# Natural Language Processing for MediaWiki: The Semantic Assistants Approach

Bahar Sateli and René Witte*
Semantic Software Lab
Department of Computer Science and Software Engineering
Concordia University, Montréal, QC, Canada
[sateli,witte]@semanticsoftware.info

## ABSTRACT

We present a novel architecture for the integration of Natural Language Processing (NLP) capabilities into wiki systems. The vision is that of a new generation of wikis that can help developing their own primary content and organize their structure by using state-of-the-art technologies from the NLP and Semantic Computing domains. The motivation for this integration is to enable wiki users – novice or expert – to benefit from modern text mining techniques directly within their wiki environment. We implemented these ideas based on MediaWiki and present a number of real-world application case studies that illustrate the practicability and effectiveness of this approach.

## Categories and Subject Descriptors

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

## 1. INTRODUCTION

Back in 2007, we originally coined the idea of a 'self-aware wiki system' that can support users in knowledge management by applying natural language processing (NLP) techniques to analyse, modify, and create its own content [7]. As an example, consider a wiki containing cultural heritage documents, like a historical encyclopedia of architecture: the large size and partially outdated terminology can make it difficult for its user to locate knowledge, e.g., for a restoration task on an old building: Couldn't the wiki organize the content by automatically creating a back-of-the book index as another wiki page? Or even create a new wiki page from its content that answers a specific question of a user, asked in natural language?

Or image a wiki used to curate knowledge for biofuel research, where expert biologist go through research publications stored in the wiki in order to extract relevant knowledge. This involves the time-consuming and error-prone task of locating biomedical entities, such as enzymes, organisms, or genes: Couldn't the wiki identify these entities in its pages automatically, link them with other external data sources, and provide semantic markup for embedded queries?

Consider a wiki used for collaborative requirements engineering, where the specification for a software product is developed by software engineers, users, and other stakeholders. These natural language specifications are known to be prone for errors, including ambiguities and inconsistencies. What if the wiki system included intelligent assistants that work collaboratively with the human users to find defects like this, thereby improving the specification (and the success of the project)?

In this paper, we present our work on a comprehensive architecture that makes these visions a reality: Based entirely on open source software and open standards, we developed an integration of wiki engines, in particular MediaWiki, with the Semantic Assistants NLP web services framework [8]. The resulting system was designed from the ground up for scalability and robustness, allowing anyone to integrate sophisticated text mining services into an existing MediaWiki installation. These services embody 'Semantic Assistants' – intelligent agents that work collaboratively with human users on developing, structuring, and analysing the content of a wiki. We developed a wide range of wiki-based applications that highlight the power of this integration for diverse domains, such as biomedical literature curation, cultural heritage data management, and software requirements specification.

The impact of integrating wikis with NLP techniques presented in this work is significant: It opens the opportunity of bringing state-of-the-art techniques from the NLP and Semantic Computing domains to wiki users, without requiring them to have an concrete knowledge in these areas. Rather, the integration will seamlessly provide them with NLP capabilities, so that no context switch is needed, i.e., the NLP service invocation is carried out from within the wiki and results are brought back to the user in his place of choice. This way, a broad range of wiki users, from laypersons to organization employees can benefit from NLP techniques, which would normally require them to use specialized tools or have expert knowledge in that area.

*corresponding author

*WikiSym '12* Aug 27–29, 2012, Linz, Austria
Copyright 2012 ACM 978-1-4503-1605-7/12/08 ...$15.00.

# 2. BACKGROUND

Before we delve into the details of our Wiki-NLP integration, we briefly introduce NLP and illustrate how various techniques from this domain can help to improve a wiki user's experience. Afterwards, we describe the Semantic Assistants framework, which we aim to integrate with MediaWiki in order to provide these NLP services.

## 2.1 Natural Language Processing

Natural Language Processing (NLP) is a branch of computer science that employs various Artificial Intelligence techniques to process content written in natural language. One of the applications of NLP is text mining – the process of deriving patterns and structured information from text – which is usually facilitated through the use of frameworks, such as the General Architecture for Text Engineering (GATE) [1]. Using these frameworks, sophisticated text mining applications can be developed that can significantly improve a user's experience in content development, retrieval, and analysis. Here, we introduce a number of standard NLP techniques to illustrate how they can support Wiki users.

**Information Extraction (IE)** is one of the most popular applications of NLP. IE identifies instances of a particular class of events, entities or relationships in a natural language text and creates a structured representation of the discovered information. A number of systems have been developed for this task, such as OpenCalais,[1] which can extract named entities like *Persons* and *Organizations* within a text and present them in a formal language, e.g., XML or RDF[2]. These techniques are highly relevant for wikis, where users often deal with large amounts of textual content. For example, using an IE service in a wiki can help users to automatically find all the occurrences of a specific type of entity, such as 'protein', and gather complementary information in form of metadata around them.

**Content Development.** In addition to generating metadata from existing content in the wiki, an NLP system can also be used in some instances to create the primary content itself. For example, in Wiktionary,[3] where information is highly structured, various techniques from computational linguistics can help to automatically populate the wiki by adding new stubs and their morphological variations. Furthermore, the NLP system can analyse the wiki content to find the entries across different languages and automatically annotate them or cross-link their pages together.

**Automatic Summarization.** Consider a wiki user who wants to create a report on a specific topic from the available related information in a wiki with the size of Wikipedia. For this task, the user has to start from one page, read its content to determine its relevance and continue browsing through the wiki by clicking on page links that might be related to his topic in mind. This manual approach is not only a time-consuming and tedious task, but also often results in neglection of information due to the existence of isolated wiki pages. In such a situation, where a user's information need is dispersed over multiple documents, NLP techniques can provide him with generic or focused *summaries*: The wiki user can collect the pages of interest or provide a topic keyword, and ask the summarization service to generate a summary with a desired length from the available information on that topic within the wiki.

**Question Answering (QA).** The knowledge stored in wikis, for example, when a wiki is used inside an organization to gather technical knowledge from employees, is a valuable source of information that can only be queried via a keyword search or indices. However, using a combination of NLP techniques, a wiki can be enhanced to allow its users to pose questions against its knowledge base in natural language. Then, after "understanding" the question, NLP services can browse through the wiki content and bring back the extracted information or a summary of a desired length to the user. QA systems are especially useful when the information need is dispersed over multiple pages of the wiki, or when the user is not aware of the existence of such knowledge and the terminology used to refer to it.

## 2.2 Semantic Assistants Framework

The open source *Semantic Assistants* framework [8] is an existing service-oriented architecture that brokers context-sensitive NLP pipelines as W3C standard web services.[4] The goal of this framework is to bring NLP techniques directly to end-users, by integrating them within common desktop applications, such as word processors, email clients, or web browsers. The core idea of the Semantic Assistants approach is to take the existing NLP frameworks and wrap concrete analysis pipelines so that they can be brokered through a service-oriented architecture, and allow desktop clients connected to the architecture to consume these NLP services via a plug-in interface.

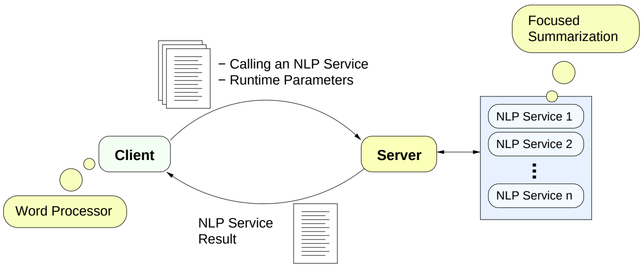


**Figure 1: The Semantic Assistants service execution workflow [8]**

NLP pipelines are stored in a repository in the Semantic Assistants architecture. They are formally described using the Web Ontology Language (OWL),[5] which allows the Semantic Assistants web server to dynamically discover them and reason about their capabilities before recommending them to the clients. Any service deployed in the repository is automatically available to all clients connected to the architecture, using standard WSDL[6] interface descriptions.

The integration of new clients with the architecture is achieved via designing plug-ins, which is facilitated by a Client-Side Abstraction Layer (CSAL). CSAL is essentially a Java Archive library of common communication and data

[1] OpenCalais, http://www.opencalais.com/

[2] Resource Description Framework, http://www.w3.org/RDF/

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

[4] Web Services Architecture, http://www.w3.org/TR/ws-arch/

[5] Web Ontology Language (OWL), http://www.w3.org/2004/OWL/

[6] Web Services Description Language (WSDL), http://www.w3.org/TR/wsdl

```
<saResponse>
<annotation type="Person" annotationSet="Annotation" isBoundless=
      "false">
      <document url="http://localhost/Test_Page">
        <annotationInstance content="Mary" start="0" end="4">
           <feature name="gender" value="female"/>
        </annotationInstance>
      </document>
 </annotation>
 <annotation type="Location" annotationSet="Annotation"
        isBoundless="false">
      <document url="http://localhost/Test_Page">
        <annotationInstance content="Canada" start="14" end="20">
           <feature name="locType" value="country"/>
        </annotationInstance>
      </document>
 </annotation>
</saResponse>
```

**Figure 2: Semantic Assistants server response structure example**

transformation functionality that can be reused by clients to communicate with the Semantic Assistants server and transform NLP results to other useful data types. When a client has been integrated into the Semantic Assistants architecture, all implementation details of the NLP integration are hidden from the user – from their point of view they only see context-sensitive assistants relevant for their task at hand. Currently, the Semantic Assistants architecture natively supports NLP pipelines developed based on the GATE framework, but also provides for the execution of pipelines based on OpenNLP[7] and UIMA[8] through their GATE integration.

As shown in Figure 1, following each NLP service execution in the Semantic Assistants server, results are gathered and passed back to the invoking client. NLP service results can be of any format, depending on the concrete components deployed in the pipeline. Some services might produce metadata in form of semantic annotation, while others may produce new files of arbitrary formats. Nevertheless, the Semantic Assistants server eventually transforms the results into a standalone XML message. The client then, directly or through using the CSAL libraries, parses the XML response and presents the results to its users. Using a uniform XML message structure keeps the implementation open for alternative output formats and allows various output formats to be added at runtime. Figure 2 shows the semantic annotations found by an Information Extraction (IE) service invoked on the text *"Mary lives in Canada."* As can be seen in the response, the IE service found one "Person" (with *gender: female*) and one "Location" entity (with *type: country*).

# 3. DESIGN

Following the description of the application of NLP services in wiki systems and how the Semantic Assistants framework provides such capabilities as web services, we now have to define our research question: Is it possible to create an integration architecture that would allow different wiki engines to benefit from NLP techniques offered by the Semantic Assistants framework?

Such an architecture should enable wiki systems to benefit from NLP techniques, without the need to have a concrete knowledge of their implementation, nor requiring extensive manipulation to their engines. This means that the integration of NLP services must not be hard-coded on the NLP providing system or the wiki engine – rather, an abstraction layer is required between the two that provides a common ground for communication. To this end, first we define a set of abstract requirements for such a Wiki-NLP integration.

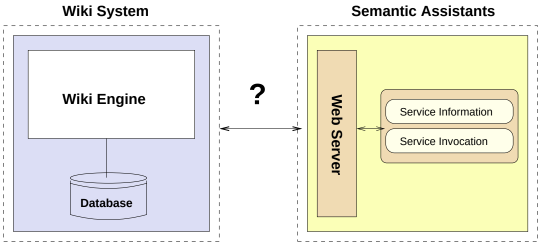


**Figure 3: Towards Wiki-NLP integration**

## 3.1 Requirements

The quick flow of content development and low learning curve associated with wikis are among the factors that have a drastic effect on their adoption as collaborative document management systems. As reviewed in Wikipatterns [4], wikis are often used as an authoring environment for externalizing organizational or personal knowledge. Therefore, wiki users in real-life vary from laypersons to highly-technical employees with various backgrounds, and limited knowledge in the field of software or language engineering or even lack thereof. Having this in mind, we hereby state the requirements of the integration of NLP capabilities inside a wiki environment.

**NLP Service Independence (R1).** NLP services are used in various domains, such as biology, software engineering or e-learning, and have different implementations. While some of these NLP services have direct real-world applications, others can serve as new inputs to larger and more complex tasks. Thus, irrespective of the NLP services' concrete implementation, the integration must offer them within a single unique interface in the wiki.

**Read Content from Wiki (R2).** The system must be able to pull out content from the wiki's database in order to provide the NLP pipelines with input documents. Based on the available wiki capabilities, the integration must be able to retrieve not only the main content of a page, but also its associated metadata, such as revisions, editor information, discussion page content and semantic annotations.

**Write Results to Wiki (R3).** The integration must be able to write the analysis results back to the wiki's database to make them persistent. Also, the integration must be flexible in terms of where it should write the results. For example, users may choose to store the results of content development services embedded in a page's main content, while having the associated metadata generated by NLP services stored in the page's discussion section.

**Seamless Integration (R4).** Employing NLP techniques on wiki content must not largely deviate from the established usage patterns of wikis. This means that NLP capabilities must be integrated within the wiki's user interface that the users are already familiar with.

**Easy Deployment (R5).** In order to benefit from NLP techniques offered by the Wiki-NLP integration, users must not need to apply major changes to the wiki engine or to

[7] OpenNLP, http://opennlp.sourceforge.net

[8] UIMA, https://uima.apache.org/

the means to access the wiki, i.e., their Web browsers. This requirement is derived from the fact that wiki end-users are diverse in their background knowledge and no assumption about their knowledge in software or language engineering should be made.

**Flexible Response Handling (R6).** According to the Semantic Assistants architecture described in Section 2.2, NLP services produce different types of output, such as semantic metadata in form of annotations or new files. The Wiki-NLP integration must be able to accordingly transform, store and present the results to users in a way that is distinguishable from the wiki's original content.

**Collection-based Analysis (R7).** An implicit goal of the Wiki-NLP integration is automating tasks that are currently done manually through the traditional ways that wikis provide. In some instances, a user's information need is scattered across multiple pages in a wiki. Satisfying such needs by hand is a cumbersome and error-prone task. Therefore, users must be able to collect pages of interest and run an NLP service on the collection at once.

**Proactive Service Execution (R8).** The system must be able to perform NLP analysis in a proactive and event-based manner. For example, when the integration is used to create a back-of-the-book index of a wiki content, it should be able to perform an automatic index generation every time a change is applied to the wiki content.

## 3.2 Developed Solution

Neither the wiki system nor the Semantic Assistants architecture can solely fulfil all of the requirements we postulated in the previous section. Therefore, we have to design an architecture that would allow the two systems to communicate with each other while keeping the modifications on the wiki engine to a minimum, in order to increase the integration acceptability. Our Wiki-NLP integration solution is thus a collaborative approach, combining the power of a light-weight MediaWiki extension and a novel server-side wiki component in the Semantic Assistants framework. The main idea in this approach, as shown in Figure 4, is to divide the responsibilities between the two systems based on their capabilities and provide them with a common ground for communication. The MediaWiki extension bears the responsibility of wiki-specific tasks, such as patrolling wiki content changes, while the server-side wiki component encompasses functionalities that are needed to employ NLP capabilities within a wiki system, such as presenting the user interface or handling service invocation requests.

### *3.2.1 The Server-side Wiki Component*

The server-side wiki component acts as an intermediator between the Semantic Assistants server, the MediaWiki engine and the user's browser. It provides a centralized entry point for handling service requests. In other words, all the requests from the user's browser are sent to this component and it will in turn provide the suppliant, e.g., the browser, with the outcome of the demanded action, such as the capability to inquire about and invoke available NLP services. The wiki component has four main responsibilities:

***I. Pre-processing of requests.*** Every service request received by the wiki component is first validated before being dispatched to the business logic. Here, the wiki component checks whether the wiki engine sending the request is supported by the integration and then validates the user credentials and the request parameters for its completeness. Pre-processing the requests provides a preemptive behaviour against malicious or faulty requests to be sent to the Semantic Assistants server.

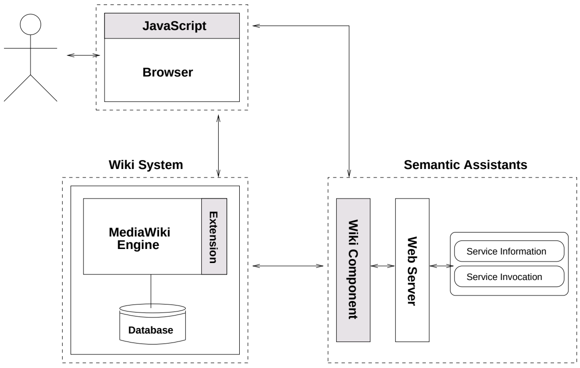


**Figure 4: Wiki-NLP integration – High-level design**

***II. Dispatching requests to the business logic.*** Following the pre-processing phase, the wiki component dispatches the request to the business logic that is responsible for executing the user request. The business logic is the unit that transforms the user request to a concrete service call to the Semantic Assistants server interface. The wiki component is then responsible for gathering the results from the server and prepares them for the client by generating the corresponding wiki markup.

***III. Controlling the display flow.*** Based on the status of the pre-processing phase or business logic execution, the wiki component maps the request to a chosen logical display. For example, based on the user request status, the wiki component decides whether to present the user with a login page, the integration user interface or delegate the request to the business logic and update the wiki page with the results. Also, if an exception occurs during the service execution, the component stores the exception in the request object and forwards the display to a web page, providing the user with detailed information about the exception.

***IV. Maintaining the ontology model.*** As we will describe in the following section, wiki instances are introduced to the integration through their ontologies. The wiki component is responsible for keeping an in-memory model of the available wiki ontologies by parsing their formal descriptions. When the model is formed in the wiki component, it can then be used at runtime to reason about each wiki's capabilities. For example, a typical query to the ontology model can be *"What are the namespaces in this wiki engine?"* or *"What file formats does the wiki engine allow to be uploaded to the database?"*. The ontology can also be used to populate the user interface of the integration and validate user requests against an underlying wiki engine.

### *3.2.2 The Client-side Wiki Extension*

As we stated earlier, the main purpose of designing a MediaWiki extension is to provide functionalities that cannot be offered through the wiki component alone, such as the proactive service execution described in Requirement R8. Since the extension is installed on the wiki and has direct access to its database, it can patrol content changes in the wiki and create dynamic service requests to the server-side

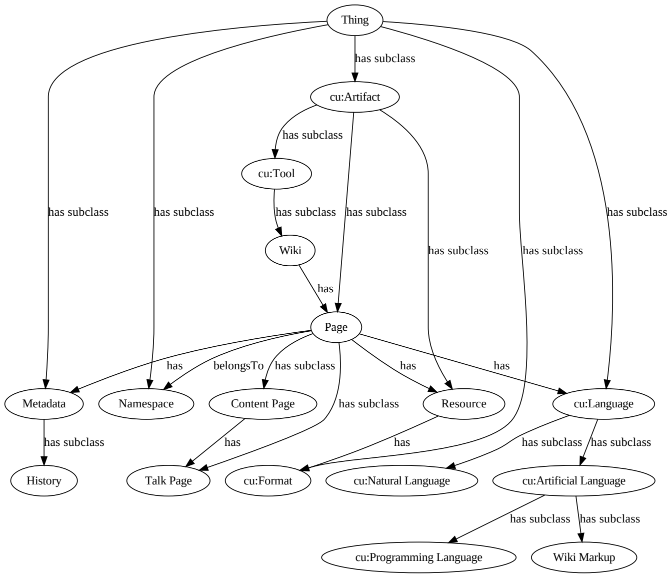


**Figure 5: Wiki Upper Ontology**

wiki component to analyze the new content or flag the already existing results as outdated when the original content of a page changes. Also, installing an extension on MediaWiki is a fairly simple task that is done with one or two lines of code, which makes the deployment of the NLP integration easier than requiring individuals to install an application on their machines or adding a plug-in to their browsers (R5).

## 3.3 A Semantic Wiki Meta-Model

We also need to make a design decision about how different MediaWiki instances can work with a single unique interface. Although all instances are based on the same core engine, they still may be of different versions or have different capabilities. For example, a MediaWiki engine that has the Semantic MediaWiki extension[9] installed can consume and export semantic metadata in form of RDF triples, while a traditional MediaWiki instance cannot. Towards this end, we adopted a semantics-based approach, in which different wiki engines are introduced to the integration architecture through their *ontology* files. By using OWL as the ontology language to formally describe a wiki, the integration does not need to know about their concrete implementation; rather it uses automatic reasoning on their ontologies to discover each wiki's structure and capabilities. To facilitate the process of ontology definition, we have designed a generic *upper ontology* for wiki systems shown in Figure 5, which also includes concepts defined in the Semantic Assistants upper ontology [8] – a multi-purpose ontology that describes five core concepts to model the relationships between users, their tasks, the artifacts involved and their format and language.

The wiki upper ontology is designed using Protégé[10] and reflects the concepts that are common to all wiki engines. Thus, for the MediaWiki engine, we merely have to instantiate the upper ontology manually or automatically using special scripts to define the concrete structure of each wiki instance, e.g., its available namespaces. Table 1 summarizes the concepts of the wiki upper ontology.

**Table 1: Concepts in the wiki upper ontology**

| Concept | Description | Example |
|---|---|---|
| Wiki | *wiki engine* | MediaWiki |
| Page | *Wiki elements encompassing textual content* | "Semantic Web" |
| Namespace | *Category names to differentiate pages at a high level* | "Help", "Project" |
| Resource | *Files with arbitrary formats* | Picture.jpg |
| Metadata | *Metadata associated with wiki pages* | History, Semantic Annotations |
| Wiki Markup | *Ontological representation of wiki syntax* | MediaWiki Markup |

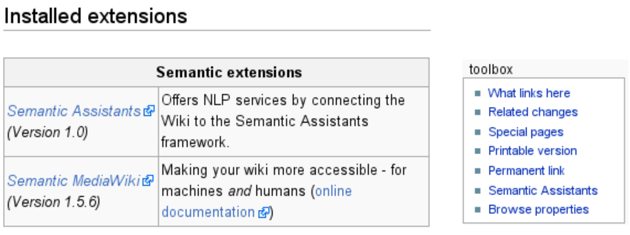
## 4. IMPLEMENTATION

We now describe how the above design is transformed into the complete implementation architecture as shown in Figure 6.

### 4.1 Semantic Assistants MediaWiki Extension

The Semantic Assistants MediaWiki extension is a lightweight extension written in PHP that modifies the wiki's native navigational menu. As shown in Figure 7, the body of the extension consists of only a few lines of code that adds a new menu item to the wiki, which provides access to the Semantic Assistants NLP user interface (R4).

```
<?php
function wfToolboxLink(&$monobook) {
 # Create a link in the menu pointing to the Wiki−NLP servlet
 print ("<li> <a href=\"http://server.example.com:8080/Wiki-NLP/SemAssistServlet?action=proxy\">Semantic Assistants</a></li>");
 return true;
}
?>
```



Installed extensions

| Semantic extensions | |
|---|---|
| Semantic Assistants<br>(Version 1.0) | Offers NLP services by connecting the Wiki to the Semantic Assistants framework. |
| Semantic MediaWiki<br>(Version 1.5.6) | Making your wiki more accessible - for machines *and* humans (online documentation) |

**Figure 7: Semantic Assistants MediaWiki extension source code and resulting view**

The extension also imports six MediaWiki templates[11] to the wiki that are used to customize the presentation of NLP results in wiki pages. Templates in MediaWiki are standard wiki pages whose content is designed to be embedded inside other pages. MediaWiki allows various parameters to be passed to a template when they are transcluded in a page to produce dynamic contents or different behaviour. Upon a successful NLP service execution, the templates are embedded in the selected wiki page and populated from the retrieved response (R6). This way, while the data model of the results is decided by the wiki component's business logic,

[9] Semantic MediaWiki extension, http://www.mediawiki.org/wiki/Extension:Semantic_MediaWiki

[10] Protégé, http://protege.stanford.edu/

[11] MediaWiki Templates, http://www.mediawiki.org/wiki/Help:Templates

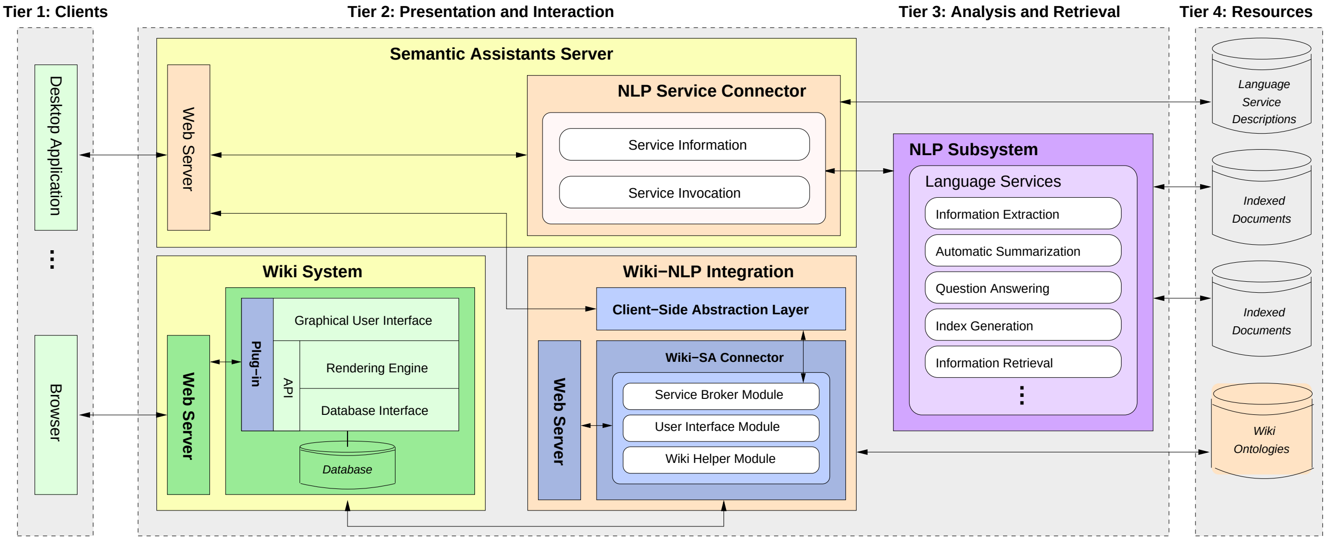


**Figure 6: The new Wiki-NLP integration module in the Semantic Assistants architecture**

e.g., *semantic annotations*, their presentation style can be modified on the wiki without requiring any changes to the integration architecture.

## 4.2 The Wiki-SA Connector

The Wiki-SA Connector component, shown in Figure 6, is technically an HTTP proxy sever written using the Java Servlet[12] technology. As mentioned earlier, it acts as an intermediator between the Semantic Assistants server, the wiki system and the user's browser by intercepting their communication. For each of these three endpoints, there exists a module in the servlet, specifically concerned with the endpoint's business logic. This way, having separate modules allows the sub-components to evolve and extend independently.

### *Service Broker Module.*

The service broker module is the connecting point of the integration to the Semantic Assistants server. Every service execution request that is received by the integration component is translated into a Java method call in this module, which in turn triggers the execution of one or multiple NLP services in the Semantic Assistants server.

### *User Interface Module.*

This module is responsible for generating the integration user interface within a wiki environment. Since wikis are accessible through Web browsers, this module is designed to generate an HTML representation of the Semantic Assistants user interface and inject it to the the user's browser using JavaScript. Figure 8 shows how the generated user interface is added to a wiki page to give its users the impression that they are still interacting with the wiki's native interface. Through this user interface, users can find the *Available Assistants* (bottom left) and invoke arbitrary NLP services by dynamically querying the Semantic Assistants repository of service descriptions. This way, any language service that is offered by a Semantic Assistants server is presented in the user interface to the user (R1). Moreover, the generated user interface allows users to combine multiple pages of the wiki in a *collection*, i.e., a list of user-selected page URLs, and invoke the NLP service on them at once (R7).

### *Wiki Helper Module.*

The wiki helper module encompasses the classes required for communicating with the MediaWiki engine. The classes in this module are responsible for providing the NLP pipelines with input data by reading wiki pages from the database (R2) and eventually transforming the results to their corresponding template markup and storing them in the wiki database (R3).

We stated in Section 3.2.1 that the Wiki-SA Connector is responsible for maintaining and reasoning on the available wiki ontologies. This process is achieved by special ontology keeper classes that upon each servlet bootstrapping, run over the wiki repository OWL files and create an in-memory model of the wikis, by parsing them using Protégé's OWL libraries. This module also provide reasoning capabilities on wiki ontologies using the SPARQL[13] language.

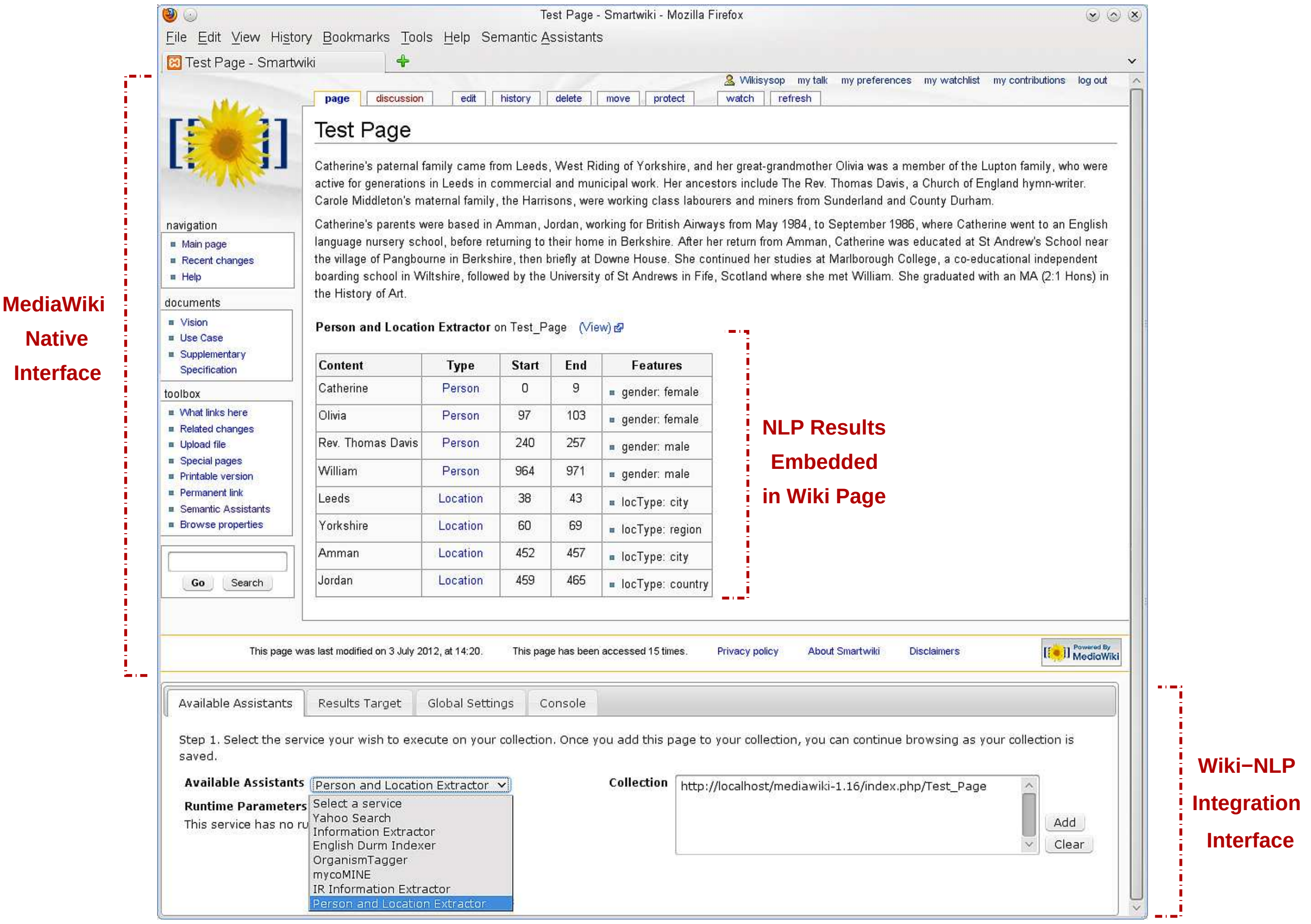
## 4.3 Storing and Presenting NLP Results

The ultimate goal of our Wiki-NLP integration is to create a "self-aware" wiki that can develop and organize its content. Therefore, unless the results from NLP services are presented to users or become persistent in the wiki, the integration would not add any valuable advancement to the current state of the underlying wiki system. Transforming the NLP service results to wiki markup is a task handled by special parser classes in the wiki helper module described in the previous section.

Following a successful NLP service execution, results are passed from the Semantic Assistants server to the service broker module in form of an XML message, as shown in the example in Figure 2. The broker module then interprets the server's XML response and parses the message into an

[12] Java Servlet API, http://download.oracle.com/docs/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/

[13] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/


Test Page - Smartwiki - Mozilla Firefox
File Edit View History Bookmarks Tools Help Semantic Assistants
Test Page - Smartwiki
Wikisysop my talk my preferences my watchlist my contributions log out
page discussion edit history delete move protect watch refresh
navigation
Main page
Recent changes
Help
documents
Vision
Use Case
Supplementary Specification
toolbox
What links here
Related changes
Upload file
Special pages
Printable version
Permanent link
Semantic Assistants
Browse properties
Go Search
Privacy policy About Smartwiki Disclaimers
Powered By MediaWiki
Available Assistants Results Target Global Settings Console
Select a service
Yahoo Search
Information Extractor
English Durm Indexer
OrganismTagger
mycoMINE
IR Information Extractor
Person and Location Extractor
Add
Clear
MediaWiki Native Interface
NLP Results Embedded in Wiki Page
Wiki–NLP Integration Interface

Test Page

Catherine's paternal family came from Leeds, West Riding of Yorkshire, and her great-grandmother Olivia was a member of the Lupton family, who were active for generations in Leeds in commercial and municipal work. Her ancestors include The Rev. Thomas Davis, a Church of England hymn-writer. Carole Middleton's maternal family, the Harrisons, were working class labourers and miners from Sunderland and County Durham.

Catherine's parents were based in Amman, Jordan, working for British Airways from May 1984, to September 1986, where Catherine went to an English language nursery school, before returning to their home in Berkshire. After her return from Amman, Catherine was educated at St Andrew's School near the village of Pangbourne in Berkshire, then briefly at Downe House. She continued her studies at Marlborough College, a co-educational independent boarding school in Wiltshire, followed by the University of St Andrews in Fife, Scotland where she met William. She graduated with an MA (2:1 Hons) in the History of Art.

**Person and Location Extractor** on Test_Page (View)

| Content | Type | Start | End | Features |
|---|---|---|---|---|
| Catherine | Person | 0 | 9 | gender: female |
| Olivia | Person | 97 | 103 | gender: female |
| Rev. Thomas Davis | Person | 240 | 257 | gender: male |
| William | Person | 964 | 971 | gender: male |
| Leeds | Location | 38 | 43 | locType: city |
| Yorkshire | Location | 60 | 69 | locType: region |
| Amman | Location | 452 | 457 | locType: city |
| Jordan | Location | 459 | 465 | locType: country |

This page was last modified on 3 July 2012, at 14:20. This page has been accessed 15 times.

Step 1. Select the service your wish to execute on your collection. Once you add this page to your collection, you can continue browsing as your collection is saved.

**Available Assistants** Person and Location Extractor

**Runtime Parameters**
This service has no ru

**Collection** http://localhost/mediawiki-1.16/index.php/Test_Page

**Figure 8: The Wiki-NLP integration user interface embedded in a wiki page**

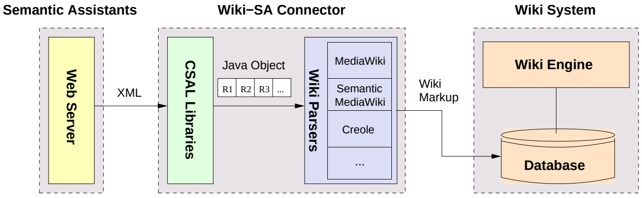


**Figure 9: Transforming NLP service results to MediaWiki markup**

array of Java objects. The service result objects are then transformed to MediaWiki markup by the wiki helper classes and prepared for the *templating mechanism*.

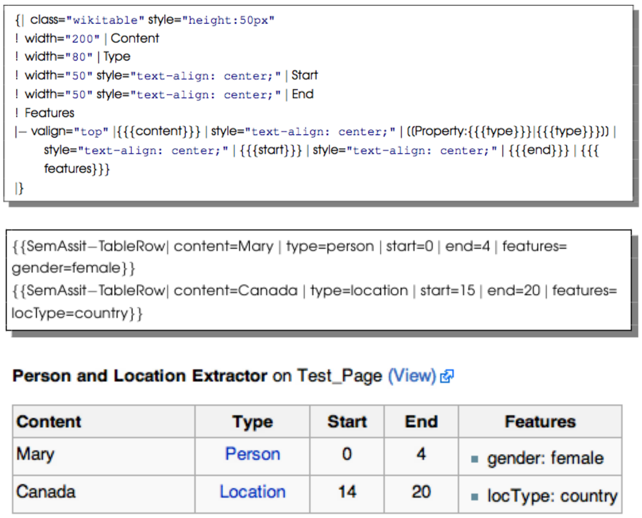
The templating mechanism, illustrated in Figure 10, is the process of embedding service results into MediaWiki templates for presentation. This mechanism separates the data model from its presentation and provides the opportunity to create multiple views for a single model for different purposes. Templating is a collaborative task performed by the Semantic Assistants Wiki-NLP component and the wiki extension. The wiki helper module prepares the markup by placing the results within their corresponding templates and storing them in the wiki's database. Once the wiki page is viewed by the user, the templates installed on the wiki by the Semantic Assistants extension will render the template markup to generate appropriate HTML representations for NLP results, such as tables or lists. Figure 9 shows the workflow of the transformation of server XML response messages to MediaWiki markup. As shown in the picture, the wiki helper module has a modular structure and uses a Factory Pattern [2] to generate wiki markup, meaning that the format of the results can be determined at runtime. This way, different parsers can be added to the integration that will produce markup for MediaWiki, Semantic MediaWiki or Creole.[14]

The next important task is to store the results in the wiki, so that they become persistent and can be used later on. This is important because, according to Requirement R8, the service execution can occur proactively and does not necessarily require user interaction. In such cases, service results, e.g., new generated content, has to be stored in the wiki so that users can access them later.

For this purpose, the classes in the wiki helper module can use the MediaWiki API[15] or rely on third party libraries to connect to the wiki database. Currently, our Wiki-NLP

[14] Creole, http://www.wikicreole.org/

[15] MediaWiki API, http://www.mediawiki.org/wiki/API:Client_code



```
{| class="wikitable" style="height:50px"
! width="200" | Content
! width="80" | Type
! width="50" style="text-align: center;" | Start
! width="50" style="text-align: center;" | End
! Features
|– valign="top" |{{{content}}} | style="text-align: center;" | ((Property:{{{type}}}|{{{type}}})) |
     style="text-align: center;" | {{{start}}} | style="text-align: center;" | {{{end}}} | {{{
     features}}}
|}
```

```
{{SemAssit–TableRow| content=Mary | type=person | start=0 | end=4 | features=
gender=female}}
{{SemAssit–TableRow| content=Canada | type=location | start=15 | end=20 | features=
locType=country}}
```

**Person and Location Extractor** on Test_Page (View)

| Content | Type | Start | End | Features |
|---|---|---|---|---|
| Mary | Person | 0 | 4 | ▪ gender: female |
| Canada | Location | 14 | 20 | ▪ locType: country |

**Figure 10: Templating mechanism for semantic annotations**

integration uses the Java Wiki Bot Framework[16] (JWBF), an open source framework for MediaWiki that allows the integration to connect to any MediaWiki engine, read wiki page information and modify their content.

## 4.4 Wiki-NLP Interaction Example

After an exhaustive description of the Wiki-NLP integration, in this section we illustrate an example scenario of how a wiki user interacts with the Wiki-NLP integration. The goal of our user in this scenario is to find the named entities in a wiki page as shown in Figure 8. The communication flow of this scenario is illustrated in Figure 11.

First, our user invokes the integration user interface by clicking on the 'Semantic Assistants' menu item (Figure 8, 'toolbox' menu), which sends a request to the integration servlet configured in the extension file (Figure 7). Once the request is processed, the user interface depicted in Figure 8 is presented to the user that allows him to see a list of available NLP services in the Semantic Assistants server and add the wiki page URL to the collection. Next, the user invokes the "Person and Location Extractor" service, which can extract named entities from a text and provide additional information, such as gender of a person or type of a location.

The service invocation results in sending an HTTP request to the integration servlet that contains the name of the selected service and the name of the wiki page. Once the request is validated by the servlet, it is then prepared for the service execution. In order to send the content of the wiki page to the designated pipeline, the servlet retrieves the list of page URLs from the collection and asks the wiki helper module to use the JWBF bot to read the content of each page from the wiki's database. The content is then cleaned up, removing noise, such as wiki markup and comments, and returned to the servlet. Using the service broker module, the servlet then makes a service execution call to the Semantic Assistants server by defining the name of the service as the "Person and Location Extractor" and the input as the wiki pages' content. Once the NLP service is finished with the analysis, the Semantic Assistants server generates an XML output, similar to the one shown in Figure 2, that contains the detected annotations as the extracted named entities, their exact character offsets in the text, as well as any other additional information found by the pipeline. The recipient of this XML message is the servlet, which parses the XML message into Java objects and sends them to the wiki helper module to be transformed to MediaWiki markup. The parser classes in the wiki helper module then place the extracted annotations in the Semantic Assistants annotation template (Figure 10) and let the JWBF bot write them back to the wiki. The servlet now embeds a *completion of service* message inside an HTTP response object and sends it to the browser to be displayed to the user. Once the user refreshes the wiki page, he can see the newly generated table that contains the Person and Location Extractor service results (Figure 8).

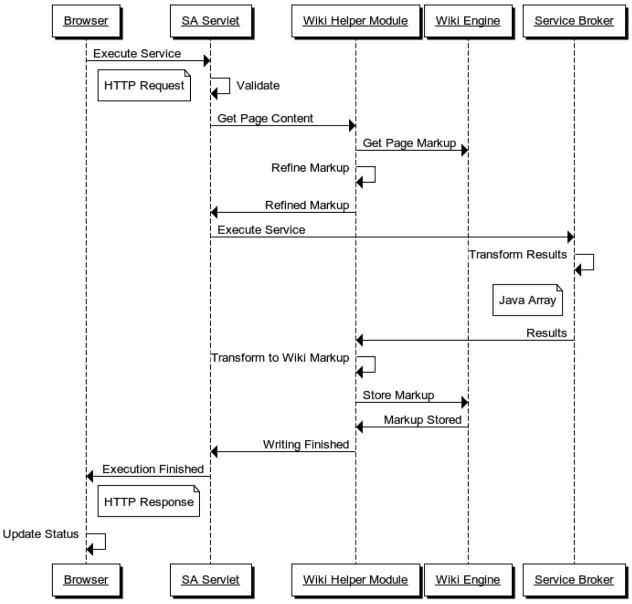


**Figure 11: Communication flow in the Wiki-SA connector component**

## 5. APPLICATIONS

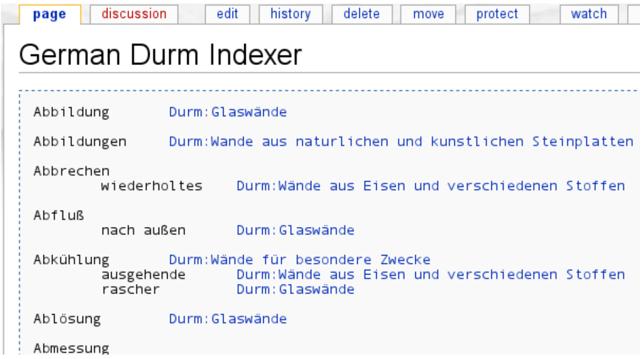
In this section, we describe how we used the Wiki-NLP integration in a number of real-world projects.

## 5.1 Cultural Heritage Data Management

Cultural heritage data is the legacy of artifacts of a society inherited from the past, such as books. To preserve these artifacts, they are often digitized and stored in nationally and internationally distributed databases. We previously developed a custom wiki integration for this scenario, the *Durm Wiki*,[17] where we demonstrated how modern semantic technologies offer the means to make these heritage documents accessible by transforming them into a semantic knowledge base [9]. Now we are able to offer the same NLP capabilities that had been implemented in the Durm wiki using our generic Wiki-NLP integration architecture, in addition to all the other services made available through the general Semantic Assistants framework.

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

[17] Durm Wiki, http://durm.semanticsoftware.info/wiki



German Durm Indexer

```
Abbildung         Durm:Glaswände

Abbildungen       Durm:Wande aus naturlichen und kunstlichen Steinplatten

Abbrechen
        wiederholtes    Durm:Wände aus Eisen und verschiedenen Stoffen

Abfluß
        nach außen      Durm:Glaswände

Abkühlung         Durm:Wände für besondere Zwecke
        ausgehende      Durm:Wände aus Eisen und verschiedenen Stoffen
        rascher         Durm:Glaswände

Ablösung          Durm:Glaswände

Abmessung
```

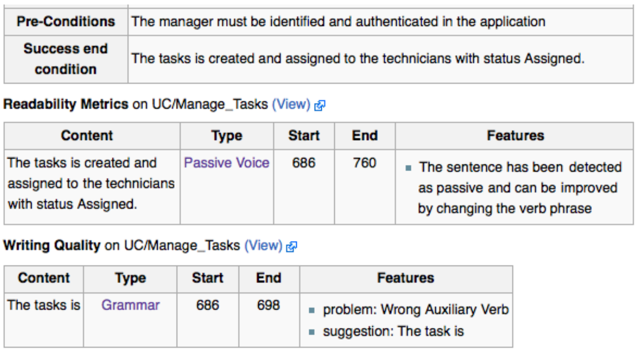
**Figure 12: Automatic back-of-the-book index generation for wiki content**

As an example NLP service, Figure 12 shows the 'back-of-the-book' style index page in the wiki that has been generated by our German Durm Indexer pipeline. The presence of an index page in the wiki that is automatically maintained not only aggregates information on a high level and helps users to find information 'at a glance', but also enables them to "discover" interesting concepts or entities that they did not know were present in the wiki. In our application scenario, a historical encyclopedia of architecture, this index helped domain experts to locate information that they could not find through information retrieval methods: since the 100-year old heritage documents contain outdated terminology no longer in use, full-text search was not sufficient to discover concepts [9]. Additional services provided summaries of wiki content based on a specific question [9]. These ideas can be applied to other heritage data projects and knowledge management wikis in general.

## 5.2 Software Requirements Engineering

Wikis, as an affordable, lightweight documentation and distributed collaboration platform, have demonstrated their capabilities in distributed requirements engineering. Software requirements specifications (SRS) documents, diagrams and images are typically authored in a collaborative manner by various stakeholders and stored in wikis as articles and resources. A requirements specifications document containing precise definitions of what stakeholders want is critical to the design of "the right product" and consequently, the success of a project. To manage SRS, we developed ReqWiki,[18] which combines Semantic MediaWiki forms and templates to structure use case-based requirements engineering, following the Unified Process (UP) methodology. Using the Wiki-NLP integration, users can request Semantic Assistants to help them improve their specifications. Examples include performing automatic quality assurance on the content of the SRS documents in order to find and remove defects, such as *Weak Phrases*, *Options*, and *Passive voice*, since the presence of these defects are usually indicative of the relative ambiguity and incompleteness of an SRS document. Figure 13 shows the results of a SRS quality assurance assistant that has been executed on a Use Case template inside the wiki. The requirements engineer can then investigate the individual defects and fix them by editing the wiki's content.

In a case study with software engineering students, this NLP integration significantly improved the quality of the resulting requirements documents, compared to a wiki without this integration. Furthermore, a usability study showed that users unfamiliar with NLP technology can easily apply the offered Semantic Assistants [5].



| | |
|---|---|
| **Pre-Conditions** | The manager must be identified and authenticated in the application |
| **Success end condition** | The tasks is created and assigned to the technicians with status Assigned. |

**Readability Metrics** on UC/Manage_Tasks (View)

| Content | Type | Start | End | Features |
|---|---|---|---|---|
| The tasks is created and assigned to the technicians with status Assigned. | Passive Voice | 686 | 760 | ▪ The sentence has been detected as passive and can be improved by changing the verb phrase |

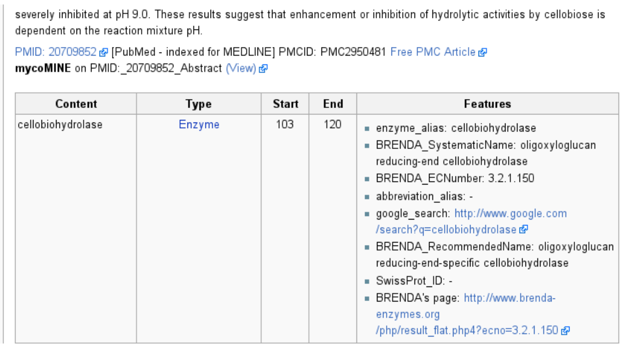
**Writing Quality** on UC/Manage_Tasks (View)

| Content | Type | Start | End | Features |
|---|---|---|---|---|
| The tasks is | Grammar | 686 | 698 | ▪ problem: Wrong Auxiliary Verb<br>▪ suggestion: The task is |

**Figure 13: Quality assurance of SRS documents**

## 5.3 Biomedical Literature Curation

Biomedical literature curation is the process of manually refining and updating bioinformatics databases. The data for curation is generally gathered from the domain literature, e.g., scientific papers, journal articles and domain-specific websites like PubMed[19] and provided to curators – domain experts – who manually browse through the data and extract domain knowledge from the literature. When a wiki system is filled with literature, NLP pipelines can greatly help with the labour-intensive task of curation. Figure 14 shows GenWiki [6], a wiki developed for biocuration in lignocellulose research, where NLP pipelines support domain experts by automatically extracting entities, such as enzymes or organisms, from full-text research papers. It also provides additional information through semantic enrichment, such as the systematic name of an enzyme entity or its corresponding webpage in the BRENDA[20] database.



severely inhibited at pH 9.0. These results suggest that enhancement or inhibition of hydrolytic activities by cellobiose is dependent on the reaction mixture pH.

PMID: 20709852 [PubMed - indexed for MEDLINE] PMCID: PMC2950481 Free PMC Article
**mycoMINE** on PMID:_20709852_Abstract (View)

| Content | Type | Start | End | Features |
|---|---|---|---|---|
| cellobiohydrolase | Enzyme | 103 | 120 | ▪ enzyme_alias: cellobiohydrolase<br>▪ BRENDA_SystematicName: oligoxyloglucan reducing-end cellobiohydrolase<br>▪ BRENDA_ECNumber: 3.2.1.150<br>▪ abbreviation_alias: -<br>▪ google_search: http://www.google.com/search?q=cellobiohydrolase<br>▪ BRENDA_RecommendedName: oligoxyloglucan reducing-end-specific cellobiohydrolase<br>▪ SwissProt_ID: -<br>▪ BRENDA's page: http://www.brenda-enzymes.org/php/result_flat.php4?ecno=3.2.1.150 |

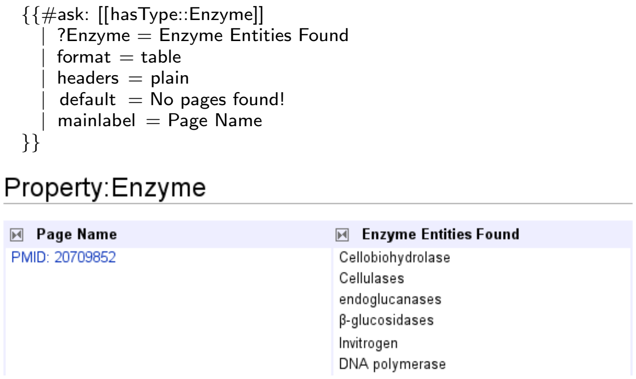
**Figure 14: Text mining of biomedical literature**

In addition to entity detection, the Wiki-NLP integration also populates the wiki with semantic metadata that is generated by the NLP services. For instance, in the above example for each enzyme entity that is found by the pipeline,

[18] ReqWiki, http://www.semanticsoftware.info/reqwiki

[19] PubMed, http://www.ncbi.nlm.nih.gov/pubmed/

[20] BRENDA Enzyme Database, http://www.brenda-enzymes.info/



```
{{#ask: [[hasType::Enzyme]]
   | ?Enzyme = Enzyme Entities Found
   | format = table
   | headers = plain
   | default = No pages found!
   | mainlabel = Page Name
}}
```

Property:Enzyme

| Page Name | Enzyme Entities Found |
|---|---|
| PMID: 20709852 | Cellobiohydrolase<br>Cellulases<br>endoglucanases<br>β-glucosidases<br>Invitrogen<br>DNA polymerase |

**Figure 15: Semantic entity retrieval**

the integration will semantically represent it using Semantic MediaWiki markup `[[hasType::Enzyme]]`, which is internally interpreted as an RDF triple by the Semantic MediaWiki parsers. The generated semantic metadata can then be exploited by users for entity retrieval purposes or exported for external application use. Figure 15 shows how wiki users can query for available wiki pages that contain a specific type of entity, namely Enzymes, and directly navigate to their pages in the wiki.

In a user study with domain experts, this NLP integration reduced the time required to curate research papers in the wiki by almost 20% [5].

## 6. RELATED WORK

*NLP-enhanced wikis* refers to wiki systems that benefit from employing NLP techniques on their content, either provided as an extension to their architecture or tightly integrated in their engine. Such wikis aid their users with content development and management by employing language analytics solutions on the wiki content.

Currently, the only existing NLP-enhanced wiki we are aware of is *Wikulu* [3]. Wikulu proposes an architecture to support users in their tasks of adding, organizing and finding content in a wiki by applying NLP techniques. The major focus of Wikulu is helping users to organize a wiki's content. The authors analyze different types of user interactions corresponding to these tasks and aim to improve the user experience by providing suggestions on *where* to add or *how* to organize content. The NLP integration in Wikulu is implemented as a proxy server that needs to be enabled on the user's browser.

In contrast to Wikulu and our previous work in the Durm project [9], in this paper we offer a general architecture that allows any wiki to benefit from various NLP techniques brokered via the Semantic Assistants framework. In other words, our Wiki-NLP integration is not only NLP service independent, but also offers a flexible and easily extensible design for the integration of various wiki engines with NLP capabilities. Unlike the Durm project, our architecture does not use an external stand-alone application, rather it brings the NLP capabilities *within* the wiki environment. Finally, in our architecture, employing NLP techniques on wiki content does not necessarily require user interaction and can be performed proactively, e.g., on a time or event basis.

## 7. CONCLUSION

Our goal was to develop an architecture for aiding wiki users in time-consuming and labour-intensive tasks, through the help of automatic text mining services. The Wiki-NLP integration architecture we propose provides a wiki-independent user interface that is populated dynamically based on the capabilities of the underlying engine. The integration of wiki engines into the architecture is facilitated through the use of ontologies. This way, we created an extensible architecture that allows more wikis to be added in the future, without the need to change any code in their implementation, allowing both sides to evolve independently. The complete architecture is available under standard open source licenses at http://www.semanticsoftware.info.

By providing a direct and seamless integration of NLP capabilities, we are now able to help wiki users to overcome common problems, such as information overload or poor organization. This way, the organized structure of wikis not only increases their acceptability and usability as a powerful, yet easy-to-use collaborative documentation platform, but also allows their users to focus on their main task in the wiki, rather than spending time on going through the usually massive amount of available unstructured information.

Finally, our Wiki-NLP integration lays the groundwork for a multitude of new projects. More and more wikis are created everyday to support various user needs. This means that, the more wikis are used in various domains, the more NLP services are demanded. Using this architecture, wikis can access NLP techniques that are beneficial to their content. At the same time, more data becomes available for NLP developers to create and train more intelligent services.

## 8. REFERENCES

[1] H. Cunningham et al. *Text Processing with GATE (Version 6)*. University of Sheffield, Dept. of Computer Science, 2011.

[2] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[3] J. Hoffart, T. Zesch, and I. Gurevych. An architecture to support intelligent user interfaces for Wikis by means of Natural Language Processing. In *International Symposium on Wikis (WikiSym '09)*, Orlando, FL, USA, 25–27 Oct. 2009.

[4] S. Mader. *Wikipatterns – A practical guide to improving productivity and collaboration in your organization*. Wiley Publication, 2008.

[5] B. Sateli. A General Architecture to Enhance Wiki Systems with Natural Language Processing Techniques. Master's thesis, Concordia University, Montréal, QC, Canada, 2012.

[6] B. Sateli, C. Murphy, R. Witte, M.-J. Meurs, and A. Tsang. Text Mining Assistants in Wikis for Biocuration. In *5th International Biocuration Conference*, page 126, Washington DC, USA, April 2012. International Society for Biocuration.

[7] 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, New York, NY, USA, 2007. ACM.

[8] R. Witte and T. Gitzinger. Semantic Assistants – User-Centric Natural Language Processing Services for Desktop Clients. In *3rd Asian Semantic Web Conference (ASWC 2008)*, volume 5367 of *LNCS*, pages 360–374. Springer, 2008.

[9] R. Witte, T. Kappler, R. Krestel, and P. C. Lockemann. Integrating Wiki Systems, Natural Language Processing, and Semantic Technologies for Cultural Heritage Data Management. In C. Sporleder, A. van den Bosch, and K. Zervanou, editors, *Language Technology for Cultural Heritage*, pages 213–230. Springer, 2011.