An Architecture to Support Intelligent User Interfaces for Wikis by Means of Natural Language Processing

Johannes Hoffart, Torsten Zesch, Iryna Gurevych
Ubiquitous Knowledge Processing Lab
Computer Science Department
Technische Universität Darmstadt, Hochschulstraße 10
D-64289 Darmstadt, Germany
www.ukp.tu-darmstadt.de

ABSTRACT

We present an architecture for integrating a set of Natural Language Processing (NLP) techniques with a wiki platform. This entails support for adding, organizing, and finding content in the wiki. We perform a comprehensive analysis of how NLP techniques can support the user interaction with the wiki, using an intelligent interface to provide suggestions. The architecture is designed to be deployed with any existing wiki platform, especially those used in corporate environments. We implemented a prototype integrating the NLP techniques keyphrase extraction and text segmentation, as well as an improved search engine. The prototype is integrated with two widely used wiki platforms: MediaWiki and TWiki.

Categories and Subject Descriptors

H.1.0 [**User/Machine Systems**]: Human factors; H.3.1 [**Content Analysis and Indexing**]: Linguistic processing; H.3.3 [**Information Search and Retrieval**]: Retrieval models, Search Process; H.3.5 [**On-line Information Services**]: Web-based services; H.5.2 [**User Interfaces**]: User-centered design; H.5.4 [**Hypertext/Hypermedia**]: Architectures, Navigation, User issues; H.5.3 [**Group and Organization Interfaces**]: Collaborative computing, Web-based interaction; I.2.7 [**Natural Language Processing**]: Text analysis

General Terms

Design, Human Factors

Keywords

Natural language processing, Wiki, User interaction, Content organization

WikiSym '09, October 25-27, 2009, Orlando, Florida, U.S.A.

1. INTRODUCTION

Wikis are a fairly new kind of content management system. In companies, wikis are often used to capture knowledge like brainstorming minutes or draft documents that would be lost when using centrally managed content management systems [1]. The benefit of using wikis for collaboration is that they provide a simple and fast way to add and edit content [19]. Majchrzak et al. [20] studied the behavior of corporate wiki users, and concluded that wikis are indeed sustainable in a corporate environment. However, some corporate wikis have been abandoned, as wikis tend to get too disorganized during growth. For example, Buffa [1] reports that the wiki at New York Times Digital, after a very promising beginning, grew to a mass of unmanageable pages with meaningless page titles and no usable link structure. Though very attractive at first, the open editing policy of wikis seems to encourage a wild growth of content, lacking a coherent structure which is necessary for finding the content that resides in the wiki.

The main problem that users report when using wikis is the disorganized content, that makes "navigation, orientation and search sometimes difficult" [1]. In wikis, search is very often used as a navigational aid, where the user knows the page to be retrieved in advance. If the search does not return a certain page the user knows to be there, the user's trust in the search is lost. Other studies revealed that problems caused by the lack of structure, like not finding content and also the unwitting recreation of existing content in knowledge management systems, are the source of high costs [7]. In wikis, this is even more problematic because of the aforementioned difficulties when searching for redundant content, and because wikis are not as well structured as classical content management systems. Nevertheless, wikis are recognized by companies to be a valuable tool to manage knowledge [20], as corporate knowledge is more easily shared and reused. Thus, it is important to find a way for improving the content organization in the wiki while still retaining its open and collaborative nature.

We propose a system to support wiki users with the tasks of adding, organizing, and finding content in a wiki by applying Natural Language Processing[1] (NLP) techniques. We analyze different types of user interactions corresponding to these tasks and identify NLP techniques that can aid the

[1]For a detailed description of Natural Language Processing, refer to [17]

user, e. g. by providing suggestions of where to add or how to organize content. In order to support these user interactions, we propose a general and extensible architecture to build an intelligent user interface for wikis using the identified NLP techniques. To be able to display suggestions to the user, the architecture needs to be coupled with the wiki interface itself. The direct integration into the interface is also necessary for giving the user a seamless experience, thus fostering usability. With the resulting system called "Wikulu"[2], we envision an interactive process where the user is involved at every step, supported by NLP techniques [14]. We do not aim at a kind of "black box" approach where the user presses a single button and is then confronted with a restructured wiki. This has two main reasons: (i) the current state of research in NLP is not advanced enough to support such content re-organization with perfect results, and (ii) users need to be able to assess the quality of the process. If changes are made in different parts of the wiki without a possibility to actually see what is going on, the acceptance of users would be low. Thus, we need an intelligent user interface that gives the user full control over every step, only suggesting possible and relevant actions.

The NLP techniques that can be employed to derive intelligent suggestions are diverse, including e. g. text segmentation [4, 16], keyphrase extraction [23], and calculating text similarity using semantic relatedness [13, 34]. The corresponding system architecture needs to be general enough to be able to integrate simple NLP techniques working on a single document as well as very complex ones requiring access to additional resources. Additionally, the architecture should work with arbitrary wiki platforms, as there are a lot of different platforms in use and our approach should not be restricted to a specific platform. The approach itself is actually not limited to wikis, it is general enough to be applied to any web-based document collection. Nevertheless, because of the increased difficulty of organizing content in wikis, the need for user support is more pressing there. Thus, we focus our research on wikis, and present a system architecture to integrate NLP techniques into wikis, keeping the architecture as independent from specific wiki platforms as possible.

The remainder of this paper is organized as follows. We analyze different types of user interaction and NLP techniques to support them in Section 2, followed by a discussion of the system architecture in Section 3. Section 4 takes a look at related work, and Section 5 concludes with a summary and an outlook.

2. THE WIKULU APPROACH

Wikulu focuses on helping the user to organize the wiki content by providing suggestions of where content could be added or how existing content can be re-organized. The task of re-organizing a wiki is already possible in current wikis, albeit very tedious, by moving pages or copying content manually. We aim at improving the user interaction in this process, so that it is easier for wiki users to organize the content.

[2]http://www.ukp.tu-darmstadt.de/projects/wikulu/

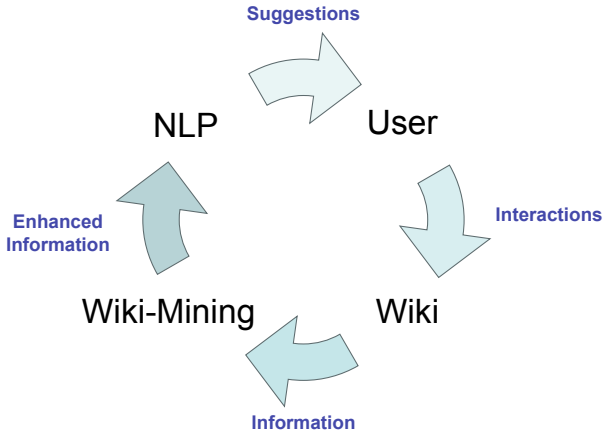


Figure 1: Intelligent User Interface Wiki Cycle

Figure 1 displays the basic steps of our approach, forming a cycle. The most important step is the interaction of the user with the wiki, which we try to improve by providing suggestions. To derive suggestions that support the user, we can make use of the wiki content that is already available. The wiki structure contains useful information, e. g. links or the revision history. The extraction of such information is called Wiki-Mining [21]. Wiki-mined information can then be utilized as additional input for the NLP algorithms, which in turn provide the suggestions. The better the wiki pages are structured, the more information can be extracted in the Wiki-Mining process, and subsequently used for the NLP algorithms. This will in turn lead to better suggestions for the user, thus closing the wiki cycle.

Helping to organize the wiki content is our major focus, but other types of interaction can also be improved. There are three major types of user interactions that arise when wikis are used: (i) adding content, (ii) organizing content, and (iii) finding content. Each type of interaction comes with specific challenges, and there are different ways to address them. Table 1 presents an overview of the interaction types which will be discussed next.

2.1 Adding Content

A major problem of collaborative content creation is the duplication of content in separate parts of the wiki. A study by Feldman and Sherman [7] reveals that "an enterprise employing 1,000 knowledge workers wastes $5 million per year because employees spend too much time duplicating information that already exists within the enterprise". Different users add the same or similar content to different parts of the wiki. This happens because most users are not aware of all the content in the wiki. Even if they search for duplicate content beforehand, they might not easily find it due to the vocabulary gap, which occurs when searching for a keyword that is not present in the content repository, although words with similar meaning or even synonyms are present.

To **reduce duplicate content**, Wikulu will search for the content the user is adding while the user is still typing. Related pages will be displayed and the user can more easily decide whether similar content is already available. By

Types of Interaction	Wikulu Support	NLP Techniques
Adding content	Detect duplicate content	Duplicate detection, Semantic relatedness
	Suggest points of insertion	Semantic relatedness, Text segmentation
Organizing content	Suggest links	Link detection, Word sense disambiguation
	Suggest tags	Keyphrase extraction, Tag prediction
	Suggest page split/merge	Text segmentation, Semantic relatedness
Finding content	Recall-oriented search	Semantic relatedness
	Show related pages while browsing	Semantic relatedness

Table 1: Possibilities to Enhance User Interaction in Wikis

checking the related pages, the user can decide whether the content is redundant or if the existing pages should be enhanced.

Wikulu will also **propose positions in existing documents** where the content could be inserted. Positions that are suitable for insertion can be found by computing the text similarity based on semantic relatedness measures [13, 34]. If a highly similar page is found, the page can subsequently be segmented into topics by means of text segmentation algorithms [4, 16]. The single segments can in turn be compared to the content that is to be inserted, identifying a possible position for insertion inside the page.

2.2 Organizing Content

If the user has added content to the wiki, Wikulu will offer **support to link** the newly added page to existing ones, thereby integrating it with the rest of the wiki. Pages that are not linked are hard to find, so this is a crucial task. Possible link targets will be displayed, and links can be added simply by confirming the suggestions. Recently, highly accurate methods to identify significant terms inside pages and link them to other pages in Wikipedia have been proposed [22, 24]. The approaches have also been tested in linking "real world" documents to Wikipedia, with similarly good results as when linking Wikipedia articles internally.

Tags are another way of organizing wikis. Originally, tags have gotten popular on Web 2.0 sites like del.icio.us[3] and Flickr[4], but have also been adopted in wikis, e. g. in SweetWiki [2] or TWiki. Tags add another dimension to the navigation in wikis. They can serve as a starting point for navigation, e. g. displayed as a tag cloud. Another way to use tags as a navigational facility is by presenting tags on an associated page as links. The user can then follow the linked tag to view a list of other pages associated with the tag. Another option to improve organization using tags is by presenting them as keywords at the beginning of a page, to give the reader a quick overview of what others have associated with the current page. Tags can also be used to improve the information retrieval components by personalizing the search based on how users distribute their tags [10]. Wikulu could assist the tagging process by **providing tag suggestions** relying on keyphrase extraction [9, 23, 28]. Tag suggestions address two problems: first, they simplify the process of adding tags, and second, they can be used to foster a standardization of tag usage, avoiding e. g. misspellings. Keyphrases are useful as tag suggestions because a lot of tags actually describe the page content and thus resemble keyphrases [12].

There are also cases where existing pages, e. g after a long time of editing, show the need for restructuring. It might be the case that pages get too long and should be split into multiple smaller pages, or that content that belongs together is actually located on different pages. Both cases signify the need to improve the structure and require re-structuring. Wikulu will **aid the process of splitting pages** by proposing section boundaries, which are identified by applying text segmentation [4, 16]. The user can initiate the split by confirming the suggested boundaries, creating either a better section structure on the original page or even new pages which are semantically more cohesive than the original one.

The need to merge pages might also arise when topically related content is added to different pages in the wiki. Wikulu will **suggest merging candidates** using text similarity measures [13, 34], as described in Section 2.1.

2.3 Finding Content

There are two ways of finding content in a wiki: searching and browsing. By searching, we mean querying the wiki for information by entering a free form text string. Browsing comprises the use of links on pages, as well as tags, categories, and other navigational facilities presented as hyperlinks.

As mentioned earlier, users might lose their trust in the search engine capabilities when a search for a document that is known to be in the wiki fails [1]. This stands in contrast to searching the web, where the document base is vast and search results should be as precise as possible. Thus, high recall is important, but many standard search engines are not able to find all relevant pages. Query terms do not always match the document terms, although they are semantically related and thus relevant to the query. Technologies to bridge the vocabulary gap have already been investigated [13, 25, 34]. Thereby, queries are not only matched word by word, but also by their semantic relatedness to document terms. For example, a user is looking for a solution to a problem, thus including "problem" as a keyword in the query. However, the page with the solution may only contain the synonym "issue". Using standard, keyword-based information retrieval, the page would not be found, but if the high semantic relatedness between "problem" and "issue" is taken into account, the appropriate page is retrieved. This technology can be configured for use with the wiki docu-

[3] http://del.icio.us/
[4] http://www.flickr.com/

ment collection, **improving recall** and hopefully the overall search effectiveness.

Browsing content is already improved by the means Wikulu will provide for organizing the wiki. Due to aiding the user while adding and organizing content, the pages will have a better structure because of the text segmentation, splitting and merging support. In addition, there will be more ways to navigate the wiki, possibly already created when the content is added. Wikulu can add another way of navigation by **presenting links to related pages**, without explicit user interaction e. g. by pressing a button. These dynamic links improve the browsing experience, and can additionally serve as suggestions for further, static links to add to the page.

2.4 Authoring a Page (A Use Case)

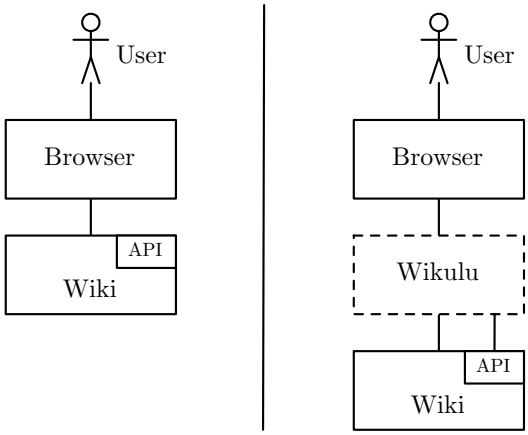
We will now have a look at how Wikulu will improve the process of adding a page to the wiki. The steps of adding content to a traditional wiki, without the support given by Wikulu, are the following:

1. Search for the content, check if it is in the wiki already (optional).
2. If it is available, do not add anything, or merely add some details to the content.
3. If there is no page with the content, either merge the new content with an existing page where it fits (retrieved in step 1), or create a new page and place it there.
4. Link the added content to other relevant pages in the wiki (optional).
5. Add tags describing the content (optional).

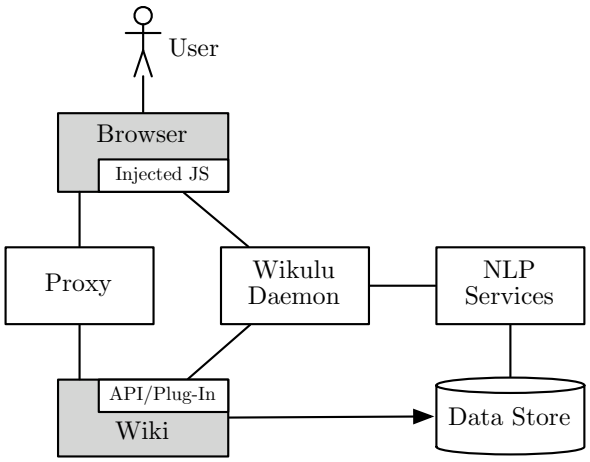
The authoring steps described above are labor-intensive when not supported by intelligent user interfaces. Those steps targeted at organizing the wiki are actually optional and can easily be skipped. Linking to related pages is often neglected, because it is tedious to identify possible link targets and add the links using a text editor. However, as there is no fixed navigation by which single pages can be reached, and horizontal navigation by following links on a page is used often [1], a high link density is essential for wikis. In the Wikulu approach, we try to address both issues of reducing duplicate content and linking newly added content appropriately. Wikulu will also suggest tags to add further navigational facilities.

Using the organization capabilities of Wikulu as presented in Section 2.1 and 2.2, the process of adding content to a wiki described above will be enhanced as follows, as the result of applying the intelligent user interface.

1. Enter the content in an editor.
2. Look at suggestions, live-updated while editing.
   (a) Check if highly similar pages already contain the created content and stop if appropriate.



(a) Traditional Wiki vs. Wikulu Approach



(b) Detailed Overview of Wikulu Components

Figure 2: Wikulu Architecture

   (b) Check suggestions for adding content to one of the highly similar pages.
   (c) Check suggestions for links to relevant pages.
   (d) Check suggested tags.

The user will only be concerned with important activities, namely creating content for the wiki. Organizational concerns will be alleviated by Wikulu's suggestions (a), (b), (c), and (d), significantly simplifying the task of organizing the wiki.

Another detail that becomes apparent in this use case is that adding and searching content are not separated as strictly in Wikulu as in traditional wikis. The same technologies that are used when "finding content" are also used for identifying similar pages when "adding content".

Figure 3: Wikulu Keyphrase Highlighting

3. ARCHITECTURE

As described in the previous sections, the architecture to enable an intelligent user interface in Wikulu has to meet the following requirements:

1. Present suggestions regarding the link structure, tags, page segments, and possible points of content insertion. These suggestions are either a response to a user request or are displayed upon loading an ordinary page in a proactive manner.

2. Allow the user to easily accept suggestions. The system needs to be able to perform actions on behalf of the user.

To satisfy these two requirements, it is necessary to integrate our services into the original wiki interface displayed in the browser. A possible solution would be to integrate the Wikulu system directly into a specific wiki platform. However, there are a lot of wiki platforms in use. MediaWiki[5], Confluence[6], and Twiki[7], in addition to a lot of other ones are used in corporate environments. When trying to support multiple of these platforms, the effort of integrating Wikulu into each wiki platform is prohibitive and a generic approach is needed. Another reason for de-coupling intelligent NLP techniques from the actual wiki platform is that we focus on providing suggestions to the user, without being targeted at any specific wiki platform, and thus there is no need to tie to any specific one. Adaptation to specific wiki platforms is still necessary, but our architecture tries to minimize the coupling points, reducing the effort of integrating Wikulu into a new platform.

To achieve a tight integration with the user interface but still maintain independence of the underlying wiki platform, we use a proxy architecture (see Figure 2) inspired by AUGUR [15], a research project on proactive user interfaces. The basic idea is to pass requests to the wiki through a proxy server, which intercepts the returned HTML representing the requested wiki page. The HTML is modified by adding JavaScript code, which in turn can be used to add additional elements to the HTML on the fly using jQuery[8]. Such elements include buttons invoking NLP techniques, providing an interface to web services with the NLP functionality (see e. g. Figure 3, where Wikulu adds a panel to invoke the functionality on top of the standard Wikipedia interface), or even search results delivered by an IR engine. In addition to buttons where the user has to explicitly activate a certain service, the proxy architecture can also be used to immediately populate the current page with relevant information, e.g. important keyphrases or links to related pages.

3.1 Wikulu Components

Figure 2 gives an overview of the Wikulu architecture. In this section, we provide a detailed description of all architecture components.

[5]http://www.mediawiki.org/
[6]http://www.atlassian.com/software/confluence/
[7]http://twiki.org/
[8]http://www.jquery.com/

User The user employs the standard browser interface of the wiki platform. Wikulu adds or changes some interface elements to allow the user to call our services or see the results. Results are shown either upon a direct user request, e. g. when initiating a search, or automatically as soon as the page is loaded. This is implemented in JavaScript, which is used to modify the HTML and to call the methods provided by the *Wikulu Daemon*, which serves as a middle tier.

Proxy The proxy is a Java Servlet that intercepts the interaction between a user's browser and the wiki. Its task is to add additional JavaScript and CSS references to the original HTML page rendered by the wiki. The references are used to load custom code into the browser, enabling us to augment the user interface in an easy and extensible way.

Wikulu Daemon The *Wikulu Daemon* is realized as a Java Servlet. It acts as a management instance, delegating calls to *NLP Services* and sending the results back to the user interface, e. g. to display possible topic boundaries in the browser, or modifying the wiki according to the results, e. g. if the user confirms a page split.

The *Wikulu Daemon* can operate with different levels of access to the wiki:

1. The first and most simple method to get data for NLP processing is to extract the textual content from an HTML page. To do this, no additional means are necessary other than accessing information that is available in the browser. This method is used e. g. for keyphrase extraction.
2. More elaborate algorithms need a refined way of accessing the wiki data. Information retrieval methods need an index that has to be created in advance, and pages need to be processed beforehand as a batch process. To get a list of all pages, crawling the wiki or screen scraping the list of all pages is a lot of effort, especially as we aim to support different wiki platforms with a minimal effort. A simpler way of reading arbitrary data from a wiki is using an API, if one is available (e. g. for MediaWiki), or by writing a plug-in, exposing the internal API where needed (e. g. for TWiki), constituting the second level of access.
3. The third and most important access level is writing data back to the wiki, which is needed for the user to accept Wikulu's suggestions. This level of access is needed e. g. to confirm suggested links or topic boundaries.

The different levels of access are especially important when the need of using Wikulu with hosted wiki platforms like SocialText[9] arises, where access to internal wiki data might not be as simple as with wikis running on a server in the corporate environment itself.

The current API to access an underlying wiki platform is de-coupled from the actual wiki platform and consists of three principal methods:

get_content Retrieve the content of a single wiki page.

write_content Write content to a single wiki page.

list_pages Retrieve a list of all pages in the wiki.

NLP Services The *NLP Services* offer the functionality of the NLP algorithms as web services. The *Wikulu Daemon* queries these web services, e. g. for keyphrases or topic boundaries. The *NLP Services* have different requirements regarding their data. Some services, like keyphrase extraction, do not need anything except the page from which to extract the keyphrases, and others, like search services, need a highly optimized data structure to operate efficiently. This data needs to be extracted from the wiki off-line, before the actual user interaction, and is stored in the *Data Store*.

The *NLP Services* themselves are based on UIMA[10], the Unstructured Information Management Architecture [8], and the text processing pipelines are constructed based on the Darmstadt Knowledge Processing Repository (DKPro)[11] components [26]. This approach allows us to quickly switch between different algorithms to fulfill NLP tasks. This is very important because we need to support a wide range of NLP tasks, and it simplifies testing algorithms for their suitability in the special wiki environment.

Data Store The data store contains all data structures needed for the NLP algorithms, like a search index for information retrieval or pages pre-processed by a part-of-speech tagger.

So far, the Wikulu architecture described above has been successfully implemented for MediaWiki and TWiki. One challenge when using a proxy architecture is the effort of keeping the data in sync between the wiki and the data store, e. g. when updating the search engine's index. This is easier to achieve with a system that is directly integrated into the wiki platform, as this allows to act on different events, e. g. page saves, which in turn can be used to update any dependent store. Currently, we have to rely on batch updates of the data store instead. We could also execute an update upon any change using Wikulu, but the wiki can still be used without a proxy and so possible changes could go unnoticed. Nevertheless, if a real need for perfectly synchronized data arises, a plug-in for a particular wiki platform can be created that takes care of this specific task. The effort of creating an additional abstraction layer for the underlying wiki platforms is in our opinion outweighed by the clean separation between the Wikulu components and the wiki platform, which allows us to re-use Wikulu user interface elements across different wiki platforms. If we had tightly integrated Wikulu into a single wiki platform, we might not have had to resort to JavaScript/AJAX manipulation of the HTML content, but render it directly within the wiki platform. However, in order to provide an interactive interface, e. g. to provide live-updates, using JavaScript is probably necessary anyway.

[9] http://www.socialtext.com/

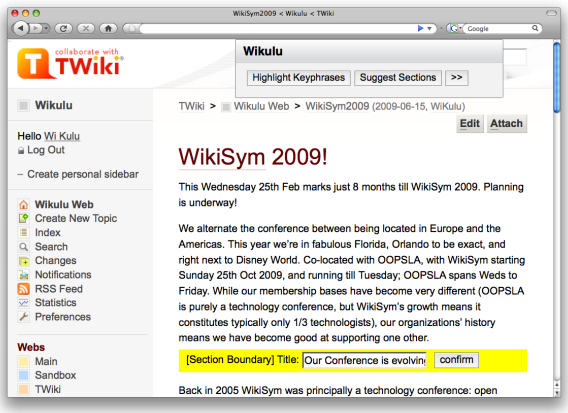
[10] http://incubator.apache.org/uima/

[11] http://www.ukp.tu-darmstadt.de/software/dkpro/

3.2 Implementation Examples

The proposed architecture is able to support a wide range of NLP techniques, see Section 2. To provide proof-of-concept examples of these techniques, we have implemented keyphrase extraction, text segmentation, and search algorithms, and will now describe how they are used in Wikulu. The search is part of the support for authoring pages, as described in the use case in Section 2.4.

(a) Wiki Page without Sections



(b) Wikulu Segmentation Suggestions

(c) Confirmed Section Boundary

Figure 4: Adding Sections to a Wiki Page

Text Segmentation. Text segmentation algorithms will serve two main purposes in Wikulu. One is to provide a page segmentation as pre-processing for content insertion or for splitting a wiki page. The second one is to help the user by suggesting possible places for section headings in the current page. The user initiates the suggestions by clicking a button that has been created using the injected JavaScript. The button click calls a JavaScript function that is mapped to a Java call on the *Wikulu Daemon*. The *Wikulu Daemon* invokes the web service running a text segmentation algorithm. The resulting offsets of the text segmentation are translated to the HTML content and Wikulu shows the possible locations for section headings using a horizontal yellow bar, which includes a link to confirm the suggested location. Technically, this is a `<div>` element that is added to the HTML content using jQuery. Clicking on the confirmation link updates the original wiki page by changing the wiki syntax of the page and storing it, again by invoking a method on the *Wikulu Daemon* using a JavaScript call. The *Wikulu Daemon* communicates with the wiki using the wiki API or a custom plug-in.

The steps of segmenting a Wiki page are shown in Figures 4(a), 4(b), and 4(c). Figure 4(a) shows a long page without any sections that needs additional structuring. By pressing the "Suggest Sections" button, a user can request suggestions for a possible page segmentation, as shown in Figure 4(b). To confirm a suggestion, the user can enter an appropriate title and then click the "confirm" button. The title is then written back to the wiki, as shown in Figure 4(c). To enhance this process of section splitting, we are going to allow the proposed location of the section boundaries to be adjusted.

Wikulu currently uses the *TextTiling* [16] algorithm for text segmentation. Roughly, it works as follows:

1. The document is tokenized (i. e. split into single words) and subdivided into token sequences of a fixed size.
2. A score between each token sequence (or group of token sequences) is computed by taking into account two features:
   - The number of words they share. The more words are shared between groups of token sequences, the higher the score.
   - The ratio of new words introduced in two adjacent token sequences. The more new words are introduced in the two sequences, the higher the score.
3. Boundaries between token sequences are identified by looking for significant lows in the scores computed in the second step.

Keyphrase Extraction. Keyphrases help users to quickly comprehend the content of a page [3]. In the current implementation, the keyphrase extraction is invoked by pressing the "Highlight Keyphrases" button. The click on the button invokes a Java method on the Wikulu Daemon, which in turn calls a Web Service running the keyphrase extraction.

The Web Service obtains the plain text version of the wiki page as input, and returns the extracted keyphrases as the result. The keyphrases are highlighted in the current wiki page (see Figure 3).

Wikulu currently uses a variant of the *TextRank* algorithm [23] to extract keyphrases. It works as follows:

- The document is split into single sentences and tokenized.
- The base form for each token is determined (e. g. "walking" is reduced to "walk").
- Keyphrase candidates are all adjective/noun phrases corresponding to specific patterns (e. g. "new car", "natural language processing").
- A co-occurrence graph of the keyphrase candidates is built. Two candidates are connected in the graph if they appear together in a certain context window.
- PageRank [27] is used to rank the graph nodes representing the keyphrases.
- The candidates that are ranked highest are presented as keyphrases.

Adding Content. A user authoring content can be supported in multiple ways, as described in Section 2.4. An early implementation of this support displays pages related to the text currently entered while adding a new page, as shown in Figure 5. The related pages are not discriminated into potential duplicates or link targets in the current implementation, and are retrieved by standard vector-based information retrieval. In the example shown in Figure 5, a user has typed the text "The founder of the mathematica", and Wikulu has found a page related to the query already present in the wiki, namely a page about the Wolfram|Alpha search engine. The two pages are related because the company developing both projects is the same — Wolfram Research — and the article about Wolfram|Alpha mentions Mathematica.

4. RELATED WORK

Augmenting wikis with NLP techniques has not attracted a lot of research attention yet. A notable exception is the work by Witte and Gitzinger [32]. They propose a multi-tier architecture to connect wikis to services providing NLP functionality, which are based on the *General Architecture for Text Engineering (GATE)*[12] [6]. Among the services provided are automatic summarization, question answering, and index generation. Although the authors hint at using advanced user interface features to present the NLP services to the user, the system makes use of a separate application to apply the services to the wiki content. The application reads the content of the wiki page, applies NLP algorithms to it, and writes the modified page back to the wiki. The system accesses the underlying wiki using the Java Wiki Bot

[12]http://gate.ac.uk/

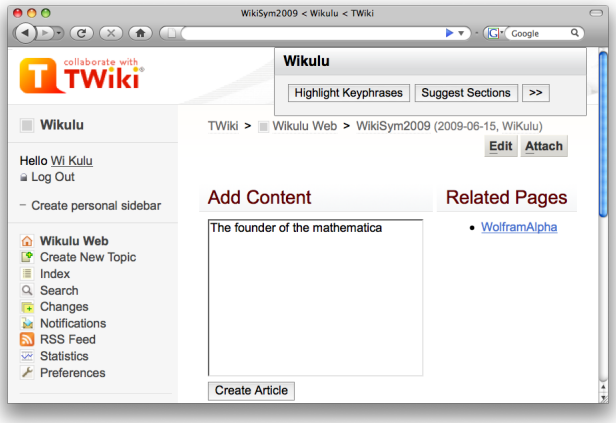


Figure 5: Wikulu Showing Related Pages While Adding Content

Framework[13], generally treating the NLP services as a separate user of the wiki. In contrast to their approach, Wikulu integrates new interface elements into the wiki's user interface, supports a wider range of interaction types, and focuses on improving the organization of a wiki.

Other non-NLP approaches to organize wikis rely on Semantic Web [30] technologies to uncover the semantics of a page, creating so-called semantic wikis. The focus here is on semantically annotating the content in the wiki, which in turn allows structured search. Semantic wikis like Semantic MediaWiki [18], Ikewiki [29], or SweetWiki [2] allow relations between different pages or between pages and content snippets to be classified as a certain type, e.g. allowing users to annotate the population figure of a city as such, rendering it machine-readable. Ikewiki also allows pages to represent concepts, the basic unit of an ontology. Machine-readable concepts and relations form the basis for the Semantic Web, and so the underlying technologies for these types of wikis are the Semantic Web ones like OWL/RDF and SPARQL. SweetWiki addresses the organizational problems in wikis by giving the users the possibility to tag pages, showing related pages for entered tags in real-time. NLP techniques like ontology learning and population [5] might be used to (semi-)automatically add semantic annotations to wikis, and the Wikulu architecture could be used to support this. However, the required NLP algorithms are not mature enough to be applied on a large scale [11]. Semantic wikis allow a fine grained manual annotation of semantic knowledge, thus organizing the semantic content of the wiki itself. Our goal, in contrast, is to aid users with their everyday tasks based on wikis, using NLP methods to improve the organization of the wiki.

A lot of work in the wiki community is done in the context of Wikipedia. There is a Firefox plug-in for Wikipedia called "Smarter Wikipedia"[14] that displays related articles as part of the current page. The plug-in is popular, with 60,000 downloads as of the time of writing, and it shows that peo-

[13]http://jwbf.sourceforge.net/
[14]http://wikiatic.com/

ple accept suggestions and find it useful when trying to find information. However, it is tailored to Wikipedia and cannot be used with other wiki platforms. Another system for Wikipedia is "Can we link it"[15], which suggests possible links for Wikipedia articles, and thus helps with authoring new or improving existing content. Our approach goes beyond the "Smarter Wikipedia" plug-in as well as "Can we link it", as it can be applied to arbitrary wiki platforms and content, and will support a much wider array of tasks.

There has been a lot of additional research on wikis and Wikipedia in the field of NLP, but mostly this work focuses on using Wikipedia as a collaboratively created semantic resource to improve NLP algorithms [21, 33]. In contrast, we focus on using NLP for improving wikis. These two approaches are not mutually exclusive, though. NLP can help to improve the wiki, which can then in turn be mined to enhance NLP algorithms, resulting in a mutual benefit (as depicted in Figure 1).

In a broader context, our work is about the content quality in wikis. Related work from the area of quality assessment in wikis has been done by Stvilia et al. [31]. They use information quality metrics to discriminate high quality articles from low quality ones in Wikipedia. The metrics are based on different features, like *readability*, *currency* and *completeness* — which also takes into account the link structure of the article. Some of the features are specific to encyclopedias, but most of them can also be applied to generic wikis. Quality assessment tries to determine the quality of wiki pages, which helps users decide which entries to trust and which ones need improvement. It does not actively support the user in improving articles of poor quality, though.

5. SUMMARY AND FUTURE WORK

In this paper, we analyzed different types of user interaction regarding the applicability of NLP techniques to support them. We found that the major interaction types are *adding*, *organizing* and *finding* content, and that we can provide benefits to the user for every type of interaction by using NLP techniques. *Adding* and *organizing* content can be simplified by providing suggestions based on NLP techniques, and *finding* content is supported by the improved organization in the wiki as well as an improved search using NLP techniques. We proposed an architecture for creating an intelligent user interface for wikis, which allows such NLP techniques to be integrated into existing wiki platforms. We implemented proof-of-concept examples demonstrating the architecture.

We will extend the Wikulu architecture by implementing NLP services needed for additional capabilities, e. g. suggesting links and tags, or detecting duplicates. The user interface enhancements will be further evaluated regarding their usability.

6. ACKNOWLEDGMENTS

The project has been supported by the Klaus Tschira Foundation under grant No. 00.133.2008. This work has been supported by the Volkswagen Foundation as part of the Lichtenberg-Professorship Program under grant No. I/82806.

[15]http://can-we-link-it.nickj.org/

7. REFERENCES

[1] M. Buffa. Intranet Wikis. *Proceedings of the IntraWebs Workshop 2006 at the 15th International World Wide Web Conference*, 2006.

[2] M. Buffa and F. Gandon. SweetWiki: Semantic Web Enabled Technologies in Wiki. *Human Factors*, pages 69–78, 2006.

[3] E. H. Chi, M. Gumbrecht, and L. Hong. Visual Foraging of Highlighted Text: An Eye-Tracking Study. In *Human-Computer Interaction. HCI Intelligent Multimodal Interaction Environments*, volume 4552 of *Lecture Notes in Computer Science*, pages 589–598. Springer, 2007.

[4] F. Y. Y. Choi, P. Wiemer-Hastings, and J. Moore. Latent Semantic Analysis for Text Segmentation. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pages 109–117, 2001.

[5] P. Cimiano. *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Springer-Verlag, New York, NY, USA, 2006.

[6] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*, pages 168–175, July 2002.

[7] S. Feldman and C. Sherman. The High Cost of Not Finding Information. *An IDC White Paper*, 2001.

[8] D. Ferrucci and A. Lally. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.

[9] E. Frank, G. W. Paynter, I. Witten, C. Gutwin, and C. G. Nevill-Manning. Domain-Specific Keyphrase Extraction. In *Proceedings of the 16th International Joint Conference on Aritificial Intelligence*, pages 668–673, San Mateo, CA, 1999. Morgan Kaufmann.

[10] J. Gemmell, A. Shepitsen, B. Mobasher, and R. Burke. Personalizing Navigation in Folksonomies Using Hierarchical Tag Clustering. *Data Warehousing and Knowledge Discovery*, 5182:196–205, 2008.

[11] L. Getoor and C. P. Diehl. Link Mining: A Survey. *SIGKDD Explorations*, 7:3–12, 2005.

[12] S. A. Golder and B. A. Huberman. Usage Patterns of Collaborative Tagging Systems. *Journal of Information Science*, 32(2):198–208, 2006.

[13] I. Gurevych, C. Müller, and T. Zesch. What to be? - Electronic Career Guidance Based on Semantic Relatedness. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 1032–1039, Prague, Czech Republic, Jun 2007. ACL.

[14] I. Gurevych and T. Zesch. Selbstorganisierende Wikis. In *Proceedings of KnowTech*, pages 317–324, Frankfurt, Germany, Oct 2008. BITKOM.

[15] M. Hartmann, D. Schreiber, and M. Mühlhäuser. Tailoring the Interface to Individual Users. In *5th International workshop on Ubiquitous User Modeling at IUI'08*, New York, NY, USA, 2008. ACM.

[16] M. A. Hearst. TextTiling: Segmenting Text Into Multi-Paragraph Subtopic Passages. *Computational*

Linguistics, 23(1):33–64, 1997.

[17] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, second edition, February 2008.

[18] M. Krötzsch, D. Vrandecic, M. Völkel, H. Haller, and R. Studer. Semantic Wikipedia. *Journal of Web Semantics*, 5:251–261, Sep 2007.

[19] B. Leuf and W. Cunningham. *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley Professional, April 2001.

[20] A. Majchrzak, C. Wagner, and D. Yates. Corporate Wiki Users: Results of a Survey. In *WikiSym '06: Proceedings of the 2006 International Symposium on Wikis*, pages 99–104, New York, NY, USA, 2006. ACM.

[21] O. Medelyan, C. Legg, D. Milne, and I. H. Witten. Mining Meaning from Wikipedia. Working Paper, arXiv:0809.4530v2, 2008.

[22] R. Mihalcea and A. Csomai. Wikify! Linking Documents to Encyclopedic Knowledge. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management*, pages 233–242, 2007.

[23] R. Mihalcea and P. Tarau. TextRank: Bringing Order into Texts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 404–411, Barcelona, Spain, July 2004.

[24] D. Milne and I. H. Witten. Learning to Link with Wikipedia. In *Proceedings of the 17th ACM Conference on Information and Knowledge Mining*, pages 509–518, New York, NY, USA, 2008. ACM.

[25] C. Müller, I. Gurevych, and M. Mühlhäuser. Closing the Vocabulary Gap for Computing Text Similarity and Information Retrieval. *International Journal of Semantic Computing*, 2(2):(253–272), 2008.

[26] C. Müller, T. Zesch, M.-C. Müller, D. Bernhard, K. Ignatova, I. Gurevych, and M. Mühlhäuser. Flexible UIMA Components for Information Retrieval Research. In *Proceedings of the LREC 2008 Workshop 'Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP'*, pages 24–27, May 2008.

[27] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[28] L. Qu, C. Müller, and I. Gurevych. Using Tag Semantic Network for Keyphrase Extraction in Blogs. In *ACM 17th Conference on Information and Knowledge Management*, pages 1381 – 1382, New York, NY, USA, Oct 2008. ACM.

[29] S. Schaffert. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In *WETICE '06: Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 388–396, 2006.

[30] N. Shadbolt, T. Berners-Lee, and W. Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[31] B. Stvilia, M. B. Twidale, L. C. Smith, and L. Gasser. Assessing Information Quality of a Community-Based Encyclopedia. In *Proceedings of the 2005 International Conference on Information Quality*, 2005.

[32] R. Witte and T. Gitzinger. Connecting Wikis and Natural Language Processing Systems. In *WikiSym '07: Proceedings of the 2007 International Symposium on Wikis*, pages 165–176, 2007.

[33] T. Zesch, C. Müller, and I. Gurevych. Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary. In *Proceedings of the Conference on Language Resources and Evaluation*, May 2008.

[34] T. Zesch, C. Müller, and I. Gurevych. Using Wiktionary for Computing Semantic Relatedness. In *Proceedings of AAAI*, pages 861–867, 2008.