related to a particular page should be stored as `index.rdf`.

For Semantic Wikis it might be more suitable to have the whole contents of WAF as a single RDF file. Unfortunately, this brings us back to the problem how to represent XHTML structures in RDF. The best approach seems to create a wiki structure ontology that mimics XHTML.

4.3 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 [3], but WikiGateway does not address the page structure translation. In order to provide WIF-translation services also for other tools, we use a service oriented architecture, as shown in Figure 3. 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 it's 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).

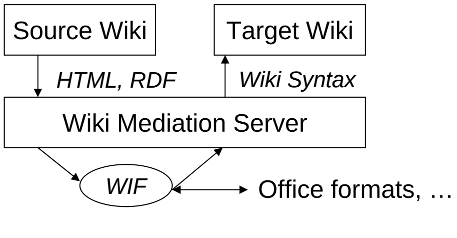

Fig. 3. Software Architecture for Remote Wiki Migration

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.

Converting from HTML to WIF an back to wiki syntax Two syntax translations are needed:

HTML to WIF. As indicated by requirement "Compactness", we choose to extract the WIF from the HTML output. As less than 1% of all web pages are valid (X)HTML [4], in the sense of passing the W3C validator[19], 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 [4], the best performing component is CyberNEKO [20]. CyberNEKO transforms any ill-formed HTML input into well-formed XML, add missing opening and closing tags as necessary. The resulting XML can be transformed with a custom Extensible Stylesheet Template (XSLT) into WIF.

WIF to Wiki Syntax. WIF should be a strict XHTML subset. 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.

5 Evaluation

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 [4]. 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 than via custom XSLT to WIF is indeed possible. Another XSLT was written to convert WIF to MediaWiki syntax.

[19] http://validator.w3.org/.

[20] http://people.apache.org/~andyc/neko/doc/html/.

6 Related Work

There are several proposals about wiki standardisation. The WikiModel[21], 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[22] 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.

SweetWiki[23] 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.

Future work will go into implementing a proof-of-concept for a Wiki Mediation Server. This paper is intended to bootstrap a discussion in the (Semantic) wiki community, in order to reach consensus about the requirements and design of a WIF. We hope to continue the discussion on the First Workshop on Semantic Wikis in June 2006 and on the web site http://www.wikisym.org/wiki/index.php/WSR_3.

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 and by the European Commission under the Nepomuk project FP6-027705. The authors would like to thank Sebastian Gerke and Werner Thiemann for their help in creating the Wiki Exchange Format. Thanks also to Dirk Riehle for initiating the BOF at the WikiSym 2005, where all this started.

References

1. M. Altheim. Inter wiki markup language (iwml), 03 2004. Cited in 3.
2. MeatballWiki. Wiki interchange format. Wiki, 08 2004. Cited in 3.
3. B. Shanks. Wikigateway: a library for interoperability and accelerated wiki development. In *WikiSym '05: Proceedings of the 2005 international symposium on Wikis*, pp. 53–66. ACM Press, New York, NY, USA, 2005. Cited in 4.3.

[21] http://wikimodel.sourceforge.net/.

[22] http://www.wikiwyg.net.

[23] http://wiki.ontoworld.org/wiki/SweetWiki.

4. M. Völkel. Extraktion von xml aus html-seiten – das wysiwyg-werkzeug d2c. Diplomarbeit, MAY 2003. Cited in 4.3 and 5.
5. M. Völkel. Semwiki - a restful distributed wiki architecture. In *Proceedings of the First International Symposium on Wikis*. San Diego, USA, OCT 2005. Cited in 3.
6. M. Völkel, *et al.* Semversion - versioning rdf and ontologies. Knowledge Web Deliverable 2.3.3.v2, University of Karlsruhe, JAN 2006. Cited in 4.1.