Connecting Wikis and Natural Language Processing Systems

René Witte Thomas Gitzinger
University of Karlsruhe (TH)
Faculty of Informatics
Institute for Program Structures and Data Organization (IPD)
Karlsruhe, Germany
witte|gitzing@ipd.uka.de

Abstract

We investigate the integration of Wiki systems with automated natural language processing (NLP) techniques. The vision is that of a "self-aware" Wiki system reading, understanding, transforming, and writing its own content, as well as supporting its users in information analysis and content development. We provide a number of practical application examples, including index generation, question answering, and automatic summarization, which demonstrate the practicability and usefulness of this idea. A system architecture providing the integration is presented, as well as first results from an initial implementation based on the GATE framework for NLP and the MediaWiki system.

Categories and Subject Descriptors H.1.2 [*User/Machine Systems*]: Human information processing; H.3.1 [*Content Analysis and Indexing*]: Abstracting methods, Indexing methods, Linguistic processing; H.5.2 [*User Interfaces*]: Natural language, User-centered design; H.5.4 [*Hypertext/Hypermedia*]: Architectures, Navigation, User issues; I.2.1 [*Applications and Expert Systems*]: Natural language interfaces; I.2.7 [*Natural Language Processing*]: Text analysis

General Terms Design, Human Factors, Languages

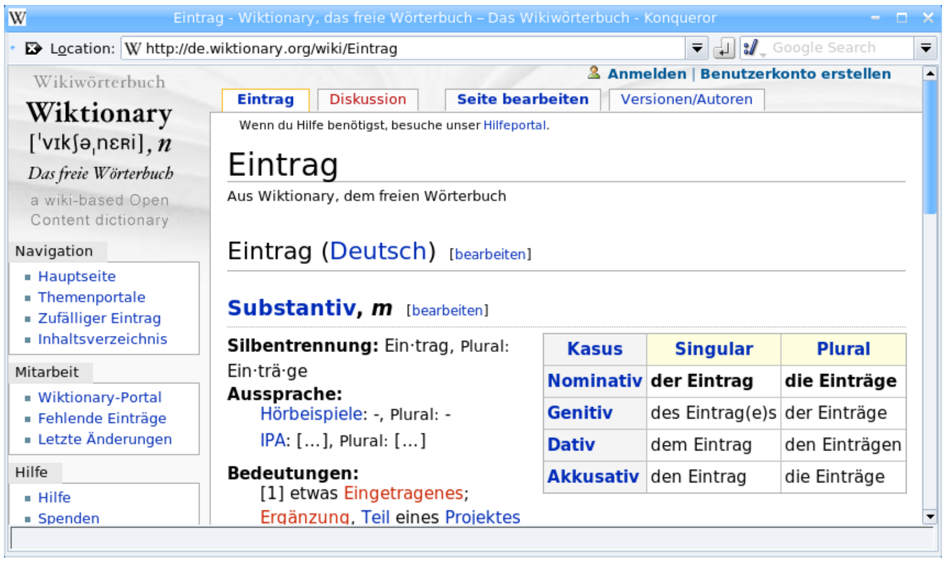
Keywords Self-aware Wiki System, Wiki/NLP Integration

1. Introduction

During the last years, we have seen an impressive proliferation of Wikis as platforms of collaborative knowledge sharing and development, with installations ranging from small collections of loosely connected articles to massive databases of several thousands of texts. Wikis are appreciated by their users for the easy, uncomplicated ways in which content can be added and modified.

However, traditional Wiki systems provide only one view of the database: The articles the users have authored. Of course, considering that they are keyword searchable, such Wikis are a huge step forward from printed document collections when it comes to usability. But there are many real-world situations where a user would like information from several articles combined. A project leader might need a list of the persons a co-worker has worked with in the past, and a reporter might want a summary of several Wiki articles on a series of events. The list of such "special" information needs is potentially endless, and it is—for collections exceeding a few dozens of documents—unfeasible to satisfy them all by hand-crafting special articles. Maintaining lists by hand is cumbersome and error-prone, and it is hard to keep them up-to-date.

Therefore, we need automated methods to satisfy such information needs. Some alleviation has been brought by semantic Wikis, such as Semantic MediaWiki (SMW) [11] or IkeWiki [16]. These systems do not only support *categories* that the subject of an article can be assigned to, but, most importantly, they also allow the user to define *relations* between the subjects of two articles. Categories might be *City*, *State* or *Country*, and if we assume *isLocatedIn* as an example relation, we have the possibility of querying the system for a list of all cities that are known to be located in a certain country. This would not be possible in a traditional Wiki, and it constitutes a great achievement of the semantic Wikis. But again, the semantic queries that we issue can produce no more facts than have been entered *by hand*. If the Wiki page on *San Francisco* does not contain an *isLocatedIn* annotation (or a semantically equivalent one), the system has no way of creating a complete list of cities located in California. While such a situation might merely ask a little more care of authors


Eintrag - Wiktionary, das freie Wörterbuch – Das Wikiwörterbuch - Konqueror

Location: http://de.wiktionary.org/wiki/Eintrag

Wikiwörterbuch
Wiktionary
['vɪkʃəˌnɛʀi], *n*
Das freie Wörterbuch
a wiki-based Open Content dictionary

Navigation
- Hauptseite
- Themenportale
- Zufälliger Eintrag
- Inhaltsverzeichnis

Mitarbeit
- Wiktionary-Portal
- Fehlende Einträge
- Letzte Änderungen

Hilfe
- Hilfe
- Spenden

Anmelden | Benutzerkonto erstellen

Eintrag | Diskussion | Seite bearbeiten | Versionen/Autoren

Wenn du Hilfe benötigst, besuche unser Hilfeportal.

Eintrag

Aus Wiktionary, dem freien Wörterbuch

Eintrag (Deutsch) [bearbeiten]

Substantiv, *m* [bearbeiten]

Silbentrennung: Ein·trag, Plural: Ein·trä·ge
Aussprache:
Hörbeispiele: -, Plural: -
IPA: […], Plural: […]
Bedeutungen:
[1] etwas Eingetragenes; Ergänzung, Teil eines Projektes

Kasus	Singular	Plural
Nominativ	**der Eintrag**	**die Einträge**
Genitiv	des Eintrag(e)s	der Einträge
Dativ	dem Eintrag	den Einträgen
Akkusativ	den Eintrag	die Einträge

Figure 1. Wiktionary: German entry for *Eintrag* (entry)

when they create articles, it constitutes a major problem when we talk about large document inventories already in place, with no semantic markup.

Thus, in this paper we propose to go one step further: Allowing a Wiki to analyze its own content using techniques from Natural Language Processing (NLP). Although a completely automated understanding of natural language is still not feasible, there already exists a number of robust language processing techniques that can significantly improve the experience of a Wiki user, supporting both content development and analysis. We show how these improvements can be delivered to the end user by connecting NLP systems with Wiki installations in an integrated architecture, providing for the first time automated content retrieval, analysis, and generation within a user's client. Our focus here is not on the development of new NLP techniques, but rather showing how existing research and results in language technology can be brought to the end user within a Wiki interface.

In the remainder of this paper, we first present a number of use cases that demonstrate how natural language processing techniques can improve the utility of a Wiki for an end user. We then present our previous work in this area, as well as related work, in Section 3. A generic architecture integrating Wiki systems with NLP is introduced in Section 4, followed by a presentation of our initial implementation in Section 5. Discussions (Section 7) and Conclusions (Section 8) resolve the paper.

2. Use Cases for the Integration of Wikis and Natural Language Processing (NLP)

In this section, we provide a number of use cases that illustrate how existing natural language processing techniques can enhance a Wiki user's experience. Concrete implementations of some of these ideas are presented in Section 6.

2.1 Index Generation

Content in Wikis is typically accessed through information retrieval based on keyword search or by navigating through a list of page names. A full-text index is usually not developed, due to the highly dynamic nature of Wiki pages. Keyword search is very targeted, and will, in most cases, bring up knowledge that we knew or suspected was there. Navigating from article to article through hyperlinks can take us to relevant knowledge that we did not expect, but it does not scale up very well when we are facing several hundreds of articles. A full-text index, organized similar to classical book indices, can help a user "discover" interesting concepts or entities that he did not know were present in a Wiki. Moreover, such an index is well-structured, and aggregates information on a high level. As shown in Section 6.1, a full-text index can be generated automatically using a combination of standard NLP techniques.

This concept can be extended to support the generation of semantic indices, for example, an index of all *persons* or *companies* mentioned in a Wiki, but also more domain-specific indices, e.g., listing all *chemicals* or *proteins* in a Bio-Wiki.

2.2 Content Development

In addition to generating meta-information about existing content, like the index described above, an NLP system can in some instances also be used to create the primary content itself. This is especially useful for highly structured information, like in the Wiktionary,[1] which attempts to create a free, multi-lingual dictionary.

Especially for morphologically complex languages like German, this requires a large amount of manual work for

[1] Wiktionary, http://en.wiktionary.org/wiki/Main_Page

creating the morphological variations for the different cases in singular/plural, as can be seen in the box in the lower right of Figure 1. However, a lot of this content can be generated automatically using methods from computational linguistics. This would allow the (at least partial) creation of Wiktionary entry stubs, which could later be approved and edited in a much more rapid fashion by human users.

To acquire the language-specific entries for the Wiktionary, the NLP system could analyze the Wikipedia for each language and automatically create new article stubs or annotate and cross-link existing ones, further adding to the idea of a "self-aware" Wiki system. However, in other scenarios, external resources could be incorporated as well. For example, a Wiki for "Protein Engineering" could be co-edited by an NLP system [19] analyzing and extracting information from full-text research papers available through PubMed, which are then further edited and enhanced by a human curator.

2.3 Automatic Summarization

Nowadays, users are commonly faced with an abundance of content when accessing Wiki systems through a search interface. The commonly displayed ranked lists with highlighted keywords are not always an effective means for locating the content needed by the user. An alternative is to enrich such a search interface with short, headline-like *summaries* (around 10 words) [3,20] that incorporate the most important concepts of a Wiki page.

In addition, full-text summaries [4, 12] can be created for each page, e.g., with a length of 100 words or more. These summaries can be read much more quickly than a full-length article, thereby helping a user to decide which Wiki pages he wants to read in full.

More advanced types of summaries can support users during both content creation and analysis. *Multi-document summaries* [24] can combine knowledge from several pages within a Wiki or even across Wiki systems. *Update summaries* [21,26] keep track of a user's reading history and only present information he hasn't read before, thereby further reducing the problem of information overload. *Contrastive Summaries* [20, 21] can support a user in highlighting differences across a number of articles (or article versions) on the same topic, thereby showing both commonalities and differences, which could help an editor in resolving different viewpoints on a complex, disputed topic.

2.4 Question-Answering

Closely related to automatic summarization is the field of question-answering (QA). So-called *focused summaries* [20, 21, 25] react to an interest explicitly stated in a user profile. This profile can consist of a number of (open ended) questions, whereby the system creates a summary of a specified length attempting to answer the questions based on the content available in the Wiki or other external resources.

For users of a (public or private) Wiki, this can be particularly effective when knowledge distributed over many pages needs to be examined, compiled, and combined—e.g., to produce a report. With this paradigm, a user might start a new Wiki page for his report by entering a number of questions he needs answered using the available content. The "self-aware Wiki" would recognize the user's need for information, analyze the questions, and automatically create a summary on the same page answering them based on its own content—resulting in a Wiki system that not only allows its users to edit questions and answers, but actively participates in the knowledge development process.

2.5 Summary

Today, Wikis excel in supporting the collaborative creation and editing of content. Developing and changing Wiki pages is often a highly creative and knowledge-intensive task. Our approach here is to support the user by incorporating natural language processing systems that either automatically perform tasks commonly done by hand (e.g., in case of the Wiktionary) or help in filtering and transforming existing information, thereby allowing the user to focus on his work while being supported with an "ambient" intelligence [14] that can sense his information need and avoids to overload him with a multitude of search hits that he has to manually filter, read, and process.

The scenarios outlined above are already possible with today's language technologies, as we will show in Section 6. However, most of this research has been performed within the area of NLP, with little regard on how the resulting advances can contribute to current user interfaces. Thus, in this research we focus on delivering existing natural language processing techniques to the end user by integrating them into a commonly used content creation platform—Wiki systems.

Note that, in this context, we are not concerned with the development or evaluation of new NLP algorithms, like for automatic summarization. This kind of work belongs to the areas of NLP and language engineering. Here, we address two new research questions: (i) how can Wiki systems be integrated with NLP systems, making use of the functionality they offer? And (ii) how can existing NLP technologies enhance the experience of a Wiki user? We address the first question in Sections 4 and 5, where we discuss the design and implementation of our Wiki/NLP-integration. The second question is motivated in this section and further elaborated on in Section 6, where we show detailed application examples for the developed architecture.

3. Existing Work

While we are not aware of previous work examining an integration of natural language processing with Wiki systems, several approaches exist that aim to automate tasks on behalf of Wiki users. In the field of automatic and semi-automatic Wiki content editing, a wide array of *bots*[2] and bot frame-

[2] A list of bots registered with the English Wikipedia can be found at http://en.wikipedia.org/wiki/Wikipedia:Registered_bots.

works have emerged to take over repeated tasks like spell checking, link checking, or disambiguation. While they provide an invaluable service of improving the quality and facilitating the maintenance of a Wiki, the work they perform is mostly on a technical and syntactical level, not on a semantic one. These bots do not perform natural language processing on Wiki content, but they do take advantage of the structure a Wiki offers—like categories—in order to repair or create content without "knowing" in any way about the meaning of that content. Our intent is to go beyond purely syntactic approaches by bringing the full potential of current language technologies to the Wiki platform in order to extend the Wiki's "consciousness" as well as its interaction with users.

We previously proposed a generic architecture integrating content creation, retrieval, and analysis within a unified interface [18]. An application of these ideas has been developed within a semantic desktop for a particular application scenario, the analysis of a history encyclopedia on architecture, with user groups from building history and architecture [22].

As mentioned in Section 1, semantic extensions to Wiki systems, based on *Semantic Web* technologies like OWL ontologies and RDF, provide the means for further structuring and automated processing of Wiki content. Current implementations of these ideas can be found in systems like Semantic MediaWiki (SMW) [11] or IkeWiki [16]. These works are complementary to our approach: They rely on explicitly provided semantic information (e.g., in RDF or OWL) that has to be manually added by a user, whereas we are concerned with automatically extracting semantic information from Wiki content, as well as generating new content with means of natural language processing. We currently investigate the obvious combination of our Wiki/NLP integration with semantic Wiki extensions.

4. System Architecture

We now present our proposed architecture integrating Wiki systems with natural language processing. We do this in two steps: First, in Section 4.1, we derive a more abstract set of requirements based on the scenarios stated above. We then design our architecture to fulfill the stated requirements in Section 4.2.

4.1 Requirements

To allow for a more precise characterization of the integration of Wiki systems with natural language processing frameworks, we derive a number of general requirements for the integration, based on the use cases described in Section 2.

Requirement #1: Content Analysis. The natural language processing system must be able to analyze existing content. It therefore needs to be able to access existing Wiki pages, as well as other (external) resources.

Requirement #2: Content Generation. As motivated in Section 2, the architecture needs to support the generation of content. Hence, it must be possible to automatically create or edit Wiki pages from an NLP system.

Requirement #3: Batch Processing. For some NLP tasks, a periodic analysis of Wiki content is desirable, for example, analyzing all pages that have been changed during a day.

Requirement #4: Service-Based Integration. Wiki systems continue to evolve. Integrating NLP into a Wiki system by directly modifying its code base would lead to a fork, due to the currently highly dynamic nature of this research. Since we want to be able to update the Wiki system independently of the NLP integration, we follow a service-oriented approach, essentially treating the NLP subsystem as another (non-human) "user," as far as the Wiki system is concerned.

Requirement #5: User Agents. Users might want to request direct help from intelligent NLP agents while developing new content. Existing language processing pipelines should be accessible from either a standard Web client or, for specialized applications, from other user clients. This complements the automated NLP tasks performed by the Wiki itself (Requirement #3).

4.2 Integration Architecture

We now discuss the integration architecture we developed based on the stated requirements. As shown in Figure 2, it follows the multi-tier design commonly used with information systems.

Tier 1: Clients. Client programs make up the first tier of the integration architecture, providing access to the system. Standard web clients facilitate access to the Wiki system and are most likely familiar to the reader. Specialized web clients—depicted in the diagram as a web client enhanced by a plugin—can additionally offer access to the NLP services, in accordance with the "User Agents" requirement (#5) identified above. An automatic script, illustrated as the lowermost client, can pull content from the Wiki, send it to the NLP system, and write results back to the Wiki. This constitutes one way of realizing batch processing, as postulated by requirement #3.

Note that, for accessing the system's NLP functionality, the bottom two clients take an intermediate step through an additional client-side connection element. This connective element offers unified access to the server-side NLP functionality and solves common client-side problems. Consequently, not every client has to re-invent the wheel when it comes to doing low-level communication with the server.

Tier 2: Presentation and Interaction. The second tier of the architecture is responsible for interacting with the user or, more generally, with clients. The underlying communication protocol handling is done by a standard web server, apart from which Tier 2 has two major elements: The Wiki System and the NLP Service Connector. In the case of the Wiki system, user interaction consists of presenting the content of the

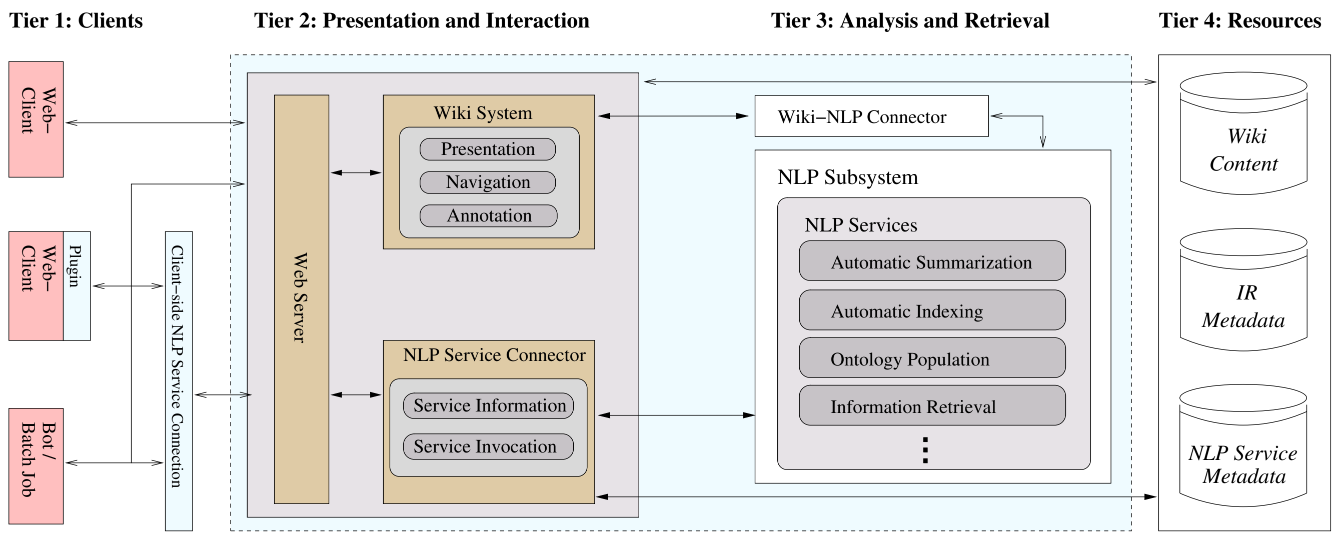


Figure 2. Integrating Wiki systems and natural language processing: Architectural overview

Wiki database and of handling user changes to the content. The functions of the *NLP Service Connector* involve accepting client requests, providing information on the available NLP services, invoking these services with user-provided parameters, and handing the produced results on to the client.

Tier 3: Analysis and Retrieval. This tier contains the functionality that we want to make available for Wikis and potentially other clients. Its heart is the NLP subsystem, with a repository of NLP services that can be of use in one or several of the use cases discussed in Section 2. Processing a user request, the NLP Service Connector in Tier 2 can tell the NLP system which service to run with which parameters. Moreover, the NLP system can itself access existing Wiki content, as well as create new content and add it to the Wiki, by means of an Wiki-NLP Connector. This serves to satisfy requirements #1 (Content Analysis) and #2 (Content Generation).

Tier 4: Resources. The backend tier is made up of the database holding the Wiki content, and additional databases holding metadata on the available NLP services as well as generated index data for information retrieval services and . For every NLP service, there exists a description in the NLP service database. These descriptions are used by the NLP Service Connector to satisfy information requests from clients. Moreover, this database helps to meet requirement #4 (Service Integration), as it enables us simply add a new description whenever a new service is added to the NLP system.

5. Implementation

In this section, we provide technical details on our implementation of the proposed architecture. The implementation has been developed to both show the feasibility of the proposed integration, as well as to gain experience with the user acceptance of novel NLP technology in an existing Wiki environment.

We based our implementation on two popular, open source products: *MediaWiki*,[3] which is a mature Wiki system best known for its use within the Wikipedia, and the *General Architecture for Text Engineering* (GATE)[4] framework [5–7] for developing and deploying NLP systems.

5.1 Content Retrieval

The NLP subsystem performs analysis tasks on a so-called *corpus*, i.e., a collection of documents. Thus, the first step in providing automatic language analysis to a Wiki system is to automatically create such a corpus containing relevant Wiki documents for the task at hand—for example, answering a question. Building a corpus involves two steps: (1) *Information Retrieval* (IR) in order to identify relevant documents and (2) *Document Retrieval* to actually download the documents into a corpus. These steps are illustrated in Figure 3 and described in more detail below.

Information Retrieval. In the following, we use the "answering a question" scenario in order to highlight the most prominent issues around Information Retrieval and NLP analysis. There are two possible retrieval options: If the Wiki system itself offers an IR interface, it can be accessed directly via a wrapper component from within GATE. The component submits the question at hand—or a post-processed variation thereof—as a query and subsequently obtains a (ranked) list of document URIs, which are then fed into the next step, document retrieval.

If the Wiki system has no built-in IR interface, we have to rely on external IR systems. GATE comes with an IR

[3] MediaWiki, http://www.mediawiki.org

[4] GATE, http://gate.ac.uk

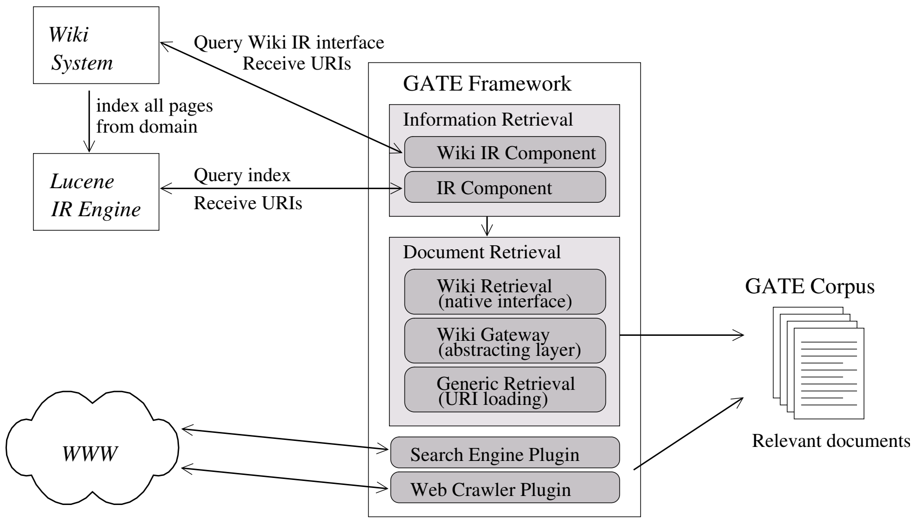


Figure 3. Different possibilities of retrieving relevant documents into the NLP system

component based on Lucene,[5] which is probably the most popular open source full-text IR engine. With Lucene, we can build a local index of a Wiki (or a part of it, like a name space), update this index periodically, and use it to obtain URIs relevant for a given query.

For the cases where a given information need cannot be satisfied from the content of the Wiki system alone, we have to conduct Information Retrieval on other sources. Possibilities include an email database, a local document collection, and the Internet. GATE also offers components to conduct Google and Yahoo! searches, which we can integrate to acquire documents from the Web.

Document Retrieval. Once we obtained the URIs of the documents of interest, we have to assemble a GATE corpus on which to run our NLP pipelines, e.g., for answering questions posed by user. As GATE is fully network transparent, we can simply feed the acquired URIs into a newly created corpus. For the Web IR components mentioned above, this step is actually integrated into the GATE components itself, i.e., they take care of both information and document retrieval.

In order to avoid "noise" and false hits in documents, we use the "printable version" of a MediaWiki page. It would further be advisable to filter out common elements of Wiki pages, such as the table of contents, copyright information, and the like. However, this has not yet been implemented. Such data cleansing steps can obviously be avoided if a Wiki provides direct access to its content via an interface, or if access is performed through an abstraction layer like WikiGateway [17].

Summary. At the end of these steps, we have a set of documents ready for processing by the NLP subsystem. This set is usually transient, i.e., after the pipeline finishes we remove the documents with the created corpus. However, the system can be configured to keep external (i.e., downloaded from the Web) or all documents, for further reference by the user or debugging purposes.

5.2 Content Analysis

The GATE environment [5–7] provides a framework for the development of component-based NLP applications. It comes with a wide range of frequently needed components performing standard NLP tasks, like tokenization and part-of-speech (POS) tagging. Such components are assembled into a processing *pipeline*, which is subsequently run on the document corpus.

Developing NLP pipelines for specific tasks is commonly done by language engineers, whereas individual components are designed by specialists in natural language processing or computational linguistics. Components, and therefore complete pipelines, are typically domain- and language-specific, although many low-level tasks can be performed in a largely language-independent fashion, or with shared resources for multiple languages. Within the scope of this paper, we do not further discuss the design of individual NLP analysis tasks, but rather focus on how these can contribute to the user's experience within a Wiki framework. Some concrete NLP applications are described in Section 6; for a more general introduction to NLP, computational linguistics, and text mining, we refer the reader to [1, 8, 9, 13], as well as the GATE user's guide.[6]

5.3 Content Creation and Editing

Once the language processing pipeline has produced a result—in the question-answering scenario this would be an essay-style summary that (hopefully) answers the user's questions—it is time to forward this result to the Wiki. Similarly to the case of acquiring content from the Wiki, there are several

[5] Lucene, `http://lucene.apache.org`

[6] GATE user's guide, `http://gate.ac.uk/sale/tao/index.html`

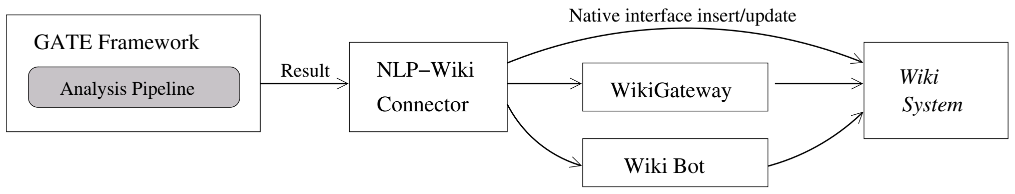


Figure 4. Multiple ways of getting NLP-generated content back into the Wiki

scenarios. The different options for automatically adding content to a Wiki are illustrated in Figure 4 and discussed in detail below.

Screen Scraping. As most Wiki systems were originally designed to output content for humans to read in a Web browser, they typically lack clean, straightforward possibilities for other software to access their content. Therefore, the first applications aiming at automated Wiki content access and manipulation employed a technique called *screen scraping*. Screen scraping applications basically work directly on the HTML pages that make up the Web-based user interface of a Wiki. They parse the code of the HTML pages and extract the page fragments representing the actual, pure Wiki content. During this process, the Wiki software is under the impression it is communicating with an ordinary user controlling a Web browser. As these approaches need to be updated every time the presentation of the content changes, more stable interfaces have been developed that interface with a Wiki system directly.

Python Wikipedia Robot Framework. In our first implementation, we employed a Web Robot *(bot)* based on the *Python Wikipedia Robot Framework (Pywikipedia)*[7] to automatically upload generated content. This framework also gives us the freedom to upload either to the article page itself (if the article is supposed to be generated content only), or to the respective discussion page (if the generated content in some way complements the existing article).

WikiGateway. A noteworthy collection of Wiki-connecting tools is WikiGateway, demonstrated at the 2005 International Symposium on Wikis [17]. It aims at enabling developers to create applications interacting with Wikis, without having to worry about the concrete Wiki engine. It comes with screen scraping functionality for several Wiki engines, but can also handle standard protocols like WebDAV, Atom, and WikiRPCInterface.

Static Content Analysis. A team at the University of Darmstadt has presented JWPL (Java Wikipedia Library), a high-performance, Java-based MediaWiki connector that offers structured access to articles, categories, the link structure, etc. [28]. However, this library works on a modified database structure optimized for speed, meaning that it is not suitable for work with live data. In effect, a snapshot of the Wiki content is converted into an immutable, static version better suited for static analyses. While static analysis is sufficient for a lot of scenarios, it might, in active Wiki communities, be necessary to work with the current content, without any delay. Moreover, the effort of converting the whole content of a Wiki, performing analyses on the converted copy, and writing the changes back to the actual Wiki might quickly become prohibitive for larger Wikis.

Native Interface. If the Wiki system has a native interface for adding content, we can directly use that interface. A very interesting and promising API is in active development for MediaWiki, and its progress can be followed online.[8]

Java Wiki Bot Framework. The *Java Wiki Bot Framework (JWBF)*[9] is a Java library supporting the MediaWiki engine. While it started out employing screen scraping techniques (see above), it now makes increasing use of the possibilities offered by the MediaWiki API. Using a native API is the most preferable way of automatically accessing Wiki content, as one can fully concentrate on accessing the up-to-date content without having to worry about its presentation intended for human users.

5.4 User Interaction

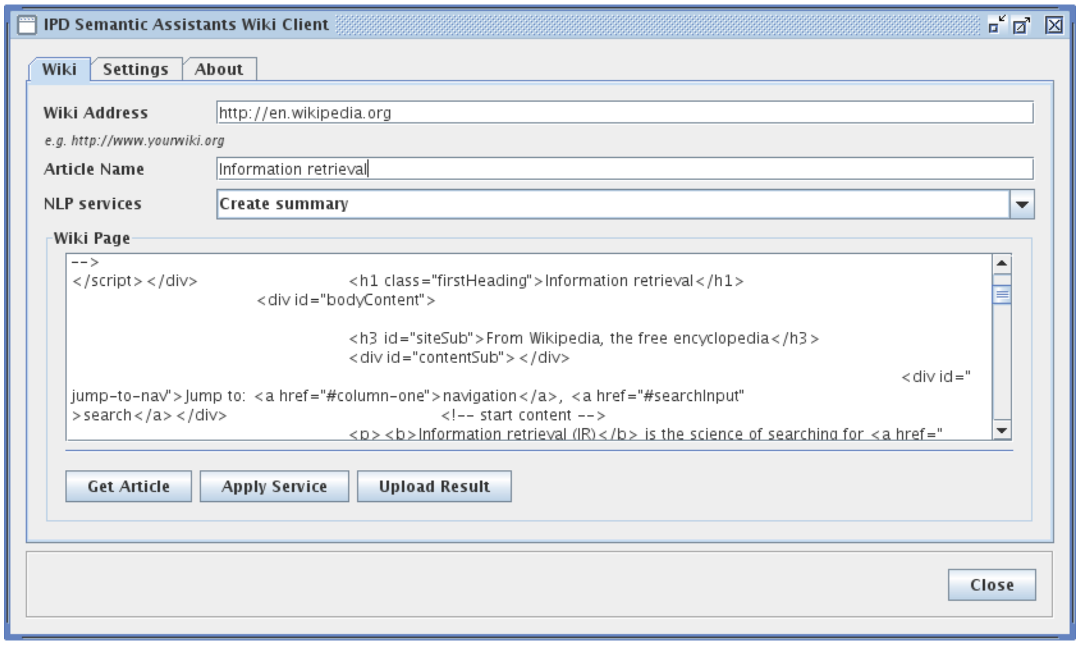
Finally, to allow a user to explicitly request NLP services from a Wiki—rather than waiting for a batch process to complete a number of pre-defined tasks like answering questions—we are experimenting with a simple client GUI that can pull articles from a Wiki and use them as parameters to invoke the NLP service of the user's choice. Optimally, of course, this functionality would be integrated into the Wiki system at hand, so the user would need nothing but her web browser to use it. While this can be achieved with a number of "Web 2.0" technologies like AJAX, we have not implemented this option yet. A screenshot of our client is shown in Figure 5. Here, the Java Wiki Bot Framework was used to establish the Wiki connection.

The screenshot shows a form where you specify a Wiki and the name of the article that you want to work on. Furthermore, there is a drop-down menu listing the available NLP services. The user can now either have the specified article analyzed directly, or she can view it first to confirm that she has selected the right one. The "Apply Service" button sends the article

[7] Pywikipedia, `http://pywikipediabot.sourceforge.net/`

[8] MediaWiki API, `http://www.mediawiki.org/wiki/API`

[9] Java Wiki Bot Framework, `http://jwbf.sourceforge.net/`



Wiki Address: http://en.wikipedia.org
e.g. http://www.yourwiki.org
Article Name: Information retrieval
NLP services: Create summary

Wiki Page

```
-->
</script></div>                    <h1 class="firstHeading">Information retrieval</h1>
                          <div id="bodyContent">

                                   <h3 id="siteSub">From Wikipedia, the free encyclopedia</h3>
                                   <div id="contentSub"></div>
                                                                                     <div id="
jump-to-nav">Jump to:  <a href="#column-one">navigation</a>,  <a href="#searchInput"
>search</a></div>                                <!-- start content -->
                                   <p><b>Information retrieval (IR)</b> is the science of searching for <a href="
```

Figure 5. A simple Wiki client for the use of NLP services with Wiki content

to the specified NLP service, whose result is then displayed back in the text area. If the result is satisfying, the user can add it to the Wiki system by pressing the "Upload Result" button.

While the approach of selecting one Wiki article and one NLP service, and then running the service on that article, is limited in its power to less complex interactions, it is easy to use, and very simple to integrate into existing Wiki systems. Therefore, it is suitable also for non-professionals and can help bringing NLP to the end user.

6. Applications

In this section, we discuss a number of experiments we performed with the implemented system. While this does not constitute a formal evaluation, the examples already demonstrate the utility of the integration of natural language processing with a Wiki interface.

6.1 Full-Text Index Generation

As motivated in the introduction, compiling lists of information concerning all pages of a Wiki is too tedious to be performed by hand. This is an excellent application example for NLP-based content analysis and generation: We developed a simple indexing component that creates a human-friendly full-text index, similar to the ones commonly found in books. To obtain the content for indexing, we retrieve pages as described in Section 5.1. Then, an NLP pipeline with the following four main steps is executed on the retrieved documents:

POS Tagging: Part-of-speech (POS) tagging assigns a label to each token in a document, like *noun, verb,* or *adjective.*

NP Chunking: A chunker[10] then combines individual words to Noun Phrase (NP) chunks, based on the generated POS tags. A noun phrase consists (optionally) of a single determiner and one or multiple modifiers, followed by the (mandatory) head noun. NPs are important since they form the linguistic base for semantic entities occurring in a text. Only the detected NPs are further used for index generation.

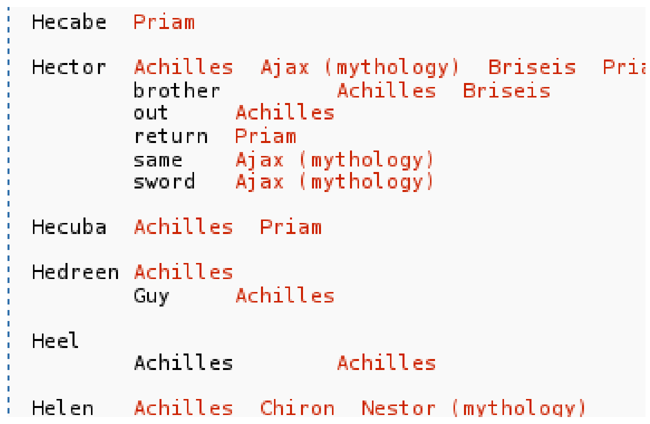
Lemmatization: To reduce each noun to its base form (e.g., mice→mouse), we perform a morphological analysis, which is a language-dependent process. For English, we rely on the analyzer distributed with the GATE architecture [7] and for German we developed our own lemmatization component [15].

Index Generation: The generation component takes the detected NPs and creates an inverted index consisting of the head noun's lemma of each noun phrase as the first-level entry and the NP's modifier(s) as the second-level one. Each entry is automatically back-linked to the page it was found on.

The output of the indexer component is then uploaded into the Wiki into its own page, as described in Section 5.3. The result is a user-friendly index, as shown in Figure 6, which lists index terms structured into two levels, with the page names linked to the actual Wiki pages containing the term or term combination.

This idea can be extended to dynamically create index pages of certain concepts, like a page of all person or or-

[10] Multi-Lingual Noun Phrase Extractor (MuNPEx), `http://www.ipd.uka.de/~durm/tm/munpex/`



```
Hecabe  Priam

Hector  Achilles  Ajax (mythology)  Briseis  Pria
        brother          Achilles  Briseis
        out      Achilles
        return   Priam
        same     Ajax (mythology)
        sword    Ajax (mythology)

Hecuba  Achilles  Priam

Hedreen Achilles
        Guy      Achilles

Heel
        Achilles        Achilles

Helen   Achilles  Chiron  Nestor (mythology)
```

Figure 6. Automatically generated index for a series of Wikipedia articles on Greek mythology

ganization names, or of domain-specific entities, like a list of all proteins in a Bio-Wiki or all method names in a software documentation Wiki. This requires additional taggers or transducers to be incorporated into the pipeline to detect these so-called named entities (NEs). Using the standard NE transducers contained in ANNIE, we successfully created lists of person, location, etc. names; a wide variety of NE detection components are available for other domains, many of them under a free/open source license.

An example for a domain Wiki where automatic index generation already provides benefits to its users is the *Durm* Wiki containing an electronic version of a historic encyclopedia on architecture [22]. Since the printed original did not contain a full-text index, it was previously impossible to obtain a comprehensive overview of the terms discussed in the encyclopedia. Here, the full-text index complements the Wiki search function, as it allows users to discover concepts that they would not have searched for, like archaic terms no longer in use.

6.2 Lexicon Generation

In previous work, we proposed to automatically generate lexicons (for the German language) based on a text mining approach [15], where a system automatically learns and updates lexicon entries by "reading" documents.[11] This work can immediately be applied as an NLP service within our Wiki/NLP-architecture, e.g., for generating content within the Wiktionary as described in Section 2.2.

Instead of manually creating each entry from scratch, we propose to pre-generate a large amount of pages automatically, which would then be checked and enriched by human users in the classical, collaborative fashion of Wikis. Technically, pre-generated vs. user-validated entries could live in different name spaces, allowing users to only query content that has been manually checked vs. a larger (but possible incomplete) knowledge base. This is consistent with other knowledge engineering approaches, e.g., in the life sciences (cf. the Swiss-Prot/UniProtKB protein databases [2]).

This approach allows for much more rapid development of Wiki content. For example, users of the German part of the Wiktionary needed three years to generate 48120 entries (as of May 4^{th}, 2007). Experiments with automatic lexicon generation showed that the same number of entries, containing at least part of this information, could be generated fully automatically within a matter of days [15].

6.3 Automatic Summarization

Summaries help reducing information overload for a user by compressing available content. Automatic summaries are not intended for a replacement of the content they summarize but rather aim to support a human user in better identifying relevant information. To this end, varying types of summaries can be generated that address specific information needs of a user, as outlined in Section 2.3. For example, very short (keyword-style) summaries of about 10 words can augment a search interface, so that a user can better decide which of a Wiki's articles he wants to read while scanning the search results. Additionally, longer summaries (about 100 words) can summarize the main concepts of an article and a multi-document summary can combine content from several articles.

We envision the more advanced types of multi-document summaries, including update, contrastive, and focused summaries (addressed in the next subsection) [21] to be the most helpful for Wiki users, as they combine information across several content pages—or even different Wikis—to provide the user with content he would otherwise have to assemble manually. For example, using update summaries, a Wiki like *Wikinews*[12] can keep track of a user's reading history and deliver event-specific summaries that contain only new information. By requesting contrastive summaries [21], a critical reader could examine how news reports on the same event differ across cultural or geographical boundaries, e.g., as reported by American vs. European vs. Asian sources, as a contrastive summary highlights not only the commonalities (where sources agree on the same event) but also finds their differences.

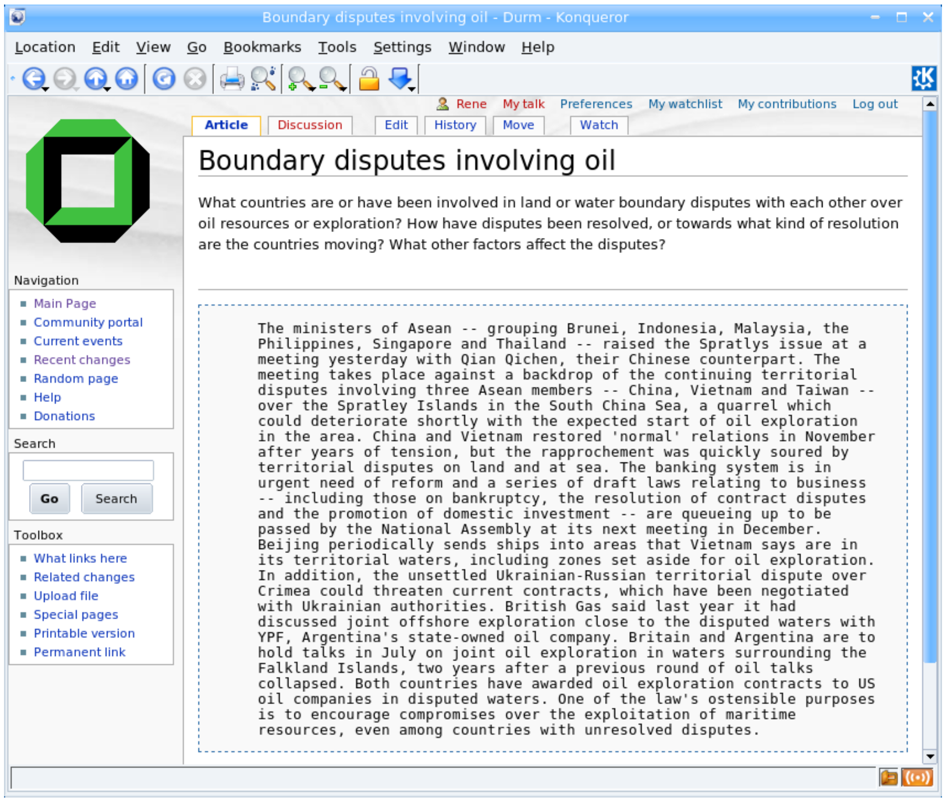
6.4 Question-Answering

As introduced in Section 2.4, automatic summarization can also take a user's current context into account, thereby creating focused summaries [20, 21]. Unlike classical question-answering (QA) aiming at answering factual questions, this kind of summary attempts to address information needs of a user that cannot be satisfied by simply stating, e.g., a name, date, or number.

The question(s) or other information given by a user form the context given to the NLP subsystem. Relevant

[11] The *Durm* German lemmatizer is available under an open source license at http://www.ipd.uka.de/~durm/tm/lemma/.

[12] Wikinews, http://en.wikinews.org/wiki/Main_Page



Boundary disputes involving oil - Durm - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Rene My talk Preferences My watchlist My contributions Log out
Article Discussion Edit History Move Watch
Navigation
Main Page
Community portal
Current events
Recent changes
Random page
Help
Donations
Search
Go Search
Toolbox
What links here
Related changes
Upload file
Special pages
Printable version
Permanent link

Boundary disputes involving oil

What countries are or have been involved in land or water boundary disputes with each other over oil resources or exploration? How have disputes been resolved, or towards what kind of resolution are the countries moving? What other factors affect the disputes?

```
The ministers of Asean -- grouping Brunei, Indonesia, Malaysia, the
Philippines, Singapore and Thailand -- raised the Spratlys issue at a
meeting yesterday with Qian Qichen, their Chinese counterpart. The
meeting takes place against a backdrop of the continuing territorial
disputes involving three Asean members -- China, Vietnam and Taiwan --
over the Spratley Islands in the South China Sea, a quarrel which
could deteriorate shortly with the expected start of oil exploration
in the area. China and Vietnam restored 'normal' relations in November
after years of tension, but the rapprochement was quickly soured by
territorial disputes on land and at sea. The banking system is in
urgent need of reform and a series of draft laws relating to business
-- including those on bankruptcy, the resolution of contract disputes
and the promotion of domestic investment -- are queueing up to be
passed by the National Assembly at its next meeting in December.
Beijing periodically sends ships into areas that Vietnam says are in
its territorial waters, including zones set aside for oil exploration.
In addition, the unsettled Ukrainian-Russian territorial dispute over
Crimea could threaten current contracts, which have been negotiated
with Ukrainian authorities. British Gas said last year it had
discussed joint offshore exploration close to the disputed waters with
YPF, Argentina's state-owned oil company. Britain and Argentina are to
hold talks in July on joint oil exploration in waters surrounding the
Falkland Islands, two years after a previous round of oil talks
collapsed. Both countries have awarded oil exploration contracts to US
oil companies in disputed waters. One of the law's ostensible purposes
is to encourage compromises over the exploitation of maritime
resources, even among countries with unresolved disputes.
```

Figure 7. Automatically generated answer (bottom) for a question (top) created by a user

content can then be retrieved from either the Wiki or an external resource and analyzed, thereby creating a summary attempting to answer the question(s). An example is shown in Figure 7, which is based on a question and data from the *Document Understanding Conference* (DUC) competition in summarization sponsored by the U.S. NIST.[13] Here, the user first creates a new page on a certain topic and enters his question. The NLP system analysis the question, creates a summary and adds the result to the same Wiki page, which could then be further edited by the user. The summary shown is the actual system output of the summarization system ERSS, as it was submitted for the 2005 DUC competition [25].

We currently foresee question-answering within Wikis in two scenarios: an interactive mode, where a summary is requested by a user using the interface described in Section 5.4, and a batch mode, where the NLP subsystem is periodically scanning within a special "question" name space for new questions, adding the answers to detected Wiki pages.

7. Discussion

Automatic summarization and question-answering are a further step towards a "self-aware Wiki" that not only passively delivers content, but actively supports a user in his tasks. In many cases, retrieving Wiki content is not an end unto itself: a user might need the information to complete a certain task, like writing a report, an e-mail, or a new Wiki page. Rather than spending time on searching and filtering the information himself, we envision a Wiki system that can sense a user's need for information and actively provide focused, condensed content relevant for the task at hand. This idea goes far beyond currently available automation tools for Wikis, like the bots employed within Wikipedia.

While there exists various previous work dealing with extracting and using content from Wikis (like the Wikipedia) for NLP tasks, we are not aware of existing work that examines the integration of language technologies into Wiki interfaces to enrich user tasks, which is an important contribution of our work. In contrast to foundation work in NLP, we are concerned with *improving tasks performed by a user* while working with Wikis (in contrast with improving *NLP tasks* by accessing content from Wikis, which is commonly done in NLP research). The proposed architecture easily allows to incorporate novel NLP components by adding them to the language processing subsystem.

Within the *Semantic Web* and semantic desktop communities, more attention is given to the semantics of content, by integrating ontology/RDF-based representations into Wiki systems, like the Semantic MediaWiki extension [11]. This work can be seen complementary to our approaches; they focus on representing and editing semantic content, while

[13] DUC, `http://duc.nist.gov`

we apply NLP techniques that can be employed to obtain semantic descriptions automatically, e.g., through ontology population [10, 23, 27].

8. Conclusions

In this paper, we presented a novel integration of Wiki systems with language technologies. Preliminary experiments based on MediaWiki and GATE show the feasibility of our approach. While the technical foundations will need to be matured, it already shows important points for future Wiki users and developers. For Wiki developers, an increased awareness of language technologies is needed, providing interfaces that allow NLP components to 'read,' 'edit,' and 'write' content, much like a human user would. For the NLP developer, we see a chance to bring contemporary language technology to a larger user base. Finally, Wiki users will, for the first time, benefit from language analysis tools that are currently restricted to expert users of (often highly expensive) text mining systems.

Acknowledgments

Ralf Krestel contributed to the automatic summarization NLP pipelines. Thomas Kappler contributed to the NLP-Wiki upload framework. Praharshana Perera contributed to the automatic index generation.

References

[1] S. Ananiadou and J. McNaught, editors. *Text Mining for Biology and Biomedicine*. Artech House, 2006.

[2] A. Bairoch, R. Apweiler, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, D. A. Natale, C. O'Donovan, N. Redaschi, and L.-S. L. Yeh. The Universal Protein Resource (UniProt). *Nucleic Acids Research*, 33(suppl 1):D154–D159, January 2005.

[3] S. Bergler, R. Witte, M. Khalife, Z. Li, and F. Rudzicz. Using Knowledge-poor Coreference Resolution for Text Summarization. In *Proceedings of the HLT/NAACL Workshop on Text Summarization (DUC 2003)*. Document Understanding Conference, 2003. http://www-nlpir.nist.gov/projects/duc/pubs/2003final.papers/concordia.final.pdf.

[4] S. Bergler, R. Witte, Z. Li, M. Khalife, Y. Chen, M. Doandes, and A. Andreevskaia. Multi-ERSS and ERSS 2004. In *Proceedings of the HLT/NAACL Workshop on Text Summarization (DUC 2004)*. Document Understanding Conference, 2004. http://www-nlpir.nist.gov/projects/duc/pubs/2004papers/concordia.witte.pdf.

[5] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 2004.

[6] H. Cunningham. GATE, a General Architecture for Text Engineering. *Computers and the Humanities*, 36:223–254, 2002. http://gate.ac.uk.

[7] 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*, 2002. http://gate.ac.uk.

[8] R. Feldman and J. Sanger. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, 2006.

[9] D. Jurafsky and J. H. Martin. *Speech and Language Processing*. Prentice Hall, 2000.

[10] A. Kiryakov, B. Popov, I. Terziev, D. Manov, and D. Ognyanoffe. Semantic Annotation, Indexing, and Retrieval. *Journal of Web Semantics*, 2(1), 2005.

[11] M. Krötzsch, D. Vrandečić, and M. Völkel. Semantic MediaWiki. In I. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *The Semantic Web – ISWC 2006*, volume 4273 of *LNCS*, pages 935–942. Springer, 2006.

[12] I. Mani. *Automatic Summarization*. John Benjamins B.V., 2001.

[13] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[14] P. Morville. *Ambient Findability*. O'Reilly, 2005.

[15] P. Perera and R. Witte. A Self-Learning Context-Aware Lemmatizer for German. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP 2005)*, pages 636–643, Vancouver, British Columbia, Canada, October 6–8 2005. Association for Computational Linguistics. http://www.aclweb.org/anthology/H/H05/H05-1080.

[16] S. Schaffert. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In *WETICE*, pages 388–396. IEEE Computer Society, 2006.

[17] B. Shanks. WikiGateway: a library for interoperability and accelerated wiki development. In D. Riehle, editor, *Int. Sym. Wikis*, pages 53–66. ACM, 2005.

[18] R. Witte. An Integration Architecture for User-Centric Document Creation, Retrieval, and Analysis. In *Proceedings of the VLDB Workshop on Information Integration on the Web (IIWeb'04)*, pages 141–144, Toronto, Canada, August 30 2004. http://rene-witte.net/downloads/witte_iiweb04.pdf.

[19] R. Witte and C. J. O. Baker. Combining Biological Databases and Text Mining to support New Bioinformatics Applications. In *Natural Language Processing and Information Systems: 10th International Conference on Applications of Natural Language to Information Systems (NLDB 2005)*, volume 3513 of *LNCS*, pages 310–321, Alicante, Spain, June 15–17 2005. Springer-Verlag.

[20] R. Witte and S. Bergler. Fuzzy Clustering for Topic Analysis and Summarization of Document Collections. In Z. Kobti and D. Wu, editors, *Proc. of the 20th Canadian Conference on Artificial Intelligence (Canadian A.I. 2007)*, LNAI 4509, pages 476–488, Montréal, Québec, Canada, May 28–30 2007. Springer.

[21] R. Witte and S. Bergler. Next-Generation Summarization: Contrastive, Focused, and Update Summaries. In *International Conference on Recent Advances in Natural Language Processing (RANLP 2007)*, Borovets, Bulgaria, September 27–29 2007.

[22] R. Witte, P. Gerlach, M. Joachim, T. Kappler, R. Krestel, and P. Perera. Engineering a Semantic Desktop for Building Historians and Architects. In *Proceedings of the Semantic Desktop Workshop at the ISWC*, volume 175 of *CEUR Workshop Proceedings*, pages 138–152, Galway, Ireland, November 6 2005. http://CEUR-WS.org/Vol-175/34_witte_engineeringsd_final.pdf.

[23] R. Witte, T. Kappler, and C. J. O. Baker. Ontology Design for Biomedical Text Mining. In *Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences*, chapter 13, pages 281–313. Springer, 2007.

[24] R. Witte, R. Krestel, and S. Bergler. ERSS 2005: Coreference-Based Summarization Reloaded. In *Proceedings of Document Understanding Workshop (DUC)*, Vancouver, B.C., Canada, October 9–10 2005. http://duc.nist.gov/pubs/2005papers/ukarlsruhe.witte.pdf.

[25] R. Witte, R. Krestel, and S. Bergler. Context-based Multi-Document Summarization using Fuzzy Coreference Cluster Graphs. In *Proceedings of Document Understanding Workshop (DUC)*, New York City, NY, USA, June 8–9 2006. http://duc.nist.gov/pubs/2005papers/ukarlsruhe.witte.pdf.

[26] R. Witte, R. Krestel, and S. Bergler. Generating Update Summaries for DUC 2007. In *Proceedings of Document Understanding Workshop (DUC) at NAACL-HLT 2007*, Rochester, NY, USA, April 26–27 2007. http://duc.nist.gov/pubs/2005papers/ukarlsruhe.witte.pdf.

[27] M. M. Wood, S. J. Lydon, V. Tablan, D. Maynard, and H. Cunningham. Populating a Database from Parallel Texts Using Ontology-Based Information Extraction. In *9th International Conference on Applications of Natural Language to Information Systems (NLDB)*, volume 3136 of *LNCS*. Springer, 2004.

[28] T. Zesch, I. Gurevych, and M. Mühlhäuser. Analyzing and Accessing Wikipedia as a Lexical Semantic Resource. In G. Rehm, A. Witt, and L. Lemnitzer, editors, *Data Structures for Linguistic Resources and Applications*, pages 197–205. Gunter Narr, Tübingen, Tuebingen, Germany, 2007.