Wikulu: An Extensible Architecture for Integrating Natural Language Processing Techniques with Wikis

Daniel Bär, Nicolai Erbs, Torsten Zesch, and 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 *Wikulu*[1], a system focusing on supporting wiki users with their everyday tasks by means of an intelligent interface. Wikulu is implemented as an extensible architecture which transparently integrates natural language processing (NLP) techniques with wikis. It is designed to be deployed with any wiki platform, and the current prototype integrates a wide range of NLP algorithms such as keyphrase extraction, link discovery, text segmentation, summarization, or text similarity. Additionally, we show how Wikulu can be applied for visually analyzing the results of NLP algorithms, educational purposes, and enabling semantic wikis.

1 Introduction

Wikis are web-based, collaborative content authoring systems (Leuf and Cunningham, 2001). As they offer fast and simple means for adding and editing content, they are used for various purposes such as creating encyclopedias (e.g. *Wikipedia*[2]), constructing dictionaries (e.g. *Wiktionary*[3]), or hosting online communities (e.g. *ACLWiki*[4]). However, as wikis do not enforce their users to structure pages or add complementary metadata, wikis often end up as a mass of unmanageable pages with meaningless page titles and no usable link structure (Buffa, 2006).

To solve this issue, we present the *Wikulu* system which uses natural language processing to support wiki users with their typical tasks of adding, organizing, and finding content. For example, Wikulu supports users with reading longer texts by *highlighting keyphrases* using keyphrase extraction methods such as *TextRank* (Mihalcea and Tarau, 2004). Support integrated in Wikulu also includes *text segmentation* to segment long pages, *text similarity* for detecting potential duplicates, or *text summarization* to facilitate reading of lengthy pages. Generally, Wikulu allows to integrate any NLP component which conforms to the standards of *Apache UIMA* (Ferrucci and Lally, 2004).

Wikulu is designed to integrate seamlessly with any wiki. Our system is implemented as an HTTP proxy server which intercepts the communication between the web browser and the underlying wiki engine. No further modifications to the original wiki installation are necessary. Currently, our system prototype contains adaptors for two widely used wiki engines: *MediaWiki*[5] and *TWiki*[6]. Adaptors for other wiki engines can be added with minimal effort. Generally, Wikulu could also be applied to any web-based system other than wikis with only slight modifications to its architecture.

In Figure 1, we show the integration of Wikulu with Wikipedia.[7] The additional user interface components are integrated into the default toolbar (highlighted by a red box in the screenshot). In this example, the user has requested keyphrase highlighting in order to quickly get an idea about the main content of the wiki article. Wikulu then invokes the

[1] Portmanteau of the Hawaiian terms *wiki* ("fast") and *kukulu* ("to organize")

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

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

[4] http://aclweb.org/aclwiki

[5] http://mediawiki.org (e.g. used by Wikipedia)

[6] http://twiki.org (often used for corporate wikis)

[7] As screenshots only provide a limited overview of Wikulu's capabilities, we refer the reader to a screencast: http://www.ukp.tu-darmstadt.de/research/projects/wikulu

Proceedings of the ACL-HLT 2011 System Demonstrations, pages 74–79,
Portland, Oregon, USA, 21 June 2011.

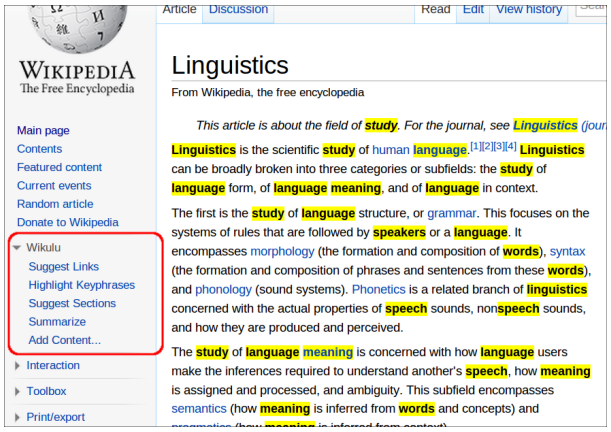

Figure 1: Integration of Wikulu with Wikipedia. The augmented toolbar (red box) and the results of a keyphrase extraction algorithm (yellow text spans) are highlighted.

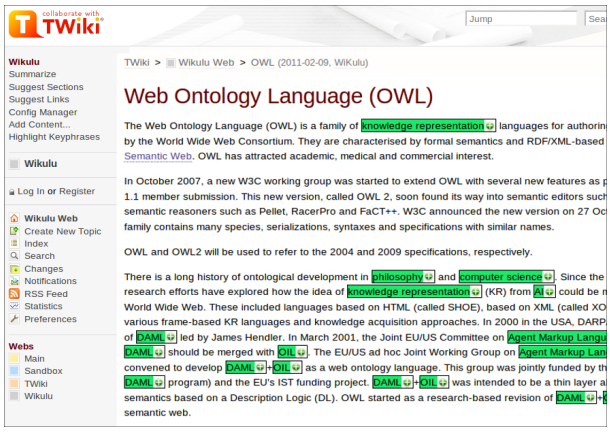

Figure 2: Automatic discovery of links to other wiki articles. Suitable text phrases to place a link on are highlighted in green.

corresponding NLP component, and highlights the returned keyphrases in the article. In the next section, we give a more detailed overview of the different types of support provided by Wikulu.

2 Supporting Wiki Users by Means of NLP

In this section, we present the different types of NLP-enabled support provided by Wikulu.

Detecting Duplicates Whenever users add new content to a wiki there is the danger of duplicating already contained information. In order to avoid duplication, users would need comprehensive knowledge of what content is already present in the wiki, which is almost impossible for large wikis like Wikipedia. Wikulu helps to detect potential duplicates by computing the text similarity between newly added content and each existing wiki page. If a potential duplicate is detected, the user is notified and may decide to augment the duplicate page instead of adding a new one. Wikulu integrates text similarity measures such as *Explicit Semantic Analysis* (Gabrilovich and Markovitch, 2007) and *Latent Semantic Analysis* (Landauer et al., 1998).

Suggesting Links While many wiki users readily add textual contents to wikis, they often restrain from also adding links to related pages. However, links in wikis are crucial as they allow users to quickly navigate from one page to another, or browse through the wiki. Therefore, it may be reasonable to augment a page about the topic *sentiment analysis* by a link to a page providing related information such as evaluation datasets. Wikulu supports users in this tedious task by automatically suggesting links. Link suggestion thereby is a two-step process: (a) first, suitable text phrases are extracted which might be worth to place a link on (see Figure 2), and (b) for each phrase, related pages are ranked by comparing their relevance to the current page, and then presented to the user. The user may thus decide whether she wants to use a detected phrase as a link or not, and if so, which other wiki page to link this phrase to. Wikulu currently integrates link suggestion algorithms by Geva (2007) and Itakura and Clarke (2007).

Semantic Searching The capabilities of a wiki's built-in search engine are typically rather limited as it traditionally performs e.g. keyword-based retrieval. If that keyword is not found in the wiki, the query returns an empty result set. However, a page might exist which is semantically related to the keyword, and should thus yield a match.

As the search engine is typically a core part of the wiki system, it is rather difficult to modify its behavior. However, by leveraging Wikulu's architecture, we can replace the default search mechanisms by algorithms which allow for *semantic search* to alleviate the vocabulary mismatch problem (Gurevych et al., 2007).

Segmenting Long Pages Due to the open editing policy of wikis, pages tend to grow rather fast.

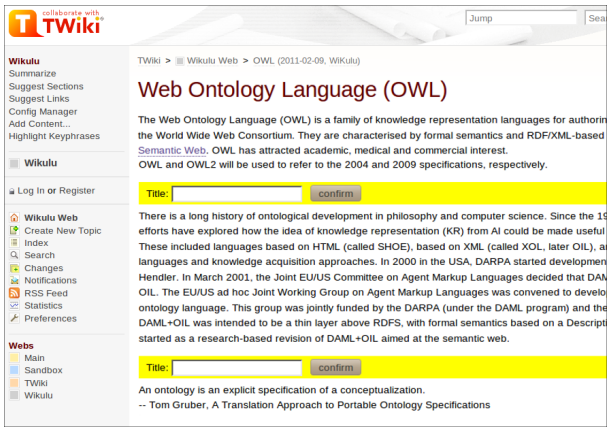

Figure 3: Analysis of a wiki article with respect to topical coherence. Suggested segment breaks are highlighted by yellow bars.

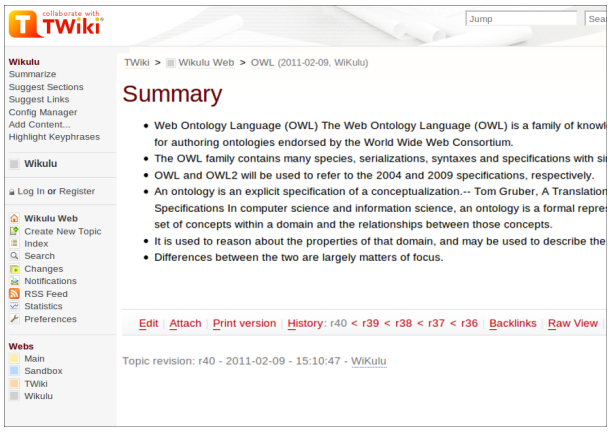

Figure 4: Extractive summary of the original wiki page shown in Figure 3

For users, it is thus a major challenge to keep an overview of what content is present on a certain page. Wikulu therefore supports users by analyzing long pages through employing text segmentation algorithms which detect topically coherent segments of text. It then suggests segment boundaries which the user may or may not accept for inserting a subheading which makes pages easier to read and better to navigate. As shown in Figure 3, users are also encouraged to set a title for each segment.[8] When accepting one or more of these suggested boundaries, Wikulu stores them persistently in the wiki. Wikulu currently integrates text segmentation methods such as *TextTiling* (Hearst, 1997) or *C99* (Choi, 2000).

Summarizing Pages Similarly to segmenting pages, Wikulu makes long wiki pages more accessible by generating an extractive summary. While generative summaries generate a summary in own words, extractive summaries analyze the original wiki text sentence-by-sentence, rank each sentence, and return a list of the most important ones (see Figure 4). Wikulu integrates extractive text summarization methods such as *LexRank* (Erkan and Radev, 2004).

Highlighting Keyphrases Another approach to assist users in better grasping the idea of a wiki page at a glance is to highlight important keyphrases (see Figure 1). As Tucker and Whittaker (2009) have shown, highlighting important phrases assists users with reading longer texts and yields faster understanding. Wikulu thus improves readability by employing automatic keyphrase extraction algorithms. Additionally, Wikulu allows to dynamically adjust the number of keyphrases shown by presenting a slider to the user. We integrated keyphrase extraction methods such as *TextRank* (Mihalcea and Tarau, 2004) and *KEA* (Witten et al., 1999).

[8] In future work, we plan to suggest suitable titles for each segment automatically.

3 Further Use Cases

Further use cases for supporting wiki users include (i) visually analyzing the results of NLP algorithms, (ii) educational purposes, and (iii) enabling semantic wikis.

Visually Analyzing the Results of NLP Algorithms Wikulu facilitates analyzing the results of NLP algorithms by using wiki pages as input documents and visualizing the results directly on that page. Consider an NLP algorithm which performs sentiment analysis. Typically, we were to put our analysis sentences in a text file, launch the NLP application, process the file, and would read the output from either a built-in console or a separate output file. This procedure suffers from two major drawbacks: (a) it is inconvenient to copy existing data into a custom input format which can be fed into the NLP system, and (b) the textual output does not allow presenting the results in a visually rich manner.

Wikulu tackles both challenges by using wiki pages as input/output documents. For instance,

by running the sentiment analysis component right from within the wiki, its output can be written back to the originating wiki page, resulting in visually rich, possibly interactive presentations.

Educational Purposes Wikulu is a handy tool for educational purposes as it allows to (a) rapidly create test data in a collaborative manner (see Section 2), and (b) visualize the results of NLP algorithms, as described above. Students can gather hands-on experience by experimenting with NLP components in an easy-to-use wiki system. They can both collaboratively edit input documents, and explore possible results of e.g. different configurations of NLP components. In our system prototype, we integrated highlighting parts-of-speech which have been determined by a POS tagger.

Enabling Semantic Wikis Semantic wikis such as the *Semantic MediaWiki* (Krötzsch et al., 2006) augment standard wikis with machine-readable semantic annotations of pages and links. As those annotations have to be entered manually, this step is often skipped by users which severely limits the usefulness of semantic wikis. Wikulu could support users e.g. by automatically suggesting the type of a link by means of relation detection or the type of a page by means of text categorization. Thus, Wikulu could constitute an important step towards the *semantification* of the content contained in wikis.

4 System Architecture

In this section, we detail our system architecture and describe what is necessary to make NLP algorithms available through our system. We also give a walkthrough of Wikulu's information flow.

4.1 Core Components

Wikulu builds upon a modular architecture, as depicted in Figure 5. It acts as an HTTP proxy server which intercepts the communication between the web browser and the target wiki engine, while it allows to run any *Apache UIMA*-compliant NLP component using an extensible plugin mechanism.

In the remainder of this section, we introduce each module: (a) the proxy server which allows to add Wikulu to any target wiki engine, (b) the JavaScript injection that bridges the gap between the client- and server-side code, (c) the plugin manager which gives access to any *Apache UIMA*-based NLP component, and (d) the wiki abstraction layer which offers a high-level interface to typical wiki operations such as reading and writing the wiki content.

Proxy Server Wikulu is designed to work with any underlying wiki engine such as *MediaWiki* or *TWiki*. Consequently, we implemented it as an HTTP proxy server which allows it to be enabled at any time by changing the proxy settings of a user's web browser.[9] The proxy server intercepts all requests between the user who interacts with her web browser, and the underlying wiki engine. For example, Wikulu passes certain requests to its language processing components, or augments the default wiki toolbar by additional commands. We elaborate on the latter in the following paragraph.

JavaScript Injection Wikulu modifies the requests between web browser and target wiki by injecting custom client-side JavaScript code. Wikulu is thus capable of altering the default behavior of the wiki engine, e.g. replacing a keyword-based retrieval by enhanced search methods (cf. Section 2), adding novel behavior such as additional toolbar buttons or advanced input fields, or augmenting the originating web page after a certain request has been processed, e.g. an NLP algorithm has been run.

Plugin Manager Wikulu does not perform language processing itself. It relies on *Apache UIMA*-compliant NLP components which use wiki pages (or parts thereof) as input texts. Wikulu offers a sophisticated plugin manager which takes care of dynamically loading those NLP components. The plugin loader is designed to run plugins either every time a wiki page loads, or manually by picking them from the augmented wiki toolbar.

The NLP components are available as server-side Java classes. Via direct web remoting[10], those components are made accessible through a JavaScript proxy object. Wikulu offers a generic language processing plugin which takes the current page contents

[9] The process of enabling a custom proxy server can be simplified by using web browser extensions such as *Multiproxy Switch* (`https://addons.mozilla.org/de/firefox/addon/multiproxy-switch`).

[10] `http://directwebremoting.org`

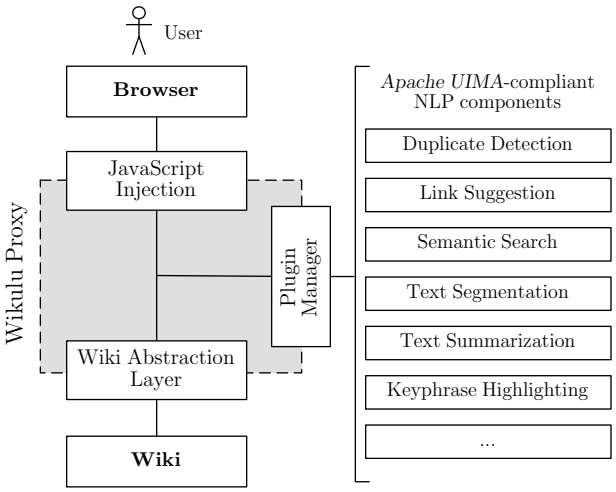

Figure 5: Wikulu acts as a proxy server which intercepts the communication between the web browser and the underlying wiki engine. Its plugin manager allows to integrate any *Apache UIMA*-compliant NLP component.

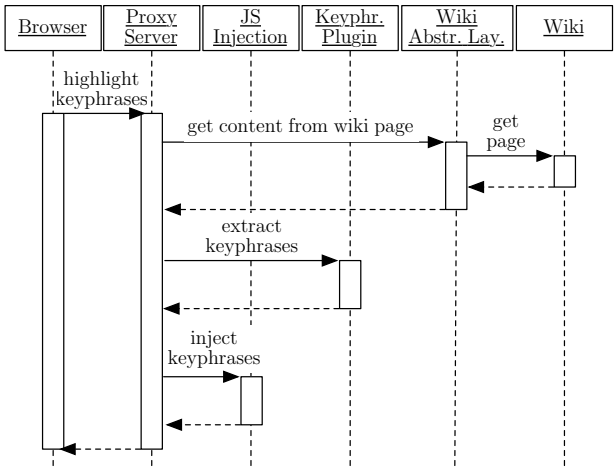

Figure 6: Illustration of Wikulu's information flow when a user has requested to *highlight keyphrases on the current page* as described in Section 4.2

as input text, runs an NLP component, and writes its output back to the wiki. To run a custom *Apache UIMA*-compliant NLP component with Wikulu, one just needs to plug that particular NLP component into the generic plugin. No further adaptations to the generic plugin are necessary. However, more advanced users may create fully customized plugins.

Wiki Abstraction Layer Wikulu communicates with the underlying wiki engine via an abstraction layer. That layer provides a generic interface for accessing and manipulating the underlying wiki engine. Thereby, Wikulu can both be tightly coupled to a certain wiki instance such as *MediaWiki* or *TWiki*, while being flexible at the same time to adapt to a changing environment. New adaptors for other target wiki engines such as *Confluence*[11] can be added with minimal effort.

4.2 Walk-Through Example

Let's assume that a user encounters a wiki page which is rather lengthy. She realizes that Wikulu's keyphrase extraction component might help her to better grasp the idea of this page at a glance, so she activates Wikulu by setting her web browser to pass all requests through the proxy server. After applying the settings, the JavaScript injection module adds additional links to the wiki's toolbar on the originating wiki page. Having decided to apply keyphrase extraction, she then invokes that NLP component by clicking the corresponding link (see Figure 6). Before the request is passed to that component, Wikulu extracts the wiki page contents using the high-level wiki abstraction layer. Thereafter, the request is passed via direct web remoting to the NLP component which has been loaded by Wikulu's plugin mechanism. After processing the request, the extracted keyphrases are returned to Wikulu's custom JavaScript handlers and finally highlighted in the originating wiki page.

5 Related Work

Supporting wiki users with NLP techniques has not attracted a lot of research attention yet. A notable exception is the work by Witte and Gitzinger (2007). They propose an architecture to connect wikis to services providing NLP functionality which are based on the *General Architecture for Text Engineering* (Cunningham et al., 2002). Contrary to Wikulu, though, their system does not integrate transparently with an underlying wiki engine, but rather uses a separate application to apply NLP techniques. Thereby, wiki users can leverage the power of NLP algorithms, but need to interrupt their current workflow to switch to a different application.

[11] `http://www.atlassian.com/software/confluence`

Moreover, their system is only loosely coupled with the underlying wiki engine. While it allows to read and write existing pages, it does not allow further modifications such as adding user interface controls.

A lot of work in the wiki community is done in the context of Wikipedia. For example, the *FastestFox*[12] plug-in for Wikipedia is able to suggest links to related articles. However, unlike Wikulu, FastestFox is tailored towards Wikipedia and cannot be used with any other wiki platform.

6 Summary

We presented Wikulu, an extensible system which integrates natural language processing techniques with wikis. Wikulu addresses the major challenge of supporting wiki users with their everyday tasks. Besides that, we demonstrated how Wikulu serves as a flexible environment for (a) visually analyzing the results of NLP algorithms, (b) educational purposes, and (c) enabling semantic wikis. By its modular and flexible architecture, we envision that Wikulu can support wiki users both in small focused environments as well as in large-scale communities such as Wikipedia.

Acknowledgments

This work has been supported by the Volkswagen Foundation as part of the Lichtenberg-Professorship Program under grant No. I/82806, and by the Klaus Tschira Foundation under project No. 00.133.2008. We would like to thank Johannes Hoffart for designing and implementing the foundations of this work, as well as Artem Vovk and Carolin Deeg for their contributions.

References

Michel Buffa. 2006. Intranet Wikis. In *Proceedings of the IntraWebs Workshop at the 15th International Conference on World Wide Web*.

Freddy Y. Y. Choi. 2000. Advances in domain independent linear text segmentation. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 26–33.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. 2002. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proc. of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 168–175.

Güneş Erkan and Dragomir Radev. 2004. LexRank: Graph-based Lexical Centrality as Salience in Text Summarization. *Journal of Artificial Intelligence Research*, 22:457–479.

David Ferrucci and Adam Lally. 2004. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, pages 1–26.

Evgeniy Gabrilovich and Shaul Markovitch. 2007. Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1606–1611.

Shlomo Geva. 2007. GPX: Ad-Hoc Queries and Automated Link Discovery in the Wikipedia. In *Preproceedings of the INEX Workshop*, pages 404–416.

Iryna Gurevych, Christof Müller, and Torsten Zesch. 2007. 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.

Marti A. Hearst. 1997. TextTiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64.

Kelly Y. Itakura and Charles L. A. Clarke. 2007. University of Waterloo at INEX2007: Adhoc and Link-the-Wiki Tracks. In *INEX 2007 Workshop Preproceedings*, pages 417–425.

Markus Krötzsch, Denny Vrandečić, and Max Völkel. 2006. Semantic MediaWiki. In *Proc. of the 5th International Semantic Web Conference*, pages 935–942.

Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. 1998. An introduction to Latent Semantic Analysis. *Discourse Processes*, 25(2):259–284.

Bo Leuf and Ward Cunningham. 2001. *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley Professional.

Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing Order into Texts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 404–411.

Simon Tucker and Steve Whittaker. 2009. Have A Say Over What You See: Evaluating Interactive Compression Techniques. In *Proceedings of the Intl. Conference on Intelligent User Interfaces*, pages 37–46.

René Witte and Thomas Gitzinger. 2007. Connecting wikis and natural language processing systems. In *Proc. of the Intl. Symposium on Wikis*, pages 165–176.

Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-Manning. 1999. KEA: Practical automatic keyphrase extraction. In *Proceedings of the 4th ACM Conference on Digital Libraries*, pages 254–255.

[12] `http://smarterfox.com`