

A GENERAL ARCHITECTURE TO ENHANCE  
WIKI SYSTEMS WITH  
NATURAL LANGUAGE PROCESSING  
TECHNIQUES

BAHAR SATELI

A THESIS  
IN  
THE DEPARTMENT  
OF  
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN  
SOFTWARE ENGINEERING  
CONCORDIA UNIVERSITY  
MONTRÉAL, QUÉBEC, CANADA

APRIL 2012

© BAHAR SATELI, 2012

CONCORDIA UNIVERSITY  
School of Graduate Studies

This is to certify that the thesis prepared

By: **Bahar Sateli**  
Entitled: **A General Architecture to Enhance Wiki Systems  
with Natural Language Processing Techniques**

and submitted in partial fulfillment of the requirements for the degree of

**Master of Applied Science in  
Software Engineering**

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

\_\_\_\_\_ Chair  
Dr. Volker Haarslev

\_\_\_\_\_ Examiner  
Dr. Leila Kosseim

\_\_\_\_\_ Examiner  
Dr. Gregory Butler

\_\_\_\_\_ Supervisor  
Dr. René Witte

Approved \_\_\_\_\_  
Chair of Department or Graduate Program Director

\_\_\_\_\_ 20 \_\_\_\_\_  
Dr. Robin A. L. Drew, Dean  
Faculty of Engineering and Computer Science

# **Abstract**

## **A General Architecture to Enhance Wiki Systems with Natural Language Processing Techniques**

Bahar Sateli

Wikis are web-based software applications that allow users to collaboratively create and edit web page content, through a Web browser using a simplified syntax. The ease-of-use and “open” philosophy of wikis has brought them to the attention of organizations and online communities, leading to a wide-spread adoption as a simple and “quick” way of collaborative knowledge management. However, these characteristics of wiki systems can act as a double-edged sword: When wiki content is not properly structured, it can turn into a “tangle of links”, making navigation, organization and content retrieval difficult for their end-users.

Since wiki content is mostly written in unstructured natural language, we believe that existing state-of-the-art techniques from the Natural Language Processing (NLP) and Semantic Computing domains can help mitigating these common problems when using wikis and improve their users’ experience by introducing new features. The challenge, however, is to find a solution for integrating novel semantic analysis algorithms into the multitude of existing wiki systems, without the need for modifying their engines. In this research work, we present a general architecture that allows wiki systems to benefit from NLP services made available through the Semantic Assistants framework – a service-oriented architecture for brokering NLP pipelines as web services. Our main contributions in this thesis include an analysis of wiki engines, the development of collaboration patterns between wikis and NLP, and the design of a cohesive integration architecture. As a concrete application, we deployed our integration to MediaWiki – the powerful wiki engine behind Wikipedia – to prove its practicability. Finally, we evaluate the usability and efficiency of our integration through a

number of user studies we performed in real-world projects from various domains, including cultural heritage data management, software requirements engineering, and biomedical literature curation.

# Acknowledgments

As Isaac Newton once said, “If I have seen further, it is by standing on the shoulders of giants.” My utmost gratitude goes to my thesis supervisor, Dr. René Witte for his invaluable guidance, endless patience and understanding throughout the course of this thesis. His positive outlook, careful editing and confidence in my research inspired me to do the best of my ability.

I would like to express my appreciation to Dr. Marie-Jean Meurs for offering me her kind support and generous contributions to the evaluation of this thesis. Thank you for always being there for me. I am indebted to you for your kind heart and sisterly love.

I would also like to thank my fellow lab mates at the Semantic Software Lab and the scientists of Concordia Centre for Structural and Functional Genomics for their careful feedback on my research work. I thank Caitlin Murphy and Elian Angius for their time and hard work, as well as tens of Software Engineering students who participated in the evaluation of this thesis.

Last but not least, I cordially thank my parents and my brother for their everlasting love and encouragements. Words fall short to express how grateful I am to feel the warmth of their presence and support at every step of my life. It is to whom this thesis is dedicated.

# Table of Contents

<b>List of Figures</b>	<b>x</b>
<b>List of Tables</b>	<b>xiii</b>
<b>List of Acronyms</b>	<b>xiv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Approach . . . . .	3
1.3 Outline . . . . .	6
<b>2 Background</b>	<b>7</b>
2.1 Wiki Systems . . . . .	7
2.1.1 System Specifications . . . . .	7
2.1.2 Markup Languages . . . . .	9
2.2 Wikis in Practice . . . . .	11
2.2.1 Wikipedia, An Encyclopedia Wiki . . . . .	11
2.2.2 Personal Information and Knowledge Management . . . . .	12
2.2.3 Software Requirements Engineering . . . . .	12
2.2.4 Enterprise Wikis . . . . .	13
2.2.5 Cultural Heritage Data Management . . . . .	14
2.3 Common Problems in Wikis . . . . .	15
2.4 Suggested NLP Solutions . . . . .	17
2.5 The Semantic Assistants Architecture . . . . .	19
2.5.1 System Overview . . . . .	19
2.5.2 System Workflow . . . . .	21

2.5.3	Service Results . . . . .	22
2.6	Résumé . . . . .	23
<b>3</b>	<b>Related Work</b>	<b>24</b>
3.1	NLP-Enhanced Wikis . . . . .	24
3.1.1	Wikulu, An Intelligent User Interface for Wikis . . . . .	24
3.2	Semantic Wikis . . . . .	27
3.2.1	Semantic MediaWiki, A Semantic Extension to MediaWiki . . . . .	27
3.2.2	IkeWiki, A Semantic Wiki for Collaborative Knowledge Management . . . . .	30
3.2.3	SweetWiki, A Semantic Web Enabled Technology Wiki . . . . .	32
3.2.4	AceWiki, A Natural and Expressive Semantic Wiki . . . . .	34
3.3	Discussion . . . . .	36
3.4	Résumé . . . . .	37
<b>4</b>	<b>Requirements Analysis</b>	<b>38</b>
4.1	End-User Requirements . . . . .	39
4.2	Wiki Developer Requirements . . . . .	41
4.3	System Requirements . . . . .	42
4.4	Discussion . . . . .	44
4.5	Résumé . . . . .	46
<b>5</b>	<b>System Design</b>	<b>47</b>
5.1	Design Alternatives . . . . .	47
5.1.1	A Browser Plug-in . . . . .	48
5.1.2	A Wiki Plug-in . . . . .	49
5.1.3	A Semantic Assistants Wiki Component . . . . .	51
5.1.4	A Proxy Server Component . . . . .	52
5.1.5	Summary . . . . .	53
5.2	The Analysis Workflow . . . . .	54
5.2.1	User Interaction . . . . .	55
5.2.2	Service Invocation . . . . .	61
5.2.3	Wiki Communication . . . . .	66
5.3	Transformation of Results . . . . .	73

5.3.1	Semantic Metadata Representation . . . . .	74
5.4	Wiki Independency . . . . .	75
5.4.1	Module-based Architecture . . . . .	75
5.4.2	Semantics-based Architecture . . . . .	75
5.5	Wiki Ontology . . . . .	76
5.6	Developed Solution . . . . .	79
5.7	Résumé . . . . .	81
<b>6</b>	<b>Implementation and Application</b>	<b>83</b>
6.1	System Overview . . . . .	83
6.2	The Semantic Assistants Servlet . . . . .	86
6.2.1	The User Interface Module . . . . .	88
6.2.2	The Wiki Helper Module . . . . .	91
6.2.3	The Semantic Assistants Broker . . . . .	92
6.2.4	The Wiki Ontology Repository . . . . .	93
6.3	The Semantic Assistants Wiki Plug-in . . . . .	94
6.4	Storing and Presenting Service Results . . . . .	96
6.4.1	Templating Mechanism . . . . .	97
6.4.2	Storing the Markup . . . . .	100
6.5	Service Execution Flow . . . . .	102
6.6	Résumé . . . . .	104
<b>7</b>	<b>Evaluation</b>	<b>106</b>
7.1	Methodology . . . . .	106
7.2	Wiki-based Cultural Heritage Data Management . . . . .	108
7.2.1	Evaluation Scenario . . . . .	109
7.2.2	Results . . . . .	110
7.3	Wiki-based Collaborative Software Requirements Engineering	111
7.3.1	Evaluation Scenario . . . . .	113
7.3.2	Results . . . . .	114
7.4	Wiki-based Biomedical Literature Curation . . . . .	118
7.4.1	Evaluation Scenario . . . . .	119
7.4.2	Results . . . . .	122
7.5	Résumé . . . . .	125



<b>8</b>	<b>Conclusions and Future Work</b>	<b>126</b>
8.1	Summary . . . . .	126
8.2	Suggestions for Future Work . . . . .	129
8.3	Conclusion . . . . .	130
	<b>Bibliography</b>	<b>131</b>
<b>A</b>	<b>Wiki Upper Ontology Description</b>	<b>139</b>
<b>B</b>	<b>MediaWiki Ontology Description</b>	<b>142</b>
<b>C</b>	<b>ReqWiki Questionnaire</b>	<b>148</b>
<b>D</b>	<b>Questionnaire Responses</b>	<b>158</b>
D.1	Graduate Students Responses . . . . .	158
D.2	Undergraduate Students Responses . . . . .	173

# List of Figures

1	Wikipedia – a free encyclopedia wiki . . . . .	2
2	A subset of MediaWiki text formatting markup . . . . .	10
3	A MediaWiki page markup importing the FOAF vocabulary . . . . .	10
4	The Semantic Assistants architecture [WG09] . . . . .	20
5	The Semantic Assistants service execution workflow [WG09] . . . . .	21
6	The DTD for Semantic Assistants response messages . . . . .	23
7	The Wikulu system architecture [HZG09] . . . . .	25
8	The Semantic MediaWiki system architecture [KVV06] . . . . .	28
9	Semantic MediaWiki markup and factbox embedded in a wiki page . . . . .	29
10	The IkeWiki system architecture [Sch06] . . . . .	31
11	The SweetWiki system architecture [BGE <sup>+</sup> 08] . . . . .	33
12	The web interface of AceWiki, presenting the logic structure of a sentence . . . . .	35
13	Design alternative using a browser plug-in . . . . .	48
14	Design alternative using a wiki plug-in . . . . .	50
15	Design alternative using a Semantic Assistants wiki component . . . . .	51
16	Design alternative using a proxy server component . . . . .	52
17	The web service description for OpenCalais . . . . .	64
18	A web service call for OpenCalais in SMW+ . . . . .	65
19	Wiki upper ontology graph . . . . .	77
20	Developed solution system architecture . . . . .	80
21	Transforming service results to wiki markup . . . . .	81
22	Wiki-NLP integration merged with the Semantic Assistants architecture . . . . .	84

23	Communication flow between wiki system, wiki web server and the Semantic Assistants servlet . . . . .	86
24	Semantic Assistants user interface generated by the servlet . .	89
25	Semantic Assistants user interface second tab . . . . .	90
26	The proxy JSP page code . . . . .	91
27	JavaScript code for on-the-fly user interface modification . . .	91
28	Java code to instantiate a wiki object . . . . .	92
29	UML class diagram presenting the wiki factory pattern . . . .	93
30	The Semantic Assistants MediaWiki plug-in code . . . . .	95
31	Semantic Assistants plug-in installed on MediaWiki . . . . .	96
32	MediaWiki Semantic Assistants service template markup . . .	98
33	MediaWiki Semantic Assistants Table template markup . . . .	99
34	Semantic Assistants annotations view in MediaWiki . . . . .	99
35	MediaWiki Semantic Assistants Block template markup . . . .	100
36	Semantic Assistants Block preview in MediaWiki, showing a summary generated from a wiki page . . . . .	100
37	Java Wiki Bot Framework used to write content to a wiki page	101
38	RDF representation of semantic metadata generated by Semantic MediaWiki . . . . .	102
39	Communication flow for user interface generation . . . . .	103
40	Communication flow for service invocation . . . . .	105
41	The Durm wiki workflow [WKKL11] . . . . .	109
42	Person and Location Extractor service results in DurmWiki . .	110
43	Output of German Durm Indexer pipeline in DurmWiki . . . .	111
44	A sample form in ReqWiki . . . . .	113
45	Presentation of SRS Defects in ReqWiki . . . . .	115
46	A sample question from the students questionnaire . . . . .	115
47	Feedback statistics from students questionnaire . . . . .	116
48	System usability feedback based on the students NLP knowledge level . . . . .	116
49	Average number of defects found in assignments . . . . .	117
50	Manual curation workflow for biomedical literature . . . . .	119
51	A wiki page containing a full-text paper . . . . .	120

52	GenWiki-assisted curation workflow for biomedical literature .	121
53	Presentation of NLP-generated annotations in GenWiki . . . .	121
54	The Semantic MediaWiki inline query for all enzyme entities in GenWiki . . . . .	124
55	Semantic query results for all enzyme entities in GenWiki, generated by NLP services . . . . .	124

# List of Tables

1	Comparison of wikis against Wiki-NLP integration requirements	46
2	Comparison of user interface alternatives . . . . .	60
3	Comparison of service invocation alternatives . . . . .	66
4	Comparison of wiki communication alternatives . . . . .	72
5	Concepts in wiki upper ontology . . . . .	79
6	List of parameters in HTTP service requests . . . . .	87
7	Mapping of requirements to evaluation scenarios . . . . .	108
8	GenWiki-assisted literature curation time . . . . .	122
9	Curation time of papers with different levels of semantic support . . . . .	123
10	Comparison of Wiki-NLP integration against architecture requirements and similar wikis . . . . .	129

# List of Acronyms

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASP	Active Server Pages
BRENDA	BRaunschweig ENzyme DAtabase
CSAL	Client-Side Abstraction Layer
CSFG	Concordia Centre for Structural and Functional Genomics
CSS	Cascading Style Sheets
CSV	Comma Separated Values
DOI	Digital Object Identifier
DTD	Document Type Definition
FAQ	Frequently Asked Questions
FOAF	Friend Of A Friend
GATE	General Architecture for Text Engineering
GRDDL	Gleaning Resource Descriptions from Dialects of Languages
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
J2EE	Java 2 Enterprise Edition
JAR	JAVA Archive
JAVA-WS	Java Web Service
JSP	Java Server Pages
JVM	Java Virtual Machine
MuNPEX	Multi-lingual Noun Phrase Extractor
NASA	National Aeronautics and Space Administration
NLP	Natural Language Processing
OWL	Web Ontology Language

PHP	Peripheral Hypertext Preprocessor
PMID	PubMed ID
RDF	Resource Description Framework
RE	Requirements Engineering
REST	REpresentational State Transfer
SA	Semantic Assistants
SMW	Semantic MediaWiki
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SRS	Software Requirements Specifications
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WSDL	Web Service Definition Language
WWSD	Wiki Web Service Description
WYSIWYG	What You See Is What You Get
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformations

# Chapter 1

## Introduction

This thesis is concerned with developing a general architecture for integrating Natural Language Processing (NLP) techniques with wiki systems. We envision 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 Web domains. The motivation of this integration is to enable wiki users – novice or expert – to benefit from NLP techniques directly within their wiki environment, by offering a seamless integration of wiki systems with NLP. We propose new human/AI collaboration patterns that mitigate common problems when using wikis and support end-users dealing with massive and usually unstructured information. Within this chapter, we first motivate our efforts and then provide an overview of the structure of our research work.

### 1.1 Motivation

Wiki systems power websites whose users can collaboratively add, modify or delete their content via a Web browser. They came into existence in 1995, when Ward Cunningham [LC01], the inventor of wikis, was dissatisfied by the conventional word processing applications' features for collaboration. His vision was to develop a relatively simple software that would enable collaborative work on documents that can be published immediately. In the words of Cunningham, a wiki is “the simplest online



database that could possibly work”<sup>1</sup>.

“Wikiwiki” is a Hawaiian term meaning “quick”. It is used for wikis to describe the characteristics of their underlying software, which allows a quick and simple way for collaborative knowledge creation and management. The ease-of-use and “open” philosophy of wiki systems has brought them to the attention of organizations, online communities and schools. One of the most popular examples of a wiki is the *Wikipedia*<sup>2</sup>. Wikipedia is a free, encyclopedia wiki with content provided by volunteers around the world. Anyone visiting this website can edit the wiki pages using his browser and a simplified syntax. Figure 1 shows a sample page from the Wikipedia website.



Figure 1: Wikipedia – a free encyclopedia wiki

The wiki philosophy is based on trusting users to provide structured content and meaningful relations between entities, and therefore imposes

<sup>1</sup>What is Wiki, <http://www.wiki.org/wiki.cgi?WhatIsWiki>

<sup>2</sup>Wikipedia, <http://www.wikipedia.org>

no strict hierarchy on its content. This means that wiki instances are provided with no pre-defined structure, but rather find their own, by dynamically adapting to their content. While the flexible structure of wikis facilitates the process of knowledge creation and linking, it also poses a challenge on their usability: As the size of a wiki grows, if content is not properly maintained, i.e., structured and linked, it can gradually turn into a “tangle of links”, making navigation, organization and content retrieval difficult for its users [Buf06]. With no explicit browsing feature built into wiki systems, users can easily miss information they don’t know exists in the wiki. Also, wiki content is mostly maintained by its visitors. Apart from web bots that can detect syntactic problems, such as vandalism<sup>3</sup>, the semantics of wiki content can only be maintained by humans. For example, as seen in Figure 1, an alert box is placed on top of the page by a user, suggesting that the content needs additional citations for verification, as well as reorganization to comply with Wikipedia’s guidelines. This type of content maintenance, when scaled to thousands of pages in a wiki, becomes nearly impossible for human capabilities.

This thesis addresses these challenges by applying techniques from the NLP and Semantic Computing domains to combine wiki content with NLP-derived metadata.

## 1.2 Approach

Natural Language Processing is a branch of computer science that uses Artificial Intelligence techniques to process content written in natural language. One of the applications of NLP is *text mining* – the process of deriving high-quality information from text. This process is facilitated by frameworks, such as the General Architecture for Text Engineering (GATE) [CMB<sup>+</sup>11]. Using these frameworks, sophisticated text mining applications can be developed, not only to derive patterns within the structured data, but also to enhance information management of a system by

---

<sup>3</sup>Vandalism is any addition, removal, or change of content in a deliberate attempt to compromise the integrity of a wiki (here, Wikipedia).

finding content based on meaning and context, using technologies from the Semantic Web domain.

Since wiki content is mostly written in unstructured natural language, we believe that employing NLP techniques, such as text mining algorithms, on the content of wikis can improve both their usability and the quality of their content. In this thesis, we are proposing an architecture to *enhance* wiki systems, rather than developing a new wiki engine. This is because the already established engines, like MediaWiki, are not likely to accept fundamental changes to their structure. One of the biggest challenges for the integration of NLP in wikis is the lack of a common standard wiki structure and syntax. Wiki systems are written in various programming languages. Each wiki system has its own database schema and uses its own proprietary grammar and markup. Therefore, one concrete solution can not be easily provided that would encompass the variety of all the existing wiki engines. Rather, an abstraction layer is needed to hide the complexity of NLP techniques from the point of view of wiki end-users and minimize the dependency on the concrete implementation of the underlying wiki engine.

Our two main research questions are: (1) considering the variety of wiki engines, is it possible to create an integration architecture that would allow wikis engines to benefit from NLP techniques, irrespective of the concrete implementation of their engine and the NLP service itself?; and (2) does the integration of such capabilities into a wiki system actually bring measurable value to its end-users?

To answer these questions, we investigate the use cases of wikis in various domains: From personal use to large-scale enterprise wikis. By studying these use cases, we are able to discover common problems that wiki users face in their daily tasks and match them with solutions from the NLP domain to derive our system requirements. We also describe a number of collaboration patterns between wiki systems and NLP domain techniques. Finally, our contribution is presented as a web-based solution that plays the role of an extension to the Semantic Assistants framework [WG09] – a multi-tier, service-oriented architecture that provides us with the NLP

functionality we need in form of web services.

Augmenting wikis with NLP techniques has not attracted a lot of research attention yet and our work is among the first to demonstrate how NLP techniques can be used in the context of wikis to improve their users' experience in content development, organization and retrieval. We are also the first to perform an extrinsic evaluation of the Wiki-NLP integration in a number of real-world projects that further proves the usability and effectiveness of this approach.

The impact of integrating wikis with NLP techniques suggested in this work is significant: It opens the opportunity of bringing state-of-the-art techniques from the NLP and Semantic Web domain to wiki users, without requiring them to have a 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 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.

When our Wiki-NLP integration is employed, not only can wiki content be developed and organized by NLP techniques, but it also becomes implicitly machine-accessible when semantic metadata is generated by NLP pipelines and stored in the wiki. This way, the sheer volume of information available in wikis can be used by machines, so that they can actively participate in the creation and organization of content.

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 techniques are demanded. Using this architecture, wikis can access NLP techniques that are beneficial to their content. On the other hand, more data is available for NLP pipeline developers to create and train more intelligent algorithms and NLP techniques.

## 1.3 Outline

In this chapter, we explained the motivation for employing NLP techniques on wiki systems and briefly described our approach towards this end. The remainder of this thesis is structured as follows:

In Chapter 2, we cover the foundations related to this research. We describe wiki systems and the underlying NLP analysis framework, which our integration is based on.

Chapter 3 covers related work, where we describe similar attempts to perform NLP analysis on wiki content, as well as a number of existing semantic wiki systems.

In Chapter 4, we introduce three target user groups for the Wiki-NLP integration and analyze their requirements in detail.

Various design alternatives for the Wiki-NLP integration are studied in Chapter 5. There, we begin by examining each design alternative against the requirements described in the previous chapter and then combine them to derive a concrete system architecture.

Chapter 6 provides an overview of the entire system and the description of the implementation of each of its components.

The evaluation of our system is covered in Chapter 7, where we examine our system along four dimensions, namely, its practicability, usability, effectiveness and efficiency.

Finally, a summary of this research work and possible future developments are discussed in Chapter 8.

# Chapter 2

## Background

In this chapter, we discuss the foundations related to our work. In particular, we describe wiki systems and illustrate a number of their use cases in different domains. Then, we give a brief introduction of the system that will provide us with the NLP services that we ultimately want to execute on wiki content.

### 2.1 Wiki Systems

Since the very first wiki developed by Ward Cunningham in 1995 [LC01], hundreds of wiki systems have been developed to serve different purposes<sup>1</sup>. However, due to a lack of precise and detailed definition of wiki systems, they are often being confused with other more common and established web-based applications, such as Content Management Systems, Weblogs ('Blogs') or discussion forums. Therefore, in order to distinguish wikis from other web-based document collections and communication tools, let us first clarify what we mean by a "wiki system".

#### 2.1.1 System Specifications

A *wiki system*, as used in the context of this research, refers to a web-based software application that allows users to collaboratively create and

---

<sup>1</sup>See [http://en.wikipedia.org/wiki/List\\_of\\_wiki\\_software](http://en.wikipedia.org/wiki/List_of_wiki_software) and <http://c2.com/cgi/wiki?WikiEngines>

edit web page content [EGHW08]. Content development is carried out via a Web browser by using a simple text syntax to create cross-links between internal pages on-the-fly. Content of wikis is composed of textual data, formatted with a wiki's special markup. It is stored as *articles* or *wiki pages*<sup>2</sup>, along with *resources* in arbitrary file formats, such as images, in the wiki's database or file system. The followings are some additional characteristics that differentiate wikis from other web-based document collections:

- Wiki content, such as web pages, page-related information and other corresponding data is stored in a central, shared repository accessible through the wiki interface.
- Authoring and editing wiki content is carried out via a simple browser interface and does not require knowledge of any web programming languages nor the possession of special tools.
- Wiki pages are uniquely identified by titles, typically noun phrases, with enough precision to avoid name clashes.
- Anyone is able to read and edit wiki page content. Additional permissions, such as adding or deleting pages, can be optionally granted by a wiki system administrator to users.
- A wiki system evolves incrementally. This means that there is no explicit option to create new pages, rather pages are created when a link pointing to them exists in the system.
- Activities and changes within a wiki system and its content are stored in the wiki database and can be viewed by any visitor to the website.
- A wiki system is able to refer to or restore a previous version of a wiki page. Therefore, each wiki page has a corresponding “history” page, where the modification history of its content, as well as the user information of its editors are listed chronologically.
- A wiki system provides a way for its users to search for wiki pages using page titles, as well as full-text keyword search.

---

<sup>2</sup>used interchangeably in this thesis



- A wiki system provides space for users to discuss changes to articles. This characteristic is intended to avoid “edit wars”, where users repeatedly undo or revert the prior user’s edits in an attempt to make their own preferred version of a page visible<sup>3</sup>.

Optionally, wiki systems may offer an extensible architecture, where additional features, such as special markup parsers or semantic reasoners, can be introduced to the wiki’s core functionality via *extensions* or *plugins*. When semantic capabilities are introduced to a wiki system, it is then known as a *semantic wiki*.

### 2.1.2 Markup Languages

*Wiki markup*, also known as *WikiText*, is a lightweight markup language used in wiki pages to inform the wiki engine how to display, categorize and process an article. Wiki markup languages are simplified alternatives to HTML for wiki users. They consist of special characters like asterisks, single quotes and equal signs, which relate to special functions in the wiki engine. For instance, as depicted in Figure 2, for MediaWiki users to emphasize a word in an article, they simply have to place the word in between a pair of two single quotes. When the page is requested by a user, the wiki engine interprets the markup and transforms the special characters, i.e., the single quotes, to the equivalent HTML `<i></i>` tags, which in turn tell the browser to show the word in italic format.

In addition to text formatting markup, semantic wikis also offer semantic markup. Semantic markup allows users to make formal descriptions of resources by annotating the pages representing them. Where a regular wiki enables users to describe resources in natural language, a semantic wiki provides the possibility to additionally describe them in a formal language [OBD06]. Semantic markup is special WikiText, added to an article’s original content, to make it accessible to machines for interpretation. The wiki engine uses the semantic markup to semantically organize its content, improve the navigation using the annotated relations, and provide users

---

<sup>3</sup>Edit war, [http://en.wikipedia.org/wiki/Edit\\_war](http://en.wikipedia.org/wiki/Edit_war)



Description	Wiki Markup	Presentation
Italic Text	" <i>italic</i> "	<i>italic</i>
Bold Text	" <b>bold</b> "	<b>bold</b>
Bullet List	* Start each line with an asterisk ** More asterisks gives deeper *** and deeper levels	<ul style="list-style-type: none"> <li>• Start each line with an asterisk <ul style="list-style-type: none"> <li>◦ More asterisks gives deeper <ul style="list-style-type: none"> <li>▪ and deeper levels</li> </ul> </li> </ul> </li> </ul>
Headings	== Level 2 == === Level 3 === ==== Level 4 =====	<b>Level 2</b> <b>Level 3</b> <b>Level 4</b>

Figure 2: A subset of MediaWiki text formatting markup

with the ability to query the annotations directly, create views from such queries and introduce background knowledge to the system [OBD06]. Figure 3 shows how the FOAF<sup>4</sup> vocabulary can be imported to a Semantic MediaWiki instance by creating a “magic” page<sup>5</sup>.

```

http://xmlns.com/foaf/0.1/[http://www.foaf-project.org/ Friend Of A Friend]
name| Type:String
homepage| Type:URI
Person| Category
knows| Type:Page
...

```

Figure 3: A MediaWiki page markup importing the FOAF vocabulary

In the International Symposium on Wikis in 2006<sup>6</sup>, an attempt was made to standardize the wiki markup language by a group of wiki developers, including Ward Cunningham, the inventor of wikis. Creole<sup>7</sup> is the result of the wiki markup standardization workshop that suggests a “common wiki markup language to be used across different Wikis”. Creole was specified by comparing major wiki engines’ markups and deciding on the

<sup>4</sup>The Friend of a Friend (FOAF) project, <http://www.foaf-project.org/>

<sup>5</sup>A magic page in MediaWiki belongs to the Mediawiki namespace and has the prefix “smw\_import\_”.

<sup>6</sup>WikiSym 2006, <http://www.wikisym.org/ws2006/>

<sup>7</sup>Creole, <http://www.wikicreole.org/>

most common choice for a particular WikiText element according to the goals and good practices formulated by a broad variety of wiki developers and users. Creole version 1.0 was finally released in 2007, but only a handful of different wiki engines adopted the suggested syntax. Consequently, at the time of this writing, no widely accepted standard wiki markup language exists and different wiki systems use their own proprietary grammar, structure, justification and keywords.

## 2.2 Wikis in Practice

Having defined the essential foundation of wiki systems, in this section we describe some use cases of wikis in various domains.

### 2.2.1 Wikipedia, An Encyclopedia Wiki

When talking about wikis, Wikipedia [Bro08] is probably the first use case of wikis that comes to mind, due to its popularity on the Internet. Wikipedia is a free, collaborative and multilingual encyclopedia supported by the non-profit Wikimedia Foundation<sup>8</sup>. It formally began in January 2001, as a “project to produce a free content encyclopedia to which anyone can contribute” [Bro08]. After a decade, it now ranks as the 6th website in the world wide web, serving 470 million people every month with billions of page views<sup>9</sup>. Wikipedia uses MediaWiki as its underlying wiki engine and contains the currently available knowledge on various subjects in its individual pages. Each visitor to the Wikipedia website can improve the existing content or add new pages to the wiki, provided that he has a reference to a published source, known as a *citation*, to verify the modified content. Using their browsers, users can add text, images and multimedia objects to wiki pages and dynamically link related topics together. Although Walters debates in [Wat07] that Wikipedia is not an acceptable citation, but

---

<sup>8</sup>Wikimedia Foundation, <http://wikimediafoundation.org/>

<sup>9</sup>According to Alexa statistics <http://www.alexa.com/siteinfo/wikipedia.org> – Retrieved on 2011-12-20

rather a means to lead one to a citable source, it is nonetheless popular as the secondary source for additional readings and in press citations [Lih04].

### **2.2.2 Personal Information and Knowledge Management**

The goal of personal knowledge management is to make knowledge workers better at capturing, sharing and using knowledge and maximizing their personal effectiveness [KBD<sup>+</sup>09]. Because of wikis' ease-of-use and flexible structure, they have become a popular authoring environment used for externalizing personal knowledge. Wikis can serve the role of a knowledge repository, where a person can use the wiki as an "idea bucket" to store his ideas, cross-link pages to other existing knowledge inside the wiki and collect related external knowledge around them.

Wikis are also used for personal information management. Everyone nowadays is dealing with enormous amount of information stored on their personal computers or the cloud. In addition to their local information, they store blog entries, journals and e-books. Using wikis, a person can consolidate all of his information into a single "trusted system" stored on his local machine and cross-link the content to their related resources. Optionally, they can also publish it on the web, for instance, as an e-portfolio.

### **2.2.3 Software Requirements Engineering**

Software requirements engineering encompasses the tasks related to capturing, determining and recording the needs of various stakeholders of a software project. 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. A wide range of tools are being used for requirements engineering purposes. These tools vary from conventional office suites to dedicated commercial tools, each bearing their own pros and cons [DRR<sup>+</sup>07]. What seems to be common among all non-proprietary requirements engineering tools is the absence of documents integrity and proper communication between stakeholders that

usually results in a chaotic situation. On the other hand, proprietary requirements engineering tools are either too complicated for non-technical stakeholders to use or come with licensing costs. These complications become even more intensified in large projects, where a significant number of stakeholders, who are usually spatially and temporally separated, are involved. Such a case is the globalization of building software projects, where the teams involved in a project are typically in different geographical locations. In this ongoing trend, the typical synchronous forms of communication, such as face-to-face conversations, are being replaced by new tools and techniques. Wikis, as an affordable, lightweight documentation and distributed collaboration platform, have demonstrated their capabilities in distributed requirements elicitation [DRR<sup>+</sup>07] and documentation [SFAV05]. Software requirements specifications, diagrams and images are typically stored in wikis as articles and resources. Using a wiki's built-in features, stakeholders can easily author new requirements, create links and back-links between different artifacts, view and keep track of the changes in specifications and discuss their ideas on a specific topic in its corresponding talk page.

#### **2.2.4 Enterprise Wikis**

Enterprise wikis build on the basic wiki idea by including certain functionality that meets the needs of organizations, such as the ability to easily create and manage many individual wikis for teams, projects, and departments [Mad08]. Wikis are ideal for managing organization products through their development process. Product designers can log their ideas and collaboratively design and document the features and specifications. Manufacturers can be invited to the wiki to discuss on materials and production process related issues. The marketing department can use the information to develop marketing materials. This way, the product information shared in the wiki can be developed by various associated groups and still remain consistent. These information can then be used by different departments, e.g., by a marketing department to develop a marketing campaign, by manufacturers to adapt to changes in the design and by a

support department as a knowledge source to support customers.

Wikis are also widely used by project managers. Each project in an enterprise can have a wiki page, where all the related information and materials are kept and easily updated. The presence of a project's page in a wiki ensures that everyone associated with the project has access to consistent information. Such dynamic information linked to other wiki pages, like reports, strategic plans or even other projects, can then become part of the project management, allowing not only the teams to work more unified, but also the executive staff to draw more confident decisions through transparent and accessible decision-making processes.

Wikis are also used as organizational knowledge base. Available information in the wiki, ranging from technical product information to marketing information, like competitors and their offerings, and educational learning best practices, when kept in a logical structure can play the role of an encyclopedia in the organization. Organization employees can update the wiki with questions, answers, issues or solutions and constantly grow the knowledge base with updated information. This way, a wiki externalizes the knowledge of different people working in relative isolation into a central place accessible to everyone. The available technical information present in the wiki can also be gathered together to play the role of a “collective, site-wide” FAQ<sup>10</sup> system, both to employees and customers of the organization.

### **2.2.5 Cultural Heritage Data Management**

Cultural heritage data is the legacy of artifacts of a society inherited from the past, such as literature. To preserve these artifacts, they are often digitized and stored in nationally and internationally distributed databases. This stored data can be turned into valuable knowledge bases by linking the text or parts thereof to a multitude of other existing supportive data, like background information, and creating a structure from which information can be automatically extracted [WGKK08].

Wiki systems can provide a fast and easy-to-use web interface for this

---

<sup>10</sup>Frequently Asked Questions

isolated data: By exploiting the power of Web 2.0 technologies, wikis allow users to discuss and collaboratively add metadata in form of separate discussion or annotations to the original data. For example, wikis are used as cultural heritage encyclopedias for different domains like architecture, language and history.

## 2.3 Common Problems in Wikis

In this section, we look at a number of problems associated with using wikis found in the literature, in particular, the use cases described in [Mad08].

**General Misuse.** The Wiki philosophy assumes that edits and comments are made in “good faith” and users are trying to help, rather than to hurt the usability or integrity of the wiki. However, intentionally or otherwise, users can exploit the democratic philosophy of wikis by outbreaking behavioral (e.g., personal attacks on other editors) or content-based (e.g., vandalising or spamming) misuse. While some of these problems, such as vandalising an article, can be detected by wiki bots on a syntactic level, other problems with semantic issues, such as the use of offensive language in discussion pages, remain in a wiki system until it is corrected by a human user.

**Loose Structure.** Wikis themselves are shipped to users without any imposed structure, unlike other content management systems and that is what makes wikis capable of adapting to their content. While this feature provides wikis with a competitive advantage over other content management systems, it relies on the trust that users will provide structured content and meaningful relations between the entities. However, this assumption is not always true, especially when users do not oblige themselves to explicitly provide a structure for their content or when already unstructured data is imported into a wiki, e.g., by bulk importing data directly into the wiki database. The result is a wiki containing unstructured, unrelated

and orphaned<sup>11</sup> articles.

**Information Overload.** As a collaborative and easy-to-use content management system, wiki systems can quickly and dramatically grow in size, especially when used in organizations or large online communities like Wikipedians<sup>12</sup>. With no explicit navigation feature in wikis, a keyword search can return hundreds or thousands of pages as a result. Although semantic queries can relieve the overload to some extent, still the user's information need is dispersed over multiple pages. For example, creating a report from a project described in tens of wiki pages or a summary of the available information on a specific topic in the wiki becomes an extremely labour-intensive and expensive task, not to mention the problem of poor search recall, i.e., not retrieving all the relevant wiki pages.

**Information Redundancy.** Redundancy in information typically happens at two different levels: at the document level and at the term level. The former points to the difficulty of finding desired information within a set of artifacts. When an existing information is not retrieved successfully, the person querying the system falsely comprehends this as absence of the content and thus attempts to create a new document for a concept that already exists. Redundancy at the term level refers to the different cognitive ability of wiki users. People tend to invent different names for the same concept. Therefore, two different persons or groups may refer to the same entity in the system environment by different terms. As we previously described in Section 2.1.1, wiki pages are uniquely identified by their titles and it is possible that the same content is stored under two or more different titles, e.g., with alternative punctuation, capitalization or spellings. Some wiki systems, like MediaWiki, manage content duplications by creating "redirect" pages to forward users. Creating redirect pages however, is a task done manually by users, as it requires reasoning on a semantic level.

---

<sup>11</sup>An article with no incoming links

<sup>12</sup>Wikipedians, <http://en.wikipedia.org/wiki/Wikipedia:Wikipedians>



## 2.4 Suggested NLP Solutions

By studying the common problems when using wikis described in the previous section, one of the main root causes seems to be derived from the situation where the human ability to retrieve or organize content becomes limited due to the large size of wiki content or the difference in background or cognitive abilities of its users. This is where, we believe, modern approaches from the NLP domain can help mitigate the information overload that wiki users are faced with. It should be noted that the term NLP encompasses a broad range of techniques from the field of Artificial Intelligence for automated generation, manipulation and analysis of natural languages. In the following, we describe some example NLP techniques to tackle the problems we defined earlier.

**Automatic Categorization.** Categories are a feature of wiki systems that enables users to classify pages on similar subjects by adding them to automatic listings. Adding pages to a specific category is done manually, by embedding special markup inside articles. The process of adding categories to articles can be automated using NLP techniques that would analyze each wiki article and assign a category to its page. Also, NLP techniques can make use of semantic methods to enrich pages with semantic metadata – if supported by the underlying wiki engine – and then create semantic categories to further organize the wiki content, for example, categorizing authors and books description pages and then semantically relating each book to its author.

**Content Development.** When a user is authoring content in the wiki, NLP techniques can help the author by providing him with complementary information retrieved from external resources, like the Web. For example, in a personal wiki, when a user creates a page to write down his ideas about a certain topic, he can ask an NLP service to perform a Web search and bring back a summary of the results, without distracting him in his task. Also, if the service is configured to find related information inside the wiki, it can prevent users from creating duplicate content by retrieving a



list of pages with similar content by invoking NLP services proactively, i.e., while the user is typing in the wiki page.

**Automatic Index Generation.** A combination of different standard NLP techniques can provide a full-text index of the wiki's textual content. Such an index helps users with the accessibility of the content that is difficult or impossible to identify or retrieve through traditional keyword search or page navigation. In case of semantic wikis, where semantic metadata is present in wiki articles, automatically generated semantic indices can further aid users by finding semantic entities in the wiki, like an index of all the mentions of a specific entity type in a wiki used for literature mining.

**Automatic Summarization.** "Wealth of information creates a poverty of attention": Herbert Simon [Sim71] states that in the presence of mass information, there is a need to allocate the attention of the target efficiently among the overabundance of information sources. Imagine 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 neglectation of information due to the existence of orphan 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.** The knowledge gathered 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. Question-answering systems are especially useful when the information need is dispersed in 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.

## 2.5 The Semantic Assistants Architecture

Following an exhaustive description of wiki systems and how NLP techniques can be used to improve wiki users’ experience, in this section we describe the Semantic Assistants architecture – the system we are trying to integrate with wiki systems in order to benefit from its NLP services.

### 2.5.1 System Overview

The Semantic Assistants project [WG09] aims to bring NLP techniques directly to end users by integrating them with common desktop applications, such as word processors, email clients, or Web browsers. The Semantic Assistants framework is a service-oriented multi-tier architecture that brokers NLP pipelines as W3C<sup>13</sup> standard Web services<sup>14</sup>. This means that any of the NLP services deployed in this architecture can be consumed by clients that are able to access the architecture interface over the Web. The Semantic Assistants architecture, as depicted in Figure 4, comprises four tiers. Tier 1 consists of the clients that access the Semantic Assistants server, either directly via its interface written in the Web Service Description Language (WSDL) [CCMW01] or the “Client-Side Abstraction Layer”. CSAL is a library of Java classes that facilitates connecting clients

---

<sup>13</sup>World Wide Web Consortium, <http://www.w3c.org>

<sup>14</sup>Web Services Addressing 1.0, <http://www.w3.org/TR/ws-addr-soap/>

by providing common client-server communication and data conversion functionality.

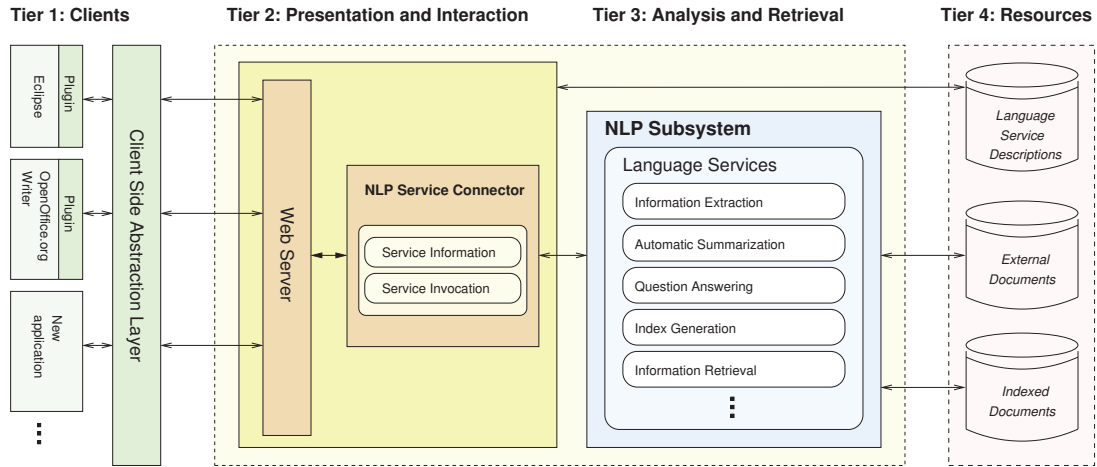


Figure 4: The Semantic Assistants architecture [WG09]

The Semantic Assistants server, written using JAX-WS<sup>15</sup> technology, resides in Tier 2 and offers its functionality via a web service endpoint, to which clients send messages and receive responses. In addition to communicating with clients, the Semantic Assistants server is also responsible for reading and querying the language service descriptions, executing requested language services, and generating response messages once a service execution is finished. The second sub-module of the server tier is the “NLP Service Connector” that has the responsibility of receiving input documents from the clients and executing NLP services inside the “NLP Subsystem” in Tier 3 to perform the analysis on their content. The NLP services are implemented as pipelines based on the General Architecture for Text Engineering (GATE) framework [CMB<sup>+</sup>11]. They reside in the NLP Subsystem and are introduced to the Semantic Assistants server by their formal description files, written in the OWL [Sah07] ontology language. Service description files stored in the “Language Service Description” repository in Tier 4 provide dynamic discovery of NLP services at runtime. In addition, Tier 4 contains other resources, such as documents metadata (e.g., indexed documents) and any other external documents that NLP services need to have access to in order to perform their analysis (e.g., documents

<sup>15</sup>Java API for XML Web Services, <http://jax-ws.java.net/>

found on the Web).

The Semantic Assistants core architecture, along with sample clients, ontologies and NLP pipelines, is released as open-source software and distributed under the AGPL3<sup>16</sup> license on the Semantic Software Lab website<sup>17</sup>.

## 2.5.2 System Workflow

On each bootstrapping of the Semantic Assistants Server, an in-memory model of all available services is formed. The Semantic Assistants Server uses this model object to respond to client requests for querying available services, executing a service and creating service responses. As illustrated in Figure 5, clients send service execution requests to the Semantic Assistants server, along with any applicable runtime parameters to customize the pipelines. The Semantic Assistants server will then provide the NLP pipelines with the input value received from the clients. Once the service execution is finished, the server uses the OWL service description file to determine the pipeline's output type and generate a proper response message (see Section 2.5.3). The response is then transmitted to the clients in form of an XML message to be interpreted and shown to the end-users.

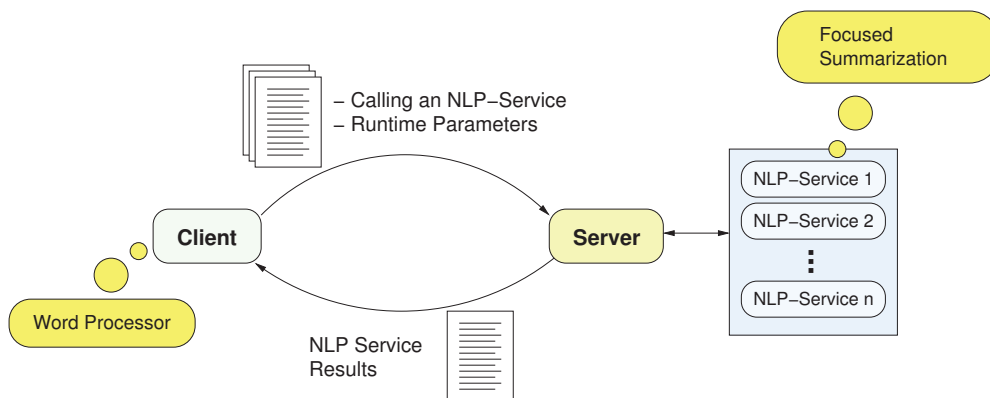


Figure 5: The Semantic Assistants service execution workflow [WG09]

<sup>16</sup>GNU Affero General Public License v3, <http://www.gnu.org/licenses/agpl-3.0.html>

<sup>17</sup>Semantic Assistants, <http://www.semanticsoftware.info/semantic-assistants>

### 2.5.3 Service Results

Once a semantic service execution finished successfully, the results are gathered from the pipeline and passed onto the clients in a uniform XML format with the document type definition shown in Figure 6. The XML is generated on the server side, according to the service output type defined in its description file. In the Semantic Assistants ontology, NLP service results are categorized into three types:

**Annotation.** Many text mining services, such as Named Entity Recognition, provide metadata about a chunk of text with precise offsets in the document that can be as long as a whole paragraph or down to the size of one token, e.g., a space token between two words. The annotation results bear the same schema as GATE Annotations [CMB<sup>+</sup>11]. Each annotation has a “content” attribute that contains the text that the annotation belongs to, a “start” and “end” offset that mark the beginning and end character offsets of the annotation, and a list of “features” that provides additional information, e.g., a *gender* feature for an annotation of type *Person*.

**Boundless Annotation.** The second service result type is the boundless annotation. Essentially, a boundless annotation is the same as a regular annotation described above, except for the fact that the annotation applies to the whole document being analyzed and not a specific text chunk. For example, a semantic service can analyze a document and return one annotation that contains a total score of its readability. Boundless annotations have the same structure as a regular annotation, with an additional attribute that unambiguously distinguishes them from regular annotations.

**File.** Semantic Assistants services can also generate new files as a result. For example, an Automatic Index Generation service can produce a new file that indexes all the noun phrases extracted from various documents with an anchor back to their resources. In the Semantic Assistants terminology, such files can be of any type and since they can be quite large in size, they are not contained in the XML response. Instead, the response message

```

1 <!DOCTYPE saResponse (
2   <!ELEMENT saResponse ( annotation*, outputFile* ) >
3   <!ELEMENT annotation ( document+ ) >
4   <!ATTLIST annotation annotationSet CDATA #IMPLIED >
5   <!ATTLIST annotation type NMTOKENS #REQUIRED >
6
7   <!ELEMENT annotationInstance ( feature* ) >
8   <!ATTLIST annotationInstance content CDATA #REQUIRED >
9   <!ATTLIST annotationInstance end NMTOKEN #REQUIRED >
10  <!ATTLIST annotationInstance start NMTOKEN #REQUIRED >
11
12  <!ELEMENT feature EMPTY >
13  <!ATTLIST feature name NMTOKEN #REQUIRED >
14  <!ATTLIST feature value CDATA #REQUIRED >
15
16  <!ELEMENT document ( annotationInstance* ) >
17  <!ATTLIST document url CDATA #IMPLIED >
18
19  <!ELEMENT outputFile EMPTY >
20  <!ATTLIST outputFile format CDATA #REQUIRED >
21  <!ATTLIST outputFile mimeType CDATA #REQUIRED >
22  <!ATTLIST outputFile url CDATA #REQUIRED >
23 )>

```

Figure 6: The DTD for Semantic Assistants response messages

contains the URL of the file that refers to the server's file system and its MIME type information. Clients can use this information to retrieve the file and choose a suitable presentation method, e.g., a new document in a word processor, or a new browser window.

## 2.6 Résumé

In this chapter, we presented the foundations related to our research work. We described the wiki systems and their use cases and looked into the system architecture of the Semantic Assistants, the system we want to integrate in order to invoke NLP services on wiki content. In the following chapter, we compare existing efforts similar to our work on the integration of semantics and NLP capabilities into wiki systems.

# Chapter 3

## Related Work

In this chapter, we investigate existing efforts similar to our research work. A multitude of wiki engines exist that are designed for the different purposes postulated in the previous chapter. Here, we focus our investigation on two specific types: (1) wiki engines that are enhanced with NLP techniques, and (2) wiki engines capable of developing, managing and querying semantic metadata.

### 3.1 NLP-Enhanced Wikis

‘NLP-enhanced wikis’ refers to wikis 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* [HZG09]. In the following section, we look at Wikulu’s architecture and detail how it improves the users’ experience for developing, organizing and finding content in the wiki by using NLP techniques.

#### 3.1.1 Wikulu, An Intelligent User Interface for Wikis

Hoffart et. al [HZG09] propose an architecture to support wiki 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 wiki content. In [HZG09], they analyze different types of user interactions corresponding to these tasks and try to improve the user experience by providing suggestions on *where* to add or *how* to organize content. The interactive user interface of Wikulu supports users with NLP techniques and involves them at every step of the analysis, using popular Web 2.0 technologies.

### System Architecture

The Wikulu architecture consists of five components, fulfilling two main requirements: The first requirement states that the system should present suggestions regarding the link structure, tags, page segments and possible points of content insertion upon a user's request or in a proactive manner. The second requirement states that the system must be able to act on behalf of the user when he accepts the suggestions. For example, when a user agrees to insert a chunk of text into a page's specific section, the system must be able to access the page content and add the corresponding markup.

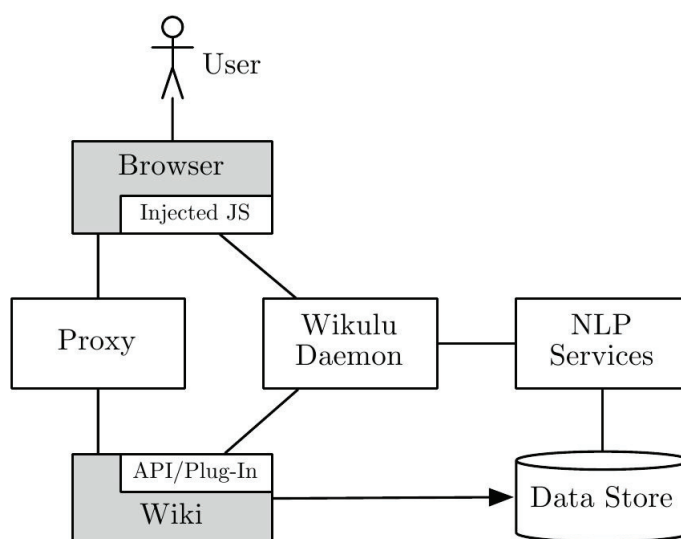


Figure 7: The Wikulu system architecture [HZG09]

The proxy component in the Wikulu architecture depicted in Figure 7



intercepts the interaction between the wiki engine and the user's browser. The proxy adds additional JavaScript and CSS references to the original HTML page rendered by the wiki, once it is enabled on the user's browser. This dynamic manipulation of wiki page content provides an easy and extensible way of augmenting system-generated suggestions with the wiki interface, rather than rendering them by integrating Wikulu directly into the wiki platform. The Wikulu Daemon component, as well as the proxy, is realized as a Java Servlet and is responsible for delegating calls to NLP services and bringing back the results to the user interface.

### **System Features**

The Wikulu architecture provides support for adding, organizing and finding content in a wiki as follows:

**Adding Content.** Wikulu helps to reduce duplicate content by calculating the semantic relatedness of existing content in the wiki and presenting suggestions to the user while typing, allowing him to decide whether the content is redundant. If the user decides to enhance a wiki page by providing new content, Wikulu will then perform a page segmentation analysis based on a TextTiling [Hea97] algorithm and suggest possible points of insertion.

**Organizing Content.** Wikulu offers support for linking newly added pages to the existing ones by calculating the semantic relatedness of the new content to the existing content of the wiki and allows users to add them as related links to the page being created. In addition, Wikulu provides tag suggestions for wiki pages through Keyphrase Extraction [WPF<sup>+</sup>04] of significant terms in each page based on the TextRank [MT04] algorithm.

**Finding Content.** [HZG09] states that the browsing feature of the wiki has already been facilitated because of the user support while adding and organizing content. In addition, Wikulu presents dynamically generated links to related pages in each page without explicit user interaction,

which in turn facilitates browsing. Regarding the content search in a wiki, Wikulu states that by providing semantic metadata for the wiki, the search recall will be improved. The NLP task used to retrieve search results is the same semantic relatedness used in displaying related pages during content creation.

## **3.2 Semantic Wikis**

The second category of wiki systems we want to investigate are the semantic wikis. As briefly explained in Section 2.1.2, the term “semantic wikis” refers to wiki systems that allow their users to formally describe the wiki’s embodied content with different degrees of formalization. Semantic wikis vary in their semantic metadata representation language; some engines offer special wiki markups, while others emphasize the use of standard languages, such as the Resource Description Framework (RDF) [KC04] or OWL [Sah07]. In the following, we detail a number of existing semantic wikis and how they aid wiki users in the task of content development and organization.

### **3.2.1 Semantic MediaWiki, A Semantic Extension to MediaWiki**

Semantic MediaWiki (SMW) [KVV06] is an extension to the MediaWiki architecture that enhances its engine, in order to allow users to annotate wiki content with explicit, machine-readable information. Using this semantic metadata, SMW offers consistency of content, as well as accessing and reusing knowledge presented in the wiki to the users. The primary objective for SMW is the seamless integration of semantic technologies into the established usage patterns of the MediaWiki system. It also has a particular focus on scalability and performance.

## System Architecture

In SMW, users explicitly provide special markup within a page. The SMW engine unambiguously translates those annotations to a formal description using the OWL language. SMW also provides various interfaces to data and tools, based on Semantic Web technologies. To reuse the knowledge contained in the wiki, formal descriptions for one or more articles can be obtained in RDF format via a web interface. Furthermore, SMW allows importing data from ontologies described in OWL, as well as mapping of wiki annotations to existing vocabularies, such as FOAF<sup>1</sup>.

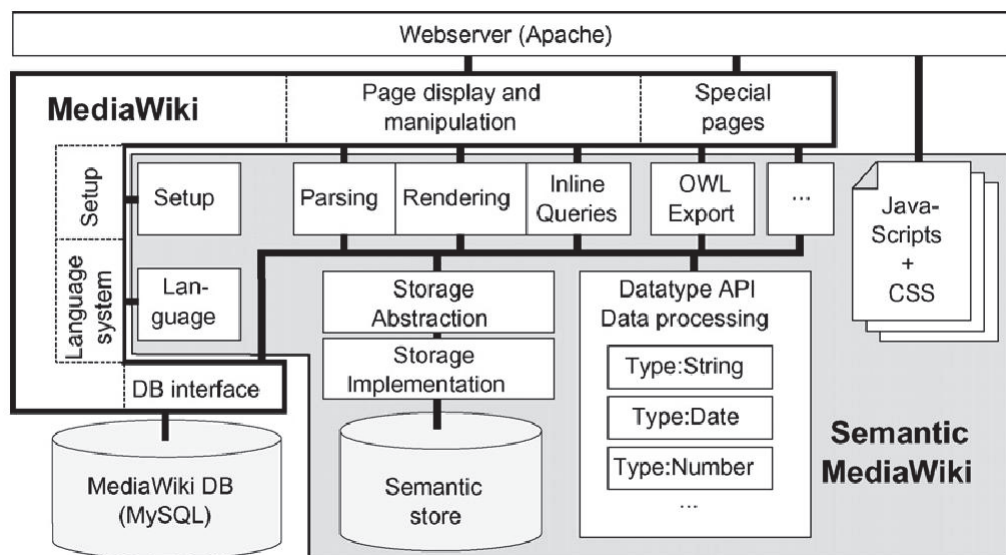


Figure 8: The Semantic MediaWiki system architecture [KVV06]

In SMW, annotations are presented in different parts of a wiki page, based on their type. For example, category links appear only at the bottom of a page, relations are displayed like normal links, and attributes just show the given value. A *factbox* at the bottom of each page enables users to view all extracted annotations: This way, the article's original text remains undisturbed and clearly separated from the semantic metadata. Figure 9 shows the semantic markup embedded in a page's markup and the corresponding factbox rendered by the SMW engine.

<sup>1</sup>FOAF, <http://www.foaf-project.org/>

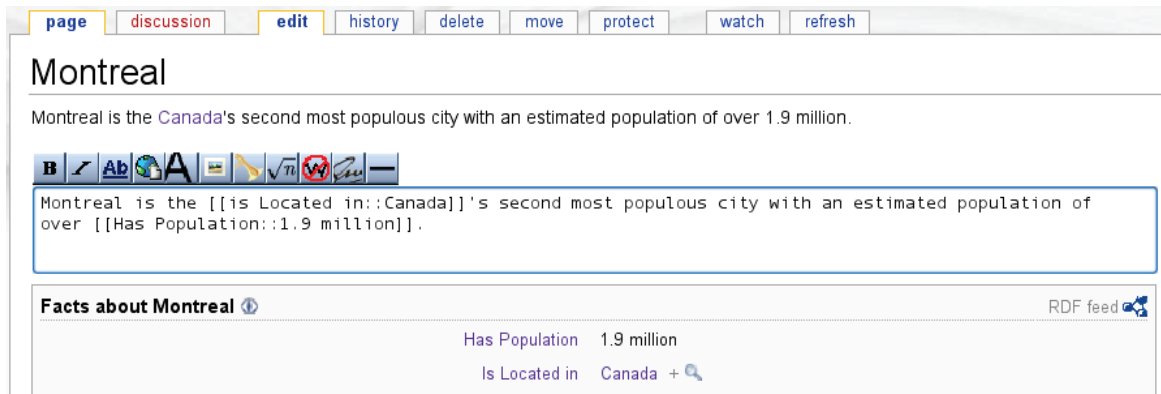


Figure 9: Semantic MediaWiki markup and factbox embedded in a wiki page

## System Features

The Semantic MediaWiki architecture provides support for adding, organizing and finding content within a wiki as follows:

**Adding Content.** SMW relies on the existence of explicitly provided semantic markup to generate semantic metadata from text. For the generation of annotations, users have to manually insert SMW markup following special rules and constraints. Then, these markups are parsed and rendered by the SMW engine and transformed to RDF triples. While this process imposes a learning curve on the user, many extensions for Semantic MediaWiki exist, providing users with WYSIWYG editors and forms that aid users in semantically annotating the content of a wiki.

**Organizing Content.** SMW uses *categories* as a mean to classify pages. Categories are a simple form of annotation that help to classify content based on their semantic relatedness to each other. This feature is already available in MediaWiki and SMW merely endows it with a formal interpretation, i.e., as RDF classes.

**Finding Content.** Users can search for articles using a simple query language that was developed based on the known syntax of MediaWiki. In SMW, the syntax for specifying an annotation is identical with the syntax

for searching it and multiple query statements are interpreted conjunctively. Therefore, users can create powerful *inline queries* that include wildcards, ranges, and subqueries and see the results in the desired format at the same location as the query.

### **3.2.2 IkeWiki, A Semantic Wiki for Collaborative Knowledge Management**

IkeWiki [Sch06] is a semantic wiki, developed to support collaborative knowledge engineering. IkeWiki is best known for its ease-of-use, support for different levels of formalization and its sophisticated, interactive user interface. IkeWiki was primarily developed as a tool for ontology engineering, however, over the time it was extended to be used in a variety of application areas. Eventually, IkeWiki development was stopped and its developers started to work on a successor, the KiWi project<sup>1</sup>. KiWi [KSB<sup>+</sup>10] provides a platform to build Social Semantic Applications based on the layout and functionalities of the IkeWiki.

#### **System Architecture**

The IkeWiki architecture was designed based on four main requirements: (1) Easy-to-use, interactive interface to support different levels of user experience, (2) Compatibility with MediaWiki markup and Semantic Web standards, (3) Support for various levels of formalization for different application areas, and (4) Support for reasoning to derive knowledge that is not stated explicitly in the wiki's content.

The browser view in IkeWiki is divided into three parts: The left column provides navigation functionality and access to tools, similar to other wikis, the centre column contains the human-readable wiki content, and the right column contains additional information about the main content, based on its metadata. The wiki engine provides means to store, update, search, version and query wiki contents. In IkeWiki, the knowledge base

---

<sup>1</sup>KiWi Project, <http://www.kiwi-project.eu/>

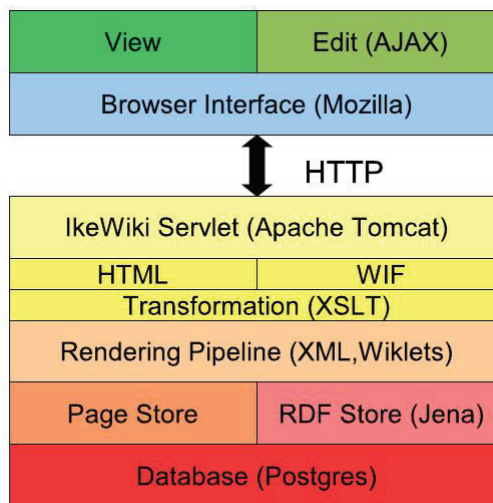


Figure 10: The IkeWiki system architecture [Sch06]

is represented using the Jena RDF framework<sup>2</sup> and data is stored in a Postgres<sup>3</sup> database. When a resource is requested, the XML page content and related RDF data are retrieved from the database and sent to the Rendering Pipelines to transform them into an enriched XML representation, called Wiki Interchange Format. The XML representation can then be sent to external web services or transformed into HTML for presentation in the user's browser.

### System Features

The IkeWiki architecture provides support for adding, organizing and finding content of a wiki as follows:

**Adding Content.** By default, the centre column of IkeWiki displays the main content. This allows the user to switch to editing mode and manually annotate the content with metadata in the form of RDF, via a WYSIWYG editor. The editor interacts with the server back-end to recognize and verify the links and annotations.

<sup>2</sup>Jena, <http://jena.sourceforge.net/>

<sup>3</sup>PostgreSQL, <http://www.postgresql.org/>

**Organizing Content.** The IkeWiki editor allows users to associate a page with one or more types available in the system, as well as annotating outgoing and incoming links with type information. The IkeWiki reasoner also automatically determines and creates annotations based on the page and link types defined in their associated semantic metadata.

**Finding Content.** The knowledge base of IkeWiki is represented using the Jena Framework. The in-memory model for knowledge representation in IkeWiki is frequently synchronized with a database model for persistent storage. Afterwards, the SPARQL Protocol and RDF Query Language (SPARQL) [QL08] engine in the RDF Store component offers semantic, type and tag-based search of the wiki content.

### 3.2.3 SweetWiki, A Semantic Web Enabled Technology Wiki

SweetWiki [BGE<sup>+</sup>08] is a semantic wiki developed based on CORESE<sup>4</sup>, an RDF engine based on Conceptual Graphs, that investigates the use of semantic web technologies to support and “ease the lifecycle” of wikis. It relies on web standards for the wiki page format (XHTML), semantic annotations (RDF) and ontologies (OWL). The main goal of SweetWiki is to improve access to information inside the wiki with faceted navigation, enhanced search tools and awareness capabilities.

#### System Architecture

The implementation of SweetWiki, as shown in Figure 11, relies on the CORESE semantic search engine for querying and reasoning. Pages in SweetWiki are directly stored as XHTML and contain the semantic metadata in form of RDFa triples. Once a page is saved or modified by a user, SweetWiki servlets use GRDDL [W3C07] to extract the semantic metadata embedded in the page markup and convert it to RDF. Users also have the

---

<sup>4</sup>Conceptual Resource Search Engine, <http://www-sop.inria.fr/edelwiss/software/corese>



chance to use an AJAX-powered WYSIWYG editor in the wiki, both for content and metadata editing.

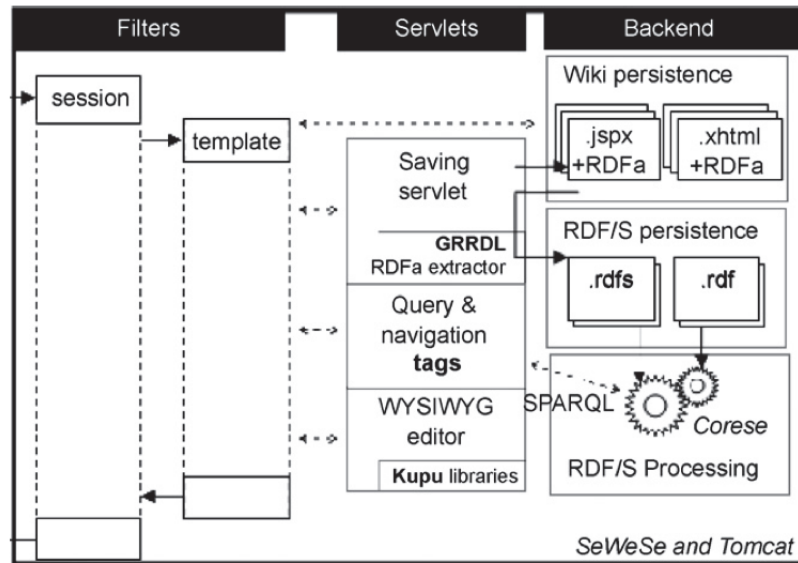


Figure 11: The SweetWiki system architecture [BGE<sup>+</sup>08]

### System Features

The SweetWiki architecture provides support for adding, organizing and finding content in a wiki as follows:

**Adding Content.** SweetWiki provides a WYSIWYG content editor with assisted annotation tools, such as auto-completion, that allows users to see embedded queries or annotations in a page. It also embeds a web-based ontology editor that can be used for editing, creating and managing ontologies, including a user's folksonomy built upon the tags.

**Organizing Content.** The SweetWiki editor is extended to support social tagging functionalities. Users can freely enter tags; an auto-completion mechanism suggests existing ones by performing SPARQL queries to find existing concepts with compatible labels.



**Finding Content.** SweetWiki implements the tag/keyword mechanism with a domain ontology shared by the whole wiki. Then, by reasoning on this explicit ontology, it can find semantically close topics, by making complex queries to find pages according to the topic they were tagged with. Editing the ontology is made possible for users via the wiki user interface to improve the navigation and querying capabilities of the wiki.

### **3.2.4 AceWiki, A Natural and Expressive Semantic Wiki**

Kuhn [Kuh08] presents AceWiki, a wiki using the Attempto Controlled English language. ACE [FSS98] looks like English, but avoids the ambiguities of natural language by restricting the syntax and defining a small set of interpretation rules. The ACE parsers then automatically translate ACE texts into Discourse Representation Structures, which are a syntactical variant of first-order logic, providing a single and well-defined formal meaning for each ACE text. Furthermore, ACE has been used as a natural language front-end to OWL, with a bidirectional mapping of ACE to OWL. This means that AceWiki is able to perform reasoning mechanisms using existing OWL reasoners.

#### **System Architecture**

The main goal of AceWiki is to improve knowledge aggregation and representation through unity and strict user guidance principles. In AceWiki, the ontology is represented in a form that is very close to natural language, using one single language for ontology definition, rules and queries. The two requirements behind AceWiki's design are (1) it should be easy to use and (2) it should support a higher degree of expressivity than a natural language. AceWiki guides its users by using a predictive editor that helps them in a step-by-step manner to create knowledge and ensures the lexical and grammatical correctness of the content. In addition, all the ontological statements written in ACE are valid English sentences and can be immediately understood by any English speaker.

The screenshot shows the AceWiki web interface. At the top left is the AceWiki logo and 'Sandbox Wiki'. The main content area displays the sentence 'Everybody who writes something is an author.' with navigation links for 'Sentence' and 'Logic'. Below the sentence is a 'Paraphrase' section: 'If somebody X1 writes something then X1 is an author.' A 'Syntax Boxes' section shows the sentence with colored boxes around 'everybody', 'who', 'writes something', and 'is an author'. A 'Syntax Tree' section shows a hierarchical tree structure for the sentence, with nodes like 'specification', 's', 'np', 'vp', 'rel\_cl', 'aux', 'det', 'nbar', 'v', and 'pn' connected by lines to the words in the sentence below.

Figure 12: The web interface of AceWiki, presenting the logic structure of a sentence

## System Features

The AceWiki architecture shown in Figure 12 provides support for adding, organizing and finding content in a wiki as follows:

**Adding Content.** As stated earlier, the use of AceWiki is limited to knowledge creation and does not allow the generation of unstructured content produced in natural language. Unlike other semantic wikis, the formal statements are not contained in annotations or considered as metadata, but rather are the main content of the AceWiki. In order to be convenient for both novice and advanced users, the stepwise creation of a sentence

can be done either by clicking on lists of proposed words or by typing the words in a text field. The semantic correctness of the content is not enforced at this level, but suitable words are retrieved on the basis of the hierarchy of concepts and roles and the domain and range restrictions of roles and shown to the user in a list.

**Organizing Content.** AceWiki uses OWL reasoners to perform semantic reasoning on inserted knowledge and only adds them to the knowledge base if it finds no conflict. Therefore, the content of the AceWiki, i.e., the formal sentences, have a sophisticated structure due to their clearly defined semantic relations. Nevertheless, the generation of orphan pages, i.e., pages that are not linked to other pages, or undefined entities, undefined relations is inevitable. That is because although the AceWiki editor enforces the content's lexical and grammatical correctness, it does not necessitate the creation of links at the time of content creation.

**Finding Content.** At the time of this writing, AceWiki offers only keyword search. Semantic queries for finding content is considered as future work.

### 3.3 Discussion

Among the wiki systems described in the previous sections, Wikulu has the most similar motivations to our research work, as they both aim to improve the wiki users' experience through the means of NLP. However, there are significant differences between the two approaches. First, in Wikulu, the execution of NLP services is implemented within their system architecture, whereas our approach will offer such capabilities not within the integration, but backed by a robust, multi-tier and service-oriented architecture – the Semantic Assistants. Using this framework, our Wiki-NLP integration will be able to offer a variety of NLP services<sup>5</sup> to multiple wikis. Second, our Wiki-NLP architecture is also concerned with content development, both the wiki's main content and semantic metadata, whereas

---

<sup>5</sup>As long as they can be brokered via the Semantic Assistants architecture

Wikulu’s emphasis is on the organization of content by providing users with interactive system suggestions.

Regarding the semantic wikis described above, as Buffa explains in [BGE<sup>+</sup>08], they can be distinguished into approaches considering “the use of wikis for ontologies” like AceWiki and approaches considering “the use of ontologies for wikis” like IkeWiki and SweetWiki. Our Wiki-NLP integration envisions the second approach, where the wiki is not used as a front-end for community ontology creation by imposing a specific structure or language on the users. Rather, it uses ontologies populated by NLP techniques to improve the wiki users’ experience. Nevertheless, many of the features of these semantic wikis, such as knowledge presentation and reasoning, can be reused in the design of our integration.

Finally, the most remarkable difference between our Wiki-NLP integration and the aforementioned wikis is that we are not aiming at creating a custom wiki *engine* with NLP capabilities but an *architecture* that would allow arbitrary wiki engines – traditional or semantically enhanced – to benefit from NLP capabilities with no or minimum possible modifications required on their concrete implementation.

### **3.4 Résumé**

Augmenting wiki systems with NLP techniques is a novel area of research and has not attracted a lot of attention yet. In this chapter, we looked at a number of existing works related to our Wiki-NLP integration. We described Wikulu, the only currently existing work on integrating wikis and NLP. Then we looked at a number of semantic wikis and how they use semantic metadata inside the wiki to improve creation, organization and retrieval of content. In the next chapter, we define the requirements for our Wiki-NLP architecture in detail.

# Chapter 4

## Requirements Analysis

Having positioned our research goal in comparison with existing work, in this chapter we define the requirements of our system in detail. We start by the ultimate goal of the Wiki-NLP integration – namely the zeroth requirement:

**Requirement #0: General Integration Architecture.** The ultimate goal of the Wiki-NLP integration is to create an extensible architecture that will allow wiki systems to make use of NLP techniques to improve the user’s experience in developing, organizing and finding wiki content.

The architecture should be general to enable various 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 on the wiki engine – rather, an abstraction layer is required between the two that provides a common ground for communication.

In the following sections, we define our system requirements from three different perspectives: the wiki systems’ end-users, the wiki developers, and the Wiki-NLP integration system as a whole.

## 4.1 End-User Requirements

The first target group of our integration are the wiki end-users. Previously, in Chapter 2, we described how wikis are used in different domains. From our examples, it can be seen that wiki end-users vary from laypersons using a wiki as a personal information management tool to highly technical employees of organizations. Therefore, different background knowledge and cognitive abilities of wiki-end users must be considered during the requirements analysis of this user group.

**Requirement #1: Seamless Integration.** 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.

**Requirement #2: NLP Service Recommendation.** Although wiki articles are all natural language documents in essence, they are still not homogeneous. For example, a person can create articles containing her personal knowledge or ideas about a specific matter, and at the same time, have articles in the wiki that contain her full-text biology research paper with a sophisticated structure and specialized terminology. Furthermore, these articles may even be written in different languages, such as English or French. Therefore, users must be able to see suitable language services for each article, based on its content (e.g., its language), as well as its context (e.g., the article’s category).

*Requirement #2.1: Context-specific NLP Services.* In order to provide wiki users with NLP services that are beneficial to their task at hand, the system must be able to only present the services that are related to the wiki end-user’s context.

*Requirement #2.2: Presentation of User Context.* Our architecture must provide a mean for wiki users to declare their context. Dey [Dey01] defines context as “any information that can be used to characterize the situation

of an entity.” Therefore, the system must be able to gather pieces of information, such as the languages that the user knows, to model the user context.

**Requirement #3: Change Visibility.** Despite the existence of a number of robust NLP techniques that can improve user experience, a complete automated understanding of natural language is still not feasible. Given the complexity of natural language, NLP systems are still not able to deliver perfect results to users. Therefore, the NLP services’ result, such as generated content or semantic metadata, must be visually distinguishable from the wiki’s original content. This way, wiki users can assess the quality of the results separately and merge them with the original wiki page or revert the changes if the delivered results are not satisfactory.

**Requirement #4: Organizing Wiki Content.** The philosophy of wikis emphasizes the quick creation of content. Additional steps required to organize the wiki, such as cross-linking related articles or providing semantic metadata, like assigning articles to categories, are optional and can be simply skipped by users. The integration shall aid users in organizing their wiki content by exploiting the wiki’s embodied semantic metadata derived from an NLP analysis of its content.

**Requirement #5: Finding Wiki Content.** Current wikis offer a full-text keyword-based search of content, which is usually authored by various users. However, because of the vocabulary gap between wiki users, query terms do not always match the terms used in articles and thus, causes the search feature to fail to retrieve relevant results. Additionally, since navigation in wikis is mostly done through linking related pages together, the absence of such links prevents users from finding the content they are looking for. Therefore, the integration shall aid users in finding content inside the wiki and discover concepts or entities that he did not know were present in the wiki.

**Requirement #6: Content Development.** In addition to generating

metadata from the wiki content, the integration must also be able to create the primary content itself. For example, in Wiktionary<sup>1</sup>, 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.

**Requirement #7: Low Learning Curve.** Employing NLP techniques on a wiki's content must not necessitate learning of software or language engineering for the wiki users. This requirement is derived from the fact that wiki end-users are diverse in their background knowledge and no assumption about thereof should be made. In other words, if deploying the integration requires critical software or language engineering skills, it will not be usable by a wiki's non-technical user group.

**Requirement #8: Collection-based Analysis.** An implicit goal of our 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 the 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. An example of such a case is generating a summary from several wiki articles. Users must be able to specify individual pages, as well as complete categories for NLP pipelines' input documents.

## 4.2 Wiki Developer Requirements

Our second target user group are the wiki developers. By wiki developers, we mean software developers implementing a wiki engine, as well as

---

<sup>1</sup>Wiktionary, <http://en.wiktionary.org/wiki/MainPage>



administrators that can enhance a wiki's capabilities by deploying third-party extensions. The important characteristic of this user group is that they may not necessarily be familiar with language engineering concepts.

**Requirement #9: Easy Deployment.** 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 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. In other words, if deploying the integration requires critical software or language engineering skills, it will not be usable by a wiki's non-technical user group.

**Requirement #10: Facilitate Client Integration.** For wiki developers that want to use our Wiki-NLP integration by embedding it into their wiki engines, or providing their users with a sophisticated user interface, the system must facilitate the integration process by hiding the complexity of NLP analysis from the developer's view, offering common functionalities of the system in way that can be easily re-used.

### 4.3 System Requirements

Finally, we have to define requirements from the point of view of our Wiki-NLP integration as a whole. In order to deliver the functionalities expected from the integration, our system requires fundamental characteristics, such as being independent from both the wiki and the Semantic Assistants system, while at the same time being able to exchange data between the two. In the following, we look at these system requirements in detail.

**Requirement #11: NLP Service Independence.** NLP services are used in various domains and have different implementations. While some of these services have direct real-world applications, others more commonly

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.

**Requirement #12: Wiki System Independence.** Considering the variety of wiki engines, the integration must employ a generic approach that allows offering NLP services within wiki engines, independent of their implementation. This requirement places considerable limitations on how the Wiki-NLP communication should be realized, due to the fact that various wiki engines have diverse architectures and are developed using different programming languages.

**Requirement #13: Flexible Response Handling.** According to the Semantic Assistants architecture described in Section 2.5, NLP services produce different types of metadata, e.g., annotations or files. The integration must be able to differentiate the type of the generated metadata in order to adequately transform, store and present the results to wiki users. The response handling mechanism must not be tied to a specific wiki engine and should be able to distinguish a wiki's original content from the developed metadata, as postulated in Requirement #3.

**Requirement #14: Read Content from Wiki.** 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 also 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.

**Requirement #15: Write Content to Wiki.** 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.

*Requirement #15.1: Write Semantic Metadata to Wiki.* In order to exploit the semantic metadata generated by NLP services, they have to be stored in the wiki's database to become persistent. Therefore, our system must store the metadata related to each article in the wiki so that it can be queried later or be reused by other external applications. Also, the system must transform the generated semantic metadata to a format that can be used by the wiki, as well as other external applications.

**Requirement #16: External Data Access.** The integration should be able to retrieve content from external sources in order to perform specific NLP analyses on a wiki's content. For example, the system must be able to read the content of a wiki article and use it as input to an Information Retrieval pipeline that finds related information from external resources, such as an Intranet.

**Requirement #17: Proactive Service Execution.** The system must be able to perform NLP analysis in a proactive and event-based manner. For example, when the integration is used to index the wiki's content, it should be able to perform an automatic index generation every time a change is applied to the wiki content.

## 4.4 Discussion

The requirements defined above detail all the functionality that needs to be implemented in order to enable wiki systems to benefit from NLP techniques. However, since we are trying to benefit from the NLP services brokered via the Semantic Assistants architecture, some of our requirements are already fulfilled, in particular:

- Organizing, finding, and development of wiki content (Requirements #4, #5 and #6) are requirements that are fulfilled by NLP services,

such as the ones described in Section 2.4, rather than the integration architecture. Therefore, since these services are independent of the Wiki-NLP integration implementation, the assumption of the existence of such services in the Semantic Assistant's NLP Subsystem component (see Section 2.5) fulfills the three mentioned requirements.

- According to the Semantic Assistants architecture, NLP services developed based on the GATE framework, irrespective of their concrete implementation, can be brokered to all of the connected clients. Therefore, Requirement #11 is fulfilled by the Semantic Assistants architecture and any NLP service that is brokered by the Semantic Assistants server can be used within the wiki system.
- Any NLP service available in the Semantic Assistants NLP Subsystem capable of performing external information retrieval task is able to access external data for content development. Therefore, Requirement #16 can be fulfilled by services offered by the Semantic Assistants architecture.
- All the NLP services in the Semantic Assistants NLP Subsystem are formally described by their OWL description files. One of the properties of each NLP service is the definition of the context in which the service becomes useful, e.g., the natural languages that the service is capable of analysing. For example, when a list of available assistants is requested from the Semantic Assistants server with a user context object containing English and German as the acceptable languages, the server then executes a SPARQL query against its services metadata repository to find suitable NLP services for the specified user context. Therefore, this feature of the Semantic Assistants architecture fulfills Requirement #2.1.

Before proceeding to the next chapter, let us take a look back at the related work we defined in Chapter 3, and examine each one against our Wiki-NLP integration requirement. The comparison shown in Table 1 helps

Table 1: Comparison of wikis against Wiki-NLP integration requirements

<b>Requirement</b>	<b>Wikulu</b>	<b>SMW</b>	<b>IkeWiki</b>	<b>SweetWiki</b>	<b>AceWiki</b>
Seamless Integration	✓	~	✗	✗	✗
NLP Service Recommendation	✓	✗	✗	✗	✗
Change Visibility	✗	~	✗	✓	✓
Low Learning Curve	✓	✗	✗	✓	✓
Easy Deployment	✓	✓	✓	✓	✓
Facilitate Client Integration	✗	✓	✗	✗	✗
Wiki System Independence	✓	✗	✗	✗	✗
Flexible Response Handling	✓	✗	✗	✗	✗
Read Content from Wiki	✓	✓	✓	✓	✓
Write Content to Wiki	✓	✓	✓	✓	✓
External Data Access	~	✓	✗	✓	✗
Proactive Service Execution	✗	✓	✗	✗	✗

✓ = fully satisfied

~ = partially satisfied or not available in literature

✗ = not satisfied

us to get inspired by the design ideas implemented in their engines, in order to fulfill our remaining requirements.

## 4.5 Résumé

In this chapter, we defined the features and functionalities required in order to integrate various wiki systems and NLP techniques. We analyzed our system requirements from the perspectives of different user groups and justified how some of them are implicitly fulfilled by the use of the Semantic Assistants architecture. In the next chapter, we describe how we derived concrete design decisions from the remaining requirements.

# Chapter 5

## System Design

In this chapter, we describe how the requirements postulated in Chapter 4 are transformed to concrete design decisions. We start by exploring design alternatives and then we elaborate on how our system components can communicate with each other. Finally, the chosen solution for integrating wiki systems and NLP services will be detailed in Section 5.6.

### 5.1 Design Alternatives

The software architecture of a system comprises software components, relations among them and their properties. Juxtaposition of system components results in different system architectures, but a good architecture is the one that is not only able to fully satisfy all the system requirements, but also considers design best practices like extensibility, reusability and modularity. In this regard, we investigate various possible juxtapositions of our system components and discuss whether each alternative is able to fulfill our integration requirements. The goal of this section is to perform a high-level design analysis to find the best place for the Wiki-NLP integration to be implemented. Our three candidate places are: (1) the user's browser, (2) the wiki engine, and (3) the Semantic Assistants architecture.

### 5.1.1 A Browser Plug-in

One of the candidate points of the Wiki-NLP integration is the user's Web browser. Generally, a user interacts with the wiki via his browser interface. The wiki user requests a page either by directly typing its address into the browser's address bar or clicking on a link inside a page. This action sends a request to the web server hosting the wiki system. The wiki engine then retrieves the page content from its database, renders it into an HTML document, and sends it back to the user's browser via its web server.

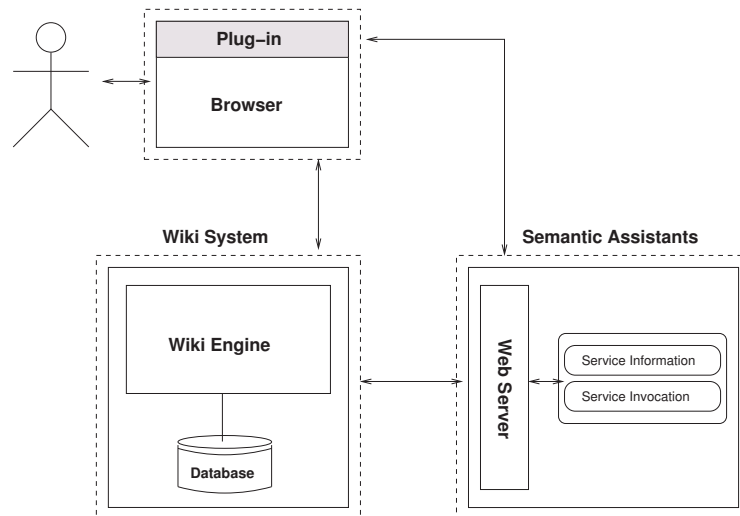


Figure 13: Design alternative using a browser plug-in

In this design alternative, the idea is that a plug-in installed on the user's browser allows him to invoke NLP services on the page content that is being viewed. As depicted in Figure 13, the plug-in lets the user interact with the Semantic Assistants server through new GUI elements in the browser interface, such as dialog windows, and invoke a desired NLP service. The plug-in is responsible for sending the page content along with any other necessary information, e.g., acceptable data formats and languages, to the Semantic Assistants server. When the service execution is completed in the Semantic Assistants server, the browser plug-in receives the results and transforms them into a user-friendly format to be displayed to the user. Optionally, the plug-in can apply its own formatting styles on-the-fly, such as text highlighting, to distinguish the generated semantic metadata from the page's original content. Up to this point, the

generated metadata is only embedded temporarily as HTML markup in the user's browser and is therefore not persistent in the wiki's database. This means that once the user navigates to another page, or closes the browser, the semantic metadata embedded in the page will be lost. Therefore, the plug-in needs to connect to the wiki's database in order to store the results – through its API or via direct queries to its database – using the authentication information provided by the user.

Since using the browser is already an established usage pattern, it is considered as a suitable location for the Wiki-NLP integration. It provides wiki users with a seamless integration and can re-use the browser interface and built-in features, such as sidebars, text selection or detecting a page language. Also, the wiki content for analysis is retrieved from the HTML representation of the page and thus, requires no interaction with its database. This means that the same browser plug-in can be used with multiple wiki systems.

Despite being independent from a wiki engines' concrete implementation to retrieve a page content, in order to write the results to the wiki, the browser still needs to know about the wiki implementation. Also, this option is browser-dependent. This means that a new plug-in must be specifically designed for each available browser. Moreover, it requires wiki users to install the plug-in on their browser, which might not be an easy task for novice users. Finally, since the plug-in's lifecycle is tied to the browser and requires user interaction, it cannot perform a proactive service execution on behalf of the user, as demanded in our system Requirement #17.

### **5.1.2 A Wiki Plug-in**

The main idea in this design alternative is that each wiki system that wants to use the Semantic Assistants NLP services has to implement all of the functionality that is needed to connect to, communicate with and consume the results from the Semantic Assistants server. This is usually realized through designing a plug-in or an add-on, using the wiki's API. Once the plug-in is installed and deployed on the wiki, it provides an adequate user interface for users to inquire about and invoke NLP services on the wiki



pages. As shown in Figure 14, the plug-in uses the wiki API to retrieve a page content directly from the wiki database. Then, it refines the input to eliminate noise, such as formatting markup, and invokes the selected NLP service with the prepared document. Similarly, once the service execution is finished, the plug-in retrieves the result from the Semantic Assistants server, transforms the response into wiki markup and stores it back in the database.

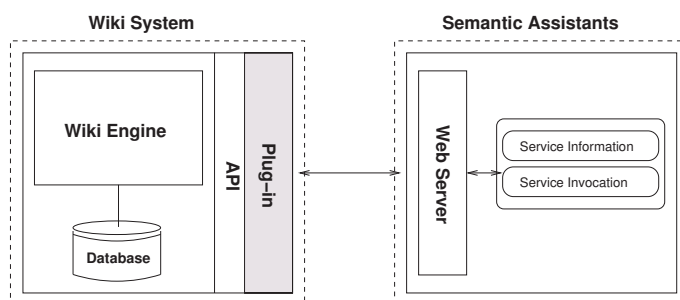


Figure 14: Design alternative using a wiki plug-in

In this design, one plug-in installed on the wiki is offered to all the users, contrary to our previous design alternative. Since the wiki plug-in resides in the wiki, it has direct access to all the wiki system components, including the wiki database and rendering engine. This feature enables the plug-in to read page content directly from the database and store back the NLP analysis results. Also, it can customize the user interface of the wiki in order to aid novice wiki users to benefit from the Wiki-NLP integration. Finally, the plug-in can also act in a proactive manner to invoke periodical NLP analysis on the wiki content, since it is aware of the changes in the wiki. For example, the plug-in can be designed in such a way that every time a new page is created in the wiki, it invokes a service to update the wiki index.

The most prominent disadvantage of this alternative is that for each existing wiki engine, we must develop a plug-in. Concerning the large number of wiki engines and their various structures, this alternative is not desirable and against our vision of a general architecture (Requirement #0).

### 5.1.3 A Semantic Assistants Wiki Component

In this design alternative, we introduce an auxiliary wiki component that eliminates the need for a plug-in, like the one we described in the previous section, and acts as a mediator between the wiki system and the Semantic Assistants server.

The auxiliary wiki component, as depicted in Figure 15, is basically an extension to the original Semantic Assistants architecture. It is solely responsible for the presentation and invocation of NLP services, as well as communicating directly with wiki systems, in order to retrieve their content and store the results afterwards. This design alternative is based on the assumption that wiki systems provide an API for this purpose that the wiki component can use to connect to the wikis' databases. Some of the more established wiki systems, such as MediaWiki, facilitate this interaction by Bot Frameworks written in different programming languages that run on wikis in a methodical, automated manner and automatically manipulate pages.

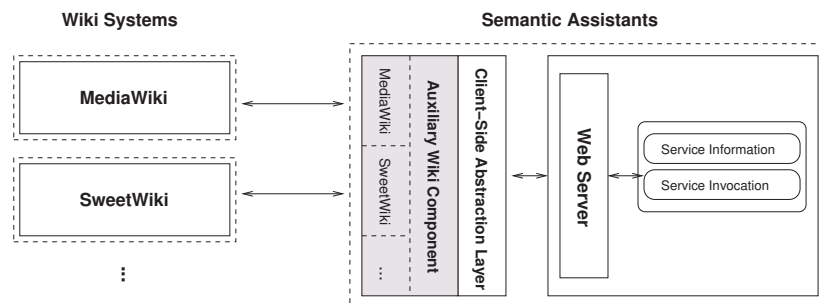


Figure 15: Design alternative using a Semantic Assistants wiki component

Having the Wiki-NLP integration on the Semantic Assistants side has the advantage of re-using the CSAL libraries. The integration can present and invoke NLP services through simple method calls, made available by the abstraction layer. However, accessing the wiki content, such as reading from or writing to its database, is only possible when the wiki component has a corresponding bot framework or fully knows about the wiki's API. This means that, when the wiki is unknown to the wiki component, it cannot employ NLP services on its content, and this makes our architecture wiki-dependent. Also, since the wiki component works independently of

the wiki engine and most of the wiki APIs do not allow modification of the wiki’s user interface, the wiki component cannot integrate an appropriate user interface inside the wiki. Therefore, it requires modification of the wiki engine or an external application to allow users to interact with the broker, similar to the one implemented in [WG07].

### 5.1.4 A Proxy Server Component

Our last design alternative, presented in Figure 16, tries to eliminate the need for an external user interface application by adding a proxy server component to the architecture. The proxy component is inspired by the Wikulu architecture [HZG09] and acts as an intermediary for interactions between the user’s browser and the wiki.

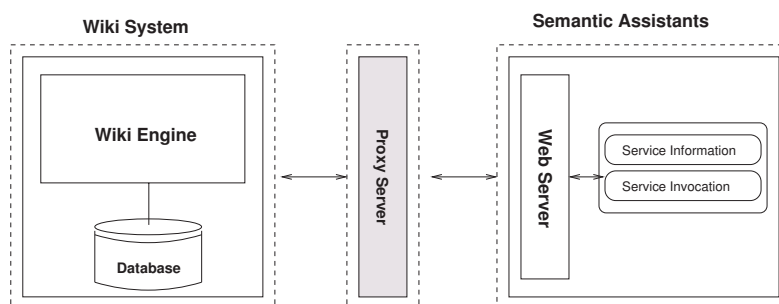


Figure 16: Design alternative using a proxy server component

When the proxy is enabled on the client-side, i.e., set on the user’s browser configuration, all the requests from the user’s browser will be routed to the proxy server. Then, the proxy server processes the request and responds with one of following:

- If the request is to view a wiki page, the proxy retrieves the page’s HTML representation from the wiki’s rendering engine, adds custom client-side code, such as JavaScript and CSS references, to the page and returns it to the user’s browser. This way, the injected client-side code allows users to inquire about or invoke NLP services on the page content.
- If the request is for an NLP service execution, the proxy server translates the request to the corresponding service call in the Semantic

Assistants Server. Similar to the previous condition, the proxy returns the results as embedded client-side code inside the original wiki-rendered page.

Following a service execution, if the user agrees to keep the changes applied to the page content, the proxy receives the confirmation request from the user and stores the metadata into the wiki database.

The generated JavaScript code that is injected into the user's browser is independent of the wiki engine and thus aligned with our wiki independence requirement. Also, using the JavaScript capabilities, the proxy can obtain valuable information from the client, such as the capabilities of the client to accept output types, the browser and the page language, which can be used to recommend appropriate NLP services. However, in order to store the service results in the wiki database, the proxy has to know about each wiki's concrete implementation or its API, and this brings wiki dependency into the architecture. Finally, the proxy component relies on user interaction to perform analysis and thus, is not able to fulfill the proactive behaviour that is expected from the integration.

### **5.1.5 Summary**

In Section 5.1, we described four different design alternatives by juxtaposing the integration components, and examining each one against our requirements postulated in Chapter 4. While some of them exhibit wiki dependency, others fail to fulfill essential requirements, such as seamless integration (Requirement #1). Therefore, neither of the design alternatives can satisfy all of the integration requirements by itself.

However, by having a closer look at the advantages and disadvantages of them, it can be seen that advantages of some of the design alternatives can compensate the disadvantages of others. For example, the disadvantage associated with the wiki component described in Section 5.1.3 is its limitation to offer a general solution to use a wiki API to access its components and generate appropriate markup, as they are both proprietary to

the underlying wiki engine and thus vary from one wiki to another. Similarly, for a wiki plug-in, such as the one described in Section 5.1.2, communicating with the Semantic Assistants server through SOAP messages and consequently resolving the results is a difficult task for the plug-in developers with no knowledge about the Semantic Assistants ontology, whereas using the wiki component alternative can facilitate this process by using the Semantic Assistants CSAL libraries. Consequently, the architecture chosen for the Wiki-NLP integration would be a combination of different components and would firmly depend on the available capabilities of the target wiki.

Nonetheless, however the system components of our Wiki-NLP integration architecture are tied together, there are some essential tasks that they need to be able to perform. In the following section, we will describe them in detail and examine their alternatives, considering the currently available approaches and technologies.

## 5.2 The Analysis Workflow

The NLP analysis of wiki content is a process that starts when a request is sent – manually or proactively – to the Wiki-NLP integration and ends when the results are made persistent in the wiki database or file system. The four main phases of this process are:

1. User interaction via an interface, where the user can select and run arbitrary NLP services and ultimately view the results;
2. Accessing the wiki content for the processing pipelines;
3. Execution of NLP services on the provided content; and
4. Writing the results back to the wiki.

In this section, we investigate various possible ways of performing these steps using available technologies.

## 5.2.1 User Interaction

A wiki user must be able to interact with a graphical interface that allows him to see appropriate NLP services related to his task at hand and invoke an arbitrary service on the selected content. Further explained in Requirement #1, the interface usage pattern must not be largely different from the wiki's, in order to provide a seamless integration and achieve user acceptance. There are various ways of providing users with an interactive user interface that allows explicit service requests to be sent from within a wiki. Having in mind the large number of different wiki systems, the ideal user interface would be one that is independent of the concrete wiki implementation and can be reused among different wiki systems.

Here, we discuss possible ways of providing a user interface for the purpose of Wiki-NLP integration.

### 1. A Standalone Desktop Application

The Wiki-NLP integration can provide the user with a standalone application external to the wiki system, capable of communicating with the wiki system and the Semantic Assistants server. This application will deliver the required user interactions using the full features of the language that it is written in. For example, in [WG07] a Java application is written that can dynamically generate lists of available NLP services, as well as appropriate form fields using the Java Swing<sup>1</sup> GUI widget toolkit, allowing the user to run a service and view the results.

### Advantages

- A standalone application provides the user with an interface, independent of the underlying wiki system and supplied with a rich array of widgets – from basic components, like buttons and check boxes, to the more complex ones, such as tables and text grids.

---

<sup>1</sup>Java Swing, <http://docs.oracle.com/javase/tutorial/ui/overview/intro.html>

- Using the rich interface that the application provides, the integration will be able to properly present the results to the user in dynamically generated widgets, such as windows and lists, and differentiate between the results and the original wiki content.

### **Disadvantages**

- Using an external application requires the user to switch contexts between the wiki system and the application and imposes an extra learning curve on them, which evidently violates the seamless integration as postulated in Requirement #1.
- Each wiki user will need to separately deploy the application on his machine, which might not be convenient for wiki users with limited background computer knowledge as explained in Requirement #9 and is also problematic when the wiki has a large number of users, e.g., in enterprise wikis.

## **2. Generating a user interface on-the-fly on the server-side**

The Wiki-NLP integration interface can also be constructed using a number of Web 2.0 technologies that provide a means for users to interact with the wiki. For example, among these technologies, Asynchronous JavaScript and XML (AJAX) [Gar05] is a popular web development technique used for creating interactive web applications using JavaScript that are able to retrieve data from the server asynchronously in the back-end. For the purpose of Wiki-NLP integration, an AJAX application can be designed in a client-side plug-in or the proxy component described in Section 5.1.4 to query the Semantic Assistants server and bring back the list of available services. It can also embed the NLP analysis results in a page, without stalling the user's interaction with the wiki. Service invocation and result presentation abilities are added to a wiki page by injecting dynamically generated HTML markup and CSS references. This way, the user interactions that normally would generate HTTP requests, e.g., clicking on a button, will take the form of JavaScript calls to the AJAX engine and the

engine asynchronously would make the request to the Semantic Assistants server. Ultimately, the AJAX engine will return the response as HTML elements styled with CSS or an XML document that can be enriched with XSLT<sup>2</sup> to distinguish the generated metadata from the wiki page's original content.

### **Advantages**

- Since generating the user interface is done on the fly using state-of-the-art web technologies and it is embedded into the original HTML pages rendered by the wiki engine, it is completely independent of the wiki's concrete implementation.
- The embedded user interface inside the wiki pages gives the user the impression that he is still using the native wiki interface, thus providing a seamless experience.
- Using custom client-side code and CSS references, the interface will be able to present the results and distinguish the generated metadata from the original wiki content.

### **Disadvantages**

- Dynamically generated user interfaces are generally complicated to design and are strongly dependent on the client-side capabilities. For example, if the user interface is heavily relying on JavaScript code, for users whose browsers do not support JavaScript or who have it turned off, the interface will not be visible.

## **3. Enhancing the Browser GUI**

Essentially, a user interacts with the wiki through his browser interface. Therefore, one of the candidate points of the integration interface is the browser environment. Many of the available browsers feature APIs that

---

<sup>2</sup>eXtensible Stylesheet Language Transformation, a style sheet language for XML documents.



enable developers to extend the browser's capabilities in a way that suits their objectives. These extensions are often known as plug-ins or add-ons. For our purpose, a specially designed plug-in can add a new menu to the user's browser to allow him to inquire about and invoke NLP services. It can also use the browser GUI to present the results, e.g., by using the browser sidebars or opening an external window.

### **Advantages**

- Using the browser GUI brings independence from the concrete implementation of the wiki systems, because the content is retrieved through the browser interface. Once the service execution is completed, results will be again presented in the browser's GUI.
- Since the integration is delivered inside the browser that users access the wiki with, no context switching is required and thus a seamless integration is provided for the user inside the browser that he is already familiar with.
- Using the browser interface gives users the ability to collect multiple pages of a wiki or just select a portion of an article to be sent for analysis, as most browsers already have native support for this kind of user interactions.
- Most of the available browsers have the ability to receive and present different kinds of media types, from plain text web pages to multimedia files, and can interpret CSS references to highlight the analysis results.
- Most of the available browsers feature easy installation of plug-ins, which facilitates the deployment of the integration for wiki users with limited computer background knowledge.

## **Disadvantages**

- A wiki can be accessed by different kinds of browsers, ranging from commercially available ones to the embedded browsers inside hand-held devices. Using this approach will require designing a new interface for each browser.
- Similar to a standalone application, using the browser GUI for our Wiki-NLP integration requires each wiki user to have the plug-in installed on his system, which might not be convenient for wiki users with limited computer knowledge and is also problematic when the wiki has a large number of users.

## **4. Enhancing The Wiki GUI**

A plug-in specifically designed for a wiki has low-level access to its components and the wiki interface. Therefore, the plug-in can contribute to the wiki's native interface with additional links, forms and HTML entities designed to aid users in their interaction with the Wiki-NLP integration. The Collection<sup>3</sup> extension in Wikipedia is an example of this interface type: A wiki user can start the application by clicking on the plug-in's link in the wiki's navigational menu. Triggered by the user, a new session starts that injects extra HTML elements into the wiki page, allowing users to collect wiki articles and perform a desired action such as adding, removing and exporting pages from the gathered collection.

## **Advantages**

- The plug-in can contribute to the graphical user interface of the wiki that the user is already familiar with, thus offering a seamless experience to the user.
- Using special styling features, the plug-in can visually distinguish the generated data from the page's original content.

---

<sup>3</sup>Collection extension for Wikipedia, <http://www.mediawiki.org/wiki/Extension:Collection>

Table 2: Comparison of user interface alternatives

No.	Requirement	Standalone Application	On-the-fly	Browser GUI	Wiki GUI
1	Seamless Integration	✗	✓	✓	✓
3	Change Visibility	✗	✓	✓	✓
9	Easy Deployment	✗	✓	✓	✓
10	Facilitate Client Integration	✗	✓	✗	✗
12	Wiki System Independence	✓	✓	✓	✗
13	Flexible Response Handling	✓	✓	✓	✓

- The plug-in only needs to be installed once on the wiki, and then all of the wiki users can benefit from the integration.
- The plug-in can handle various system response types, based on the available capabilities of the wiki.

### Disadvantages

- Since the plug-in is using the API that is specific to a wiki system, it can only be used on that specific wiki and its clones. Thus, this option does not provide a general solution that can be used on different wiki systems.

Table 2 provides an overview of the comparison of user interface alternatives against the requirements described in Chapter 4.

## 5.2.2 Service Invocation

The second phase of the NLP analysis on wiki content is service invocation. Interacting through the provided user interface, a user selects an NLP service from a list of available services. After an optional step of customizing the pipeline at runtime by providing parameters, a request from the client-side is sent to the Semantic Assistants server, asking for the execution of a specific pipeline on a set of inputs, namely wiki pages or literal text chunks.

In order to use the services in the Semantic Assistants server, clients have to make web service calls over the HTTP protocol. In view of this, let us have a look at the alternative ways of invoking a service from a wiki.

### 1. Direct Java Calls via CSAL

Previously, we described in Section 2.5 that the Semantic Assistants architecture offers an abstraction layer that contains most of the common functionalities needed to connect to the server and consume the results. This thin layer, available as a JAR file, offers convenience methods to make the server communication simpler to use on the client-side. Therefore, wikis written in Java or any language that has native support to handle Java objects only need to make simple Java calls to the designated methods in order to invoke a service. This way, the creation of HTTP requests will be hidden from the client behind the Semantic Assistants abstraction layer.

### Advantages

- In addition to service invocation methods, the Semantic Assistants CSAL libraries offer more convenience methods, such as pre-defined user dialogs for providing service runtime parameters, as well as system settings, such as customizing user interface or server properties. Reusing this code can further facilitate the integration of new clients.
- Using Java for service execution provides more functionalities for

fault management and exception handling, compared to directly communicating with the server over HTTP. In particular, in SOAP requests, a fault flows from the server to the client in the form of a SOAP fault envelope that only consists of the fault code and a human readable explanation. The information about the cause of the fault and more specific details are optional in SOAP fault envelope syntax and can be omitted, which in turn complicates the fault handling on the client-side.

## **Disadvantages**

- While this option offers convenient service invocation methods, as well as the benefits of reusing the Semantic Assistants client-side abstraction layer, it places a major constraint on the range of wikis. For all of the non-Java compatible wiki engines, the gap between the implementation language and Java must be filled with third-party applications or libraries. For example, the PHP/Java Bridge<sup>4</sup> is an XML-based network protocol, which can be used to connect a native script engine with a Java virtual machine. Therefore, re-using CSAL functionalities is subject to the existence of such libraries.

## **2. Dynamically Generated SOAP Messages**

Simple Object Access Protocol (SOAP) [BEK<sup>+</sup>00] is a protocol specification that uses XML for exchanging information between applications via HTTP. There are plenty of libraries and toolkits in different languages available that allow developers to create and consume web services based on SOAP messages. The Semantic Assistants server exposes a WSDL<sup>5</sup> service description with SOAP bindings that specifies the location of the web service and the operations it provides. Using this WSDL file, wikis can create a service execution SOAP message from user inputs in the integration interface and send it to the Semantic Assistants server.

---

<sup>4</sup>PHP/Java Bridge, <http://php-java-bridge.sourceforge.net/>

<sup>5</sup>Web Service Description Language, <http://www.w3.org/TR/wsdl>

Similarly, when the service execution is completed, the results will be transmitted to the client in form of a SOAP response message. The wiki then receives the message and extracts the NLP analysis results from the XML document embedded inside the response envelope.

### **Advantages**

- Since SOAP messages use XML as the standard message exchange format, the Semantic Assistants server response can be directly placed in its body and transmitted to the clients.
- Because SOAP messages are created programmatically by the wiki system, an automatic generation of such requests can realize a proactive behaviour for requesting service execution on the server. For example, dynamic SOAP messages can be created automatically by the wiki system to invoke an NLP service to update the wiki index everytime a change is applied to the wiki content.

### **Disadvantages**

- Although SOAP is an open standard, not all languages offer appropriate libraries or development environment support. This limitation can make the creation of SOAP messages difficult for plug-in developers and system integrators.

### **3. SMW+ Wiki Web Services**

SMW+<sup>6</sup> is a collaborative semantic wiki software suite, developed by Ontoprise<sup>7</sup>, for embedding structured data within small business operations, such as knowledge and project management. The SMW+ software bundle is composed of the MediaWiki engine, along with a number of its extensions.

---

<sup>6</sup>SMW+, <http://smwforum.ontoprise.com/>

<sup>7</sup>Ontoprise GmbH, <http://www.ontoprise.de/>

The Data Import extension<sup>8</sup> is one of the core extensions of SMW+ that enables users to integrate external data into the wiki from various sources, such as CSV files, SPARQL endpoints and SOAP and RESTful web services. Web services have to be defined prior to be used within the wiki using the Wiki Web Service Description (WWSO) syntax provided by the Data Import extension. A WWSO document is a dedicated article under the namespace `WebService` that defines the web service URI, the methods it provides and the possible result formats. Figure 17 presents a WWSO example, which introduces the OpenCalais<sup>9</sup> web service to the SMW+ wiki.

```

1 <WebService>
2   <uri name="http://api.opencalais.com/enlighten/?wsdl" />
3   <protocol>SOAP</protocol>
4   <method name="Enlighten" />
5   <parameter name="content" optional="false" path="/parameters/content" />
6   <parameter name="licenseID" optional="false" path="/parameters/licenseID" />
7   <parameter name="paramsXML" optional="true" path="/parameters/paramsXML" />
8   <result name="result" >
9     <part name="company" path="//EnlightenResult" xpath="//rdf:Description[./
      rdf:type/@rdf:resource='http://s.opencalais.com/1/type/em/e/Company
      ']/c:name"/>
10    <part name="person" path="//EnlightenResult" xpath="//rdf:Description[./
      rdf:type/@rdf:resource='http://s.opencalais.com/1/type/em/e/Person' ]/
      c:name"/>
11  </result>
12  <displayPolicy>
13    <once/>
14  </displayPolicy>
15  <queryPolicy>
16    <maxAge value="10"></maxAge>
17    <delay value="0"/>
18  </queryPolicy>
19  <spanOfLife value="0" expiresAfterUpdate="true" />
20 </WebService>

```

Figure 17: The web service description for OpenCalais

The Data Import extension interprets each WWSO article into a web service object and uses it to invoke its methods. After the successful instantiation of the web service, one can directly call its methods within the

<sup>8</sup>Data Import Extension for MediaWiki, [http://www.mediawiki.org/wiki/Extension:Data\\_Import\\_Extension](http://www.mediawiki.org/wiki/Extension:Data_Import_Extension)

<sup>9</sup>OpenCalais, <http://www.opencalais.com/>

wiki articles, either by using the designated GUI or inserting inline web service calls. Web service calls are inline `{{#ws}}` markup that provide the parameters specified in the definition of a call. Figure 18 shows the web service call for OpenCalais, requesting a list of named entities of type “Person” found in the provided content. The Data Import extension parser engine parses the `{{#ws}}` markup and assigns the actual values to the parameters and ultimately makes a service request to the web service URI. When the service execution terminates, the markup gets replaced, either by the results presented in the requested format or a fault message received from the server response.

```
1 {{#ws:OpenCalais
2 | content = Albert Einstein was a physicist.
3 | ?result.person
4 | _format= list
5 }}
```

Figure 18: A web service call for OpenCalais in SMW+

This idea can be reused by the integration to invoke Semantic Assistants services from within the wiki articles. Special markups entered by the user or generated by the integration user interface can represent a service invocation request and then be transformed into a web service invocation request by a plug-in, a proxy or a wiki component. Likewise, the analysis results retrieved from the Semantic Assistants server will be transformed into the requested format and presented in the wiki page.

### Advantages

- Because the service description and invocation syntax is exclusive to the extension and is independent of the wiki’s native syntax, it can be reused on various wiki systems.
- The web service extension needs to be installed once on the wiki and the integration will be available to all wiki users.



## Disadvantages

- One of our Wiki-NLP integration goals is to provide wiki end-users with NLP services that are beneficial to their task at hand. The list of available services are dynamically generated considering the user context. This means that wiki end-users are not necessarily aware of the existence of such services. However, using this approach requires them to have a knowledge of the service information, in particular, their input and output types, prior to using them.
- The web services are described using formal languages that might not be convenient for wiki users without background computer knowledge.

Table 3 provides an overview of the comparison of service invocation alternatives against the requirements described in Chapter 4.

Table 3: Comparison of service invocation alternatives

No.	Requirement	Java Calls	SOAP Messages	WWS
10	Facilitate Client Integration	✓	✗	✓
12	Wiki System Independence	✓*	✓	✓
13	Flexible Response Handling	✓	✓	✓
17	Proactive Service Execution	✓	✓	✓

\* if the wiki is compatible with Java

### 5.2.3 Wiki Communication

Following the elaboration on user interaction and alternative ways of communicating with the Semantic Assistants server, this part focuses on the other end of the communication channel: the wiki system. Communication with wikis, as explained in Steps (2) and (4) in the introduction of Section 5.2, is divided into two sub-tasks:

**Reading from the wiki.** For NLP pipelines to perform analysis on wiki content, they need to directly access the content of a single or a collection

of wiki pages. Usually, sending wiki page URLs to services in the Semantic Assistants server is sufficient for GATE pipelines to retrieve the content and perform analyses. However, each wiki page contains not only the article content, but also the wiki's navigation menu, header and footer that are considered as noise and not meant to be analyzed. Moreover, when a user wants to perform an analysis on a portion of an article, e.g., only one paragraph, or when the content is not yet saved, i.e., user has typed the content into the wiki page editor but not yet saved to the wiki database, the usual fetching of content through the page URL would not retrieve the correct input. Therefore, different levels of access to the wiki content must be considered for the integration.

**Writing to the wiki.** The generated data from NLP services has to be stored in the wiki database to become persistent. In order to store the results, the integration must be able to communicate with the wiki engine to write to its database.

The following are a number of possible ways to provide a communication channel between the wiki system and the NLP pipelines inside the Semantic Assistants architecture.

### **1. Web Scraping**

Web scraping is the technique of extracting information from websites. Web pages, such as wiki articles, are built using markup languages and typically contain a wealth of useful data in textual form. The process of web scraping mostly entails two main tasks: First, the web scraping API should acquire access to the page containing the required data, e.g., by using HTTP clients. Second, the scraping API has to transform the retrieved text into a structured document, such as XML. There are plenty of web scraping APIs that allow developers to write scripts for customizing the scraping methods. For the purpose of Wiki-NLP integration, this approach can be used to extract textual content from an article inside a wiki page and send it to the Semantic Assistants server for analysis.

## Advantages

- The scraping API uses HTML pages rendered by the wiki engine to parse their markup and extract the actual article content. Therefore, the content retrieval process will be independent of the wiki system.
- Since the article content is already rendered by the wiki engine, the system does not need to interpret the wiki-specific markup by itself.
- In addition to user-triggered web scraping, scraper bots can automatically collect content on a scheduled basis to monitor the wiki content as it changes over time.

## Disadvantages

- Using the web scraping technique only partially satisfies the wiki communication requirement: it is still not possible to write the generated metadata back to the wiki database.
- Since web scraping is usually done by web crawlers, the typical complications associated with crawlers, such as being slowed or blocked by a website administrator, is present and thus can interrupt the communication between the wiki and the Semantic Assistants server.
- The embedded metadata in HTML pages rendered by the wiki engine are not visible in the page and therefore will be missed by the web scraper bot. For instance, in Semantic MediaWiki, properties are assigned to annotations using special markup that is not visible in the HTML representation of the article. Therefore, for example, the statement `Montreal [[isLocatedIn::Canada]]` will only print out the word `Montreal` in the HTML output; the semantic property is not visible to the scraper and will not be provided to NLP pipelines for analysis.
- In addition to an article's content, wiki pages usually contain navigational menus, footers and headers that are considered as data noise

and have to be removed from the content prior to sending it to analysis pipelines. Considering the various implementations of wiki engines, removing such noise from data is a difficult task.

## 2. Using Wiki Bot Frameworks

Wiki bots are computer programs that run on wiki systems in a methodical, automated manner and automatically manipulate their pages. They access the wiki via an available API to perform repetitive tasks, such as maintaining the wiki links, finding vandalism or mass editing. Recently, many bot frameworks have been developed that allow programmers to build bots to manage wiki content without the need to know the concrete implementation of their engines. For instance, at the time of this writing, MediaWiki has more than twenty client API and bot frameworks<sup>10</sup> written in ten different languages, such as Java, Ruby or Python. Therefore, if the integration is written in Java, it can use the MediaWiki Java bot framework<sup>11</sup> that allows connecting to, reading from and writing content to a MediaWiki-clone wiki system. Similar bot frameworks for other wikis can also facilitate the communication between the Semantic Assistants system and the designated wiki engines.

### Advantages

- Using bot frameworks, the Wiki-NLP integration can instantiate a bot object inside its implementation and communicate with the wiki without knowing the wiki's concrete implementation.
- Since bot frameworks provide low-level access to the wiki components, an article's content can be read directly from the database, which allows the pipelines faster access, compared to the web scraping method, and without the problem of dealing with noise.

---

<sup>10</sup>MediaWiki API, [http://www.mediawiki.org/wiki/API:Client\\_code](http://www.mediawiki.org/wiki/API:Client_code)

<sup>11</sup>Java Wiki Bot Framework, <http://jwbf.sourceforge.net/>

## **Disadvantages**

- Wiki bot frameworks are developed using the underlying wiki API, which is proprietary to that specific engine. Therefore, using the bot framework to communicate with a wiki will bring dependency on its API.
- Using wiki bots is restricted to the availability of an external bot framework. This means that, if a bot framework or a similar medium to access the wiki is not present, the integration will not be able to communicate with the wiki to perform the analysis.

### **3. Using The Wiki API**

The most preferable way of communicating with a wiki system is using its API, because it provides direct, low-level access to the data contained in the wiki's database. The difference between the wiki API and wiki bots is that a wiki bot uses the API to perform functions on the wiki and thus provides a more abstract way of communicating with the wiki engine for developers. In addition, the wiki's native API is written in the same language that the wiki system is written in, whereas bot frameworks are available in various programming languages. However, using the API over a bot framework provides a faster access to the wiki components, compared to a bot framework by eliminating the need for passing through an abstraction layer. Also, the wiki API exposes the full functionality of the wiki system, whereas a bot framework may only offer a portion of the wiki's available functionality.

## **Advantages**

- Many wiki APIs, such as the MediaWiki API, can also transform an article's content to a wide range of other formats, such as XML, which can be directly consumed by the analysis pipelines.
- In addition to retrieving article contents, Wiki APIs also offer access to meta information, such as the wiki system, user and revision information that might be useful for the analysis pipelines.

## **Disadvantages**

- Each wiki's API is proprietary to its engine and thus, cannot be reused on other wiki engines to communicate with its components.
- The API can only be used if the wiki offers such a feature. This means that, if the wiki does not expose an API, the integration will not be able to communicate with the wiki to perform the analysis.
- If the Wiki-NLP integration and the wiki API are written in two different languages, there is still a need for an intermediary component to enable the communication between the two parts.

## **4. Direct Access To The Wiki Database**

Essentially, all of the wiki content, such as user information, article content and their associated metadata are stored in a single central database. Wiki systems provide access to their database through a graphical user interface, as well as APIs for system developers. However, in the absence of an appropriate API or when only a subset of wiki functionality is provided by the API, the integration will have to directly access the wiki database to read the data or write the results.

## **Advantages**

- Direct communication with a wiki database provides faster access to its content compared to the previously described alternatives, since there are no additional indirections between the integration and the wiki content.
- By having the knowledge of the wiki database schema, in addition to the raw wiki article content, further data can be extracted that is not generally exposed by the wiki API, such as an article's related RDF data, permissions, etc.

## Disadvantages

- Direct access to a wiki database requires concrete knowledge of the wiki database schema. Since each wiki system has a different database structure, using this approach will make the integration dependent on the wiki implementation.
- Direct access to a wiki database also requires sufficient privileges for manipulating the data. For anonymous bots or proxy servers, such permissions might be denied by the wiki system and thus, the communication will not be possible.
- Obviously, the integration needs to be updated every time the wiki database schema is modified.

Table 4: Comparison of wiki communication alternatives

No.	Requirement	Scraping	Wiki Bots	Wiki API	Wiki DB
10	Facilitate Client Integration	✓	✓	✗	✗
12	Wiki System Independence	✓	✗	✗	✗
13	Flexible Response Handling	✗	✓	✓	✗
14	Read Content from Wiki	✓	✓	✓	✓
15	Write Content to Wiki	✗	✓	✓	✓

Table 4 provides an overview of the comparison of wiki communication alternatives against the requirements described in Chapter 4. While web scraping methods provide access for reading a wiki article, they are not able to write the results back to the wiki database. Other options, such as wiki API or bots, provide the integration with the ability to communicate with the wiki database, but introduce wiki dependency to the integration architecture.

## 5.3 Transformation of Results

In this section, we discuss our design decision towards fulfilling Requirement #13, namely, Flexible Response Handling. In Section 2.1.2, we discussed wiki markup languages and how they vary in syntax and grammar from one wiki engine to another. This inconsistency and the lack of a common standard markup imposes a significant constraint on how our integration can present service results in a way that can be consumed by the wiki engines and subsequently by their users. We also described in Section 2.5.3 that following a successful service execution, results are gathered from the pipeline by the Semantic Assistants server and passed to the client as an XML message. However, this XML message is not usable per se for our end-users with limited knowledge of XML or the Semantic Assistants ontology. Therefore, there exists a need to transform the service results to one of the following formats:

**HTML Markup.** In this option, our integration parses the response XML message. Based on the indicated output type, it produces the corresponding HTML markup for, e.g., annotations, or anchors to result files on the Semantic Assistants server.

Although transforming results to HTML markup provides a user-friendly representation of the results, as it can be formatted with CSS, it is not guaranteed that all wiki engines will allow HTML markup to be saved into their database. For example, the MediaWiki engine does not allow any HTML markup to be embedded in an article except for a handful of standard text formatting tags like `<code>`, `<div>`, and `<font>`. Moreover, this solution breaks the *Separation of Concerns* principle as it combines the data model with its presentation and thus, makes the system maintenance difficult.

**Wiki Markup.** In this option, our integration has to know the exact syntax and grammar of the destination wiki engine to produce proper markup. Based on the available features of the wiki, the integration then produces appropriate markup for, e.g., annotations, wiki links, or new articles.



Compared to the previous solution, producing wiki markup has the advantage that it is not concerned with *how* the results are rendered and displayed to the user, as it relies on the wiki rendering engine to transform markup to an HTML representation. However, it is not easily extensible and requires modifications on the integration core code to include new grammars, as more wikis are integrated into the architecture.

### 5.3.1 Semantic Metadata Representation

In addition to natural language content created by NLP services, we also need to consider how our integration will present the *semantic* metadata generated by pipelines to the wiki, as described in Requirement #15.1. In order to store semantic metadata in the wiki and make it accessible for the wiki's reasoning engine or external applications, the semantic results can be transformed to one of the following formats:

**Wiki Semantic Markup.** Many wiki engines have built-in capabilities to handle semantic metadata. For this, they either provide users with a separate editor to define semantic entities in the wiki or use a special markup. For example, in Semantic MediaWiki (see Section 3.2.1), entities can be defined by using a special SMW syntax. When a page is saved to the database, the SMW engine transforms the semantic markup to standard RDF triples format and stores them in the database. While using this option hides the complexity of transforming results to standard representation formats, it requires the integration to know about each wiki's semantic markup syntax.

**Standard Representation Languages.** The semantic metadata generated by NLP pipelines can be transformed to a standard representation format using formal languages, like OWL, and then transmitted to the wiki. Some wiki engines, like SweetWiki, allow the formal representation of semantic data in RDF language to be imported into their database. For example,

for each annotation that is retrieved from a service result, it can be transformed into an RDF triple using semantic frameworks like Jena<sup>12</sup>. Finally, all the prepared triples can be exported to the XML format and transmitted to the wiki database.

## 5.4 Wiki Independency

One of the most discussed features of our Wiki-NLP integration is wiki independence, as articulated in Requirement #12. Our integration envisions an approach that can provide wiki systems with NLP techniques, without requiring knowledge of their concrete implementation. Here, we detail design alternatives towards achieving this goal.

### 5.4.1 Module-based Architecture

In this option, each wiki engine's structure and syntax is implemented as a module that can be added to the Wiki-NLP architecture. This way, the integration has to provide a public interface that each wiki can implement to describe its structure and capabilities. At runtime, the wiki sending a request to the Wiki-NLP integration has to announce its identity so that the system can instantiate its corresponding module. This behaviour can be achieved by using the Template Method design pattern [GHJV95]. The integration will provide the abstract behaviour that is needed for communicating with a wiki system and each wiki module will provide the concrete implementation. Using highly cohesive and loosely coupled wiki modules will minimize the dependency of the integration to wikis, while the finer granularity of the system makes it easier to understand, maintain, and extend.

### 5.4.2 Semantics-based Architecture

In this option, each wiki engine's structure and syntax is introduced to the Wiki-NLP architecture through its *ontology*. "Ontology", in the context of

---

<sup>12</sup>Jena Semantic Web Framework, <http://jena.sourceforge.net/>

computer science, is a term used to refer to the shared understanding of some domain of interest [UG96]. An ontology contains a set of concepts, e.g., entities or attributes, their definitions and inter-relationships. They are mainly used to formalize knowledge, by using formal languages to allow machines to read and reason about it. In our Wiki-NLP integration, we can use ontologies to formally describe wiki systems and their capabilities. This way, if the ontology is expressive enough to describe a wiki system, the Wiki-NLP integration does not need to know about the concrete implementation of the wiki engines, rather it uses automatic reasoning on their ontologies to discover their structure and capabilities. Typical queries to the automatic reasoning system are “*What are the namespaces in this wiki engine?*” or “*What file formats does the wiki engine allow to be uploaded to the database?*”. Also, having a wiki ontology allows the wiki engines and the Wiki-NLP integration to evolve separately. For example, if a new namespace is added to a wiki engine, only the corresponding engine ontology needs to change and therefore, no code modification on the integration side is required.

## 5.5 Wiki Ontology

Ontologies are typically expressed using declarative languages and we chose OWL [Sah07] – an XML-based language for describing knowledge and sharing ontologies endorsed by the World Wide Web Consortium – for this purpose. The wiki ontology, as shown in Figure 19 is designed using Protégé<sup>13</sup>, and reflects the concepts that are common to all wiki engines and thus, plays the role of an upper ontology for different wiki engines. This way, any new wiki ontology can import this ontology in its description file and reuse the concepts that have been already defined. The wiki ontology OWL description can be found in Appendix A.

Our wiki ontology imports the Semantic Assistants Concept Upper ontology [WG09], a multi-purpose ontology that describes five core concepts to model the relationships between users, their tasks, the artifacts involved

---

<sup>13</sup>Protégé, <http://protege.stanford.edu/>

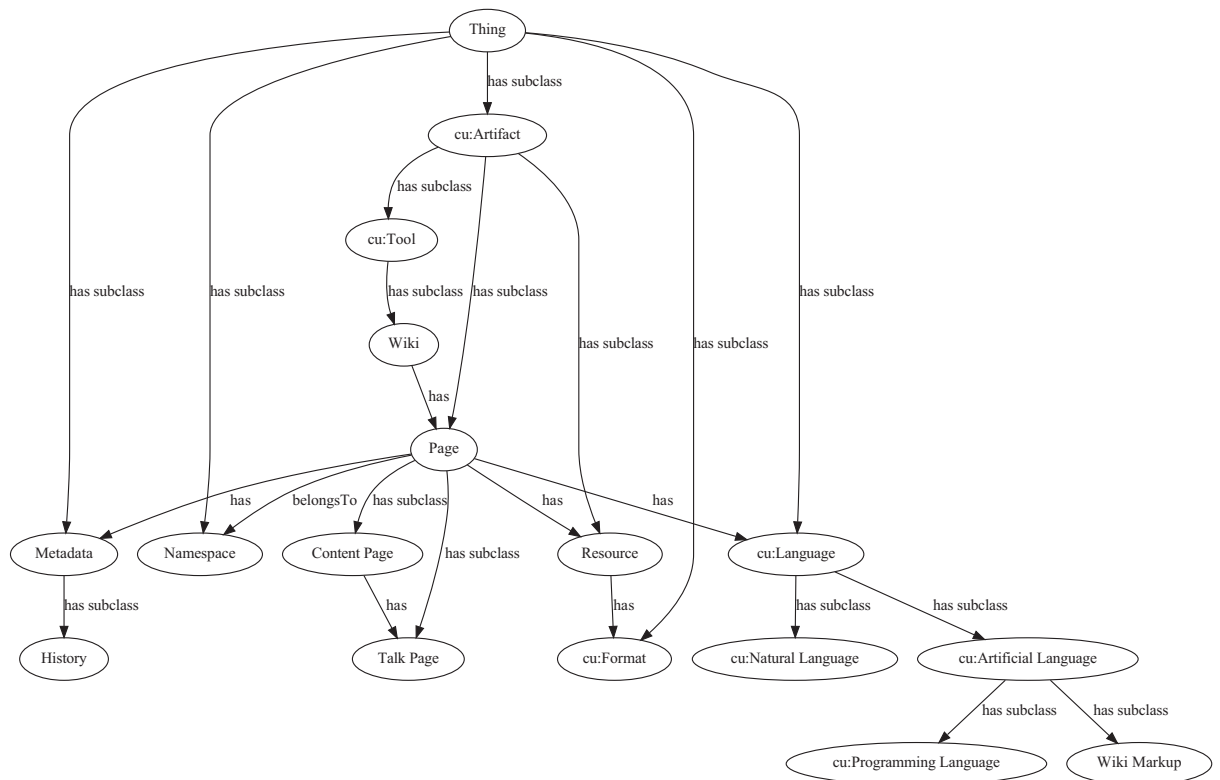


Figure 19: Wiki upper ontology graph

and their format and language. Therefore, the prefix “cu” in the graph nodes denotes that the concept is imported from the concept upper ontology. One of the main concepts defined in the upper ontology is “Artifact”, which is the parent concept for all kinds of objects like documents, files, NLP services, parameters, and annotations and most of our wiki ontology concepts are subclasses of this artifact class.

The main concepts of our ontology, as summarized in Table 5, are as follows:

**Wiki.** This class represents the wiki engines that we want to integrate with NLP techniques. Each engine’s ontology has exactly one instance of this class in its description that specifies the name and the version of the wiki’s underlying engine.

**Page.** Pages are the constituent elements of wikis and present the actual content of the wiki’s database. Pages in wikis are identified by their unique titles and are divided into “content” pages, i.e., articles, and “talk” pages<sup>14</sup> that provide users with a place to discuss their opinions about an article.

**Namespace.** Namespaces are pre-defined categories in wikis that semantically classify pages at a high level. For example, the “Help” namespace in the wiki implies that all the pages inside this namespace are intended to help users of the wiki. Namespaces are also used to avoid name clashes in a wiki; therefore, each page in the wiki *belongsTo* exactly one namespace.

**Resource.** In addition to articles, wikis also contain resources with arbitrary formats. For example, a wiki page can have zero or more pictures, videos or other multimedia objects embedded in its content. The available formats that a resource can have are inherited from the Semantic Assistants concept upper ontology.

**Metadata.** As we defined in the wiki system specification section, each wiki page can have additional data about its main content. For example, each page in the wiki has one “History” page that chronologically lists the modification history of its content, as well as the user information of its editors. In addition, in semantic wikis, semantic metadata such as annotations are added to the page content. This concept defines all the metadata types that can be associated with a wiki page.

**Wiki Markup.** This concept is the parent node for an ontological representation of a wiki markup language and is considered as an artificial language that the wiki page contents are written in. Each wiki engine ontology can optionally instantiate this class to allow the NLP integration to transform the wiki markup to other formats, and vice versa. This node is also considered in design of our ontology for when a standard wiki markup language emerges.

---

<sup>14</sup>For wikis that combine the content and talk pages into one page, the “has” relationship is recursive to the content page.

Table 5: Concepts in wiki upper ontology

<b>Concept</b>	<b>Description</b>	<b>Example</b>
Wiki	Classes of wiki engines	“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	Revision History, Semantic Annotations
Wiki Markup	Ontological representation of wiki syntax	MediaWiki Markup

## 5.6 Developed Solution

In this section, we derive a concrete architecture for our Wiki-NLP integration. The chosen solution described here is essentially a collaborative approach, combining the power of a client-side wiki plug-in and a server-side wiki connector component as described in Section 5.1, working hand-in-hand to deliver the NLP capabilities within a wiki system. Figure 20 presents our developed solution architecture, where the Wiki-NLP integration components are highlighted.

The main idea of this architecture is to divide the responsibilities between the two communication points, based on their capabilities. The wiki plug-in, specifically designed for the Wiki-NLP integration, bears the responsibility of wiki-specific tasks, such as accessing the database or generating markup. The server-side wiki component, on the other hand, encompasses the common functionalities that can be reused by different wikis, such as presenting the Semantic Assistants user interface, handling service invocation requests and refining the Semantic Assistants server response messages.

The wiki component – we call it the Wiki-SA Connector – is a proxy server that acts as an intermediary between the Semantic Assistants server and a wiki system. It has three sub-components: (1) a user-interface

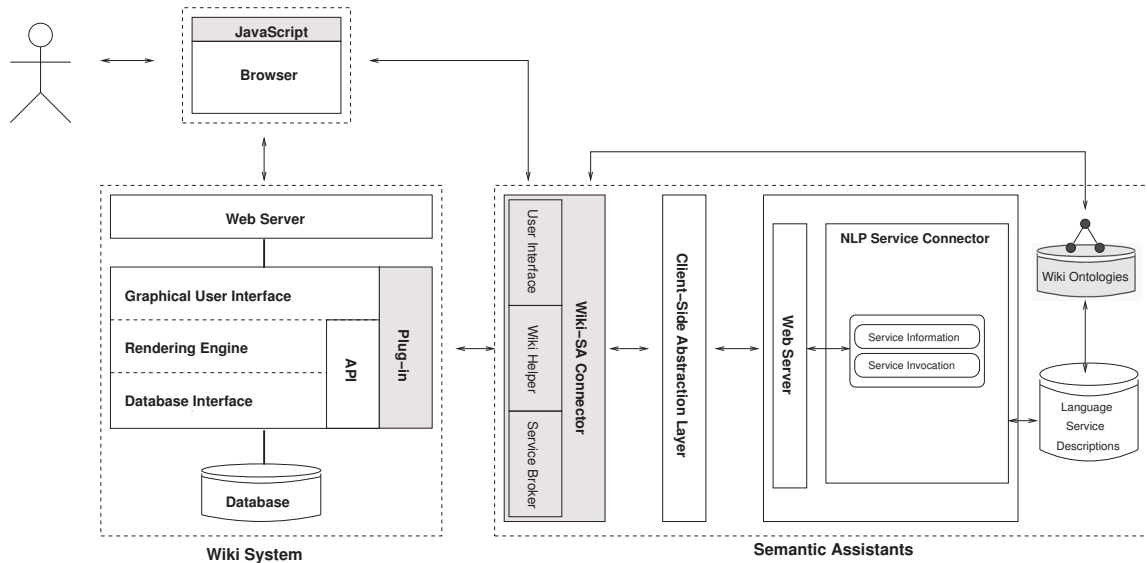


Figure 20: Developed solution system architecture

module that can inject custom code into the user’s browser, e.g., to present the integration interface, (2) a wiki helper to communicate with its known wiki engines, and (3) a service broker module that is responsible for delegating user requests to the Semantic Assistants server.

The wiki plug-in is typically implemented using the wiki API and therefore, it can be used for customizing the wiki user interface for Wiki-NLP integration, or directly accessing wiki system components, such as its database. The plug-in is supposed to be a light-weight extension to the wiki system architecture and is considered in the architecture to perform tasks that are not already provided by the Wiki-SA Connector, e.g., modifying a wiki’s interface.

This system design provides a separation of concerns between the architecture components, which facilitates system maintenance and the integration of new wiki plug-ins: The wiki plug-in does not need to worry about how to handle the complex mechanism of requesting NLP services and resolving the result objects. Reciprocally, the wiki component does not need to be concerned with accessing the wiki database and generating the appropriate markup based on the wiki engine requesting the service. In other words, the Wiki-SA connector will generate the data model, i.e., NLP service results, and the wiki plug-in provides users with the presentation

of the results.

The only disadvantage of this architecture is that it requires both components to be available to deploy NLP analysis on the wiki content. This means that each component in this architecture will not be able to deliver the results to the user without the presence of the other component.

For the transformation of results, as discussed in the previous section, we adopted the approach of transforming Semantic Assistants server responses to wiki markup, when such capability is available in the wiki helper module. Here, the integration will interpret service results into Java objects and the wiki helper module will transform them to wiki markup, using its available parsers. This way, not the integration but the wiki engine is responsible for formatting the markup and presenting it to the user in such a way that the original content of the article and the generated meta-data can be clearly distinguished. Figure 21 presents an abstract diagram for this process.

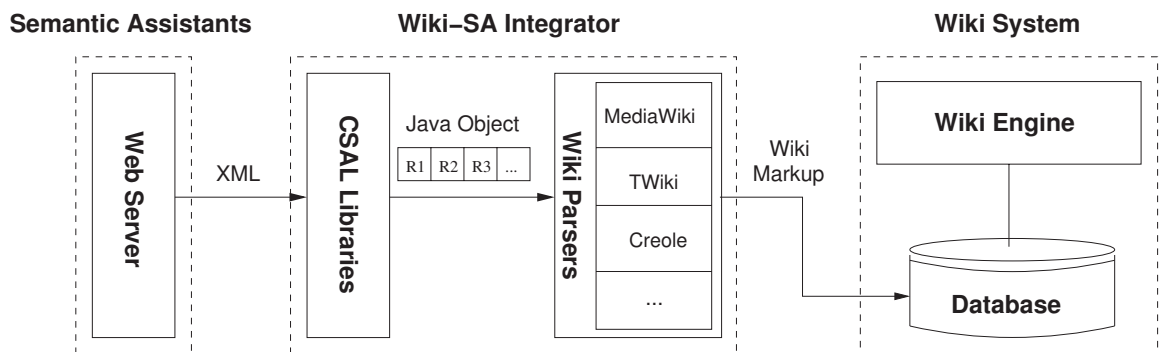


Figure 21: Transforming service results to wiki markup

## 5.7 Résumé

This chapter presented how we translated the requirements identified in the previous chapter into concrete system components. First, in Section 5.1, we looked at various ways of juxtaposing the system components to employ NLP services on wiki content brokered by the Semantic Assistants architecture. Then, in Section 5.2, we described the four phases of



NLP analysis over wiki content and discussed how they can be realized through the use of available technologies. We also presented an ontology that we developed to achieve wiki independency in our architecture in Section 5.4. Finally, in Section 5.6, we presented our developed solution, which depicts an architecture for the integration of NLP services and wiki systems. In the next chapter, we will discuss the implementation details of our chosen solution.

# Chapter 6

## Implementation and Application

This chapter details the steps taken during the implementation process of the solution developed in Chapter 5. We start this chapter by defining the essential components of our system design and then look into their implementations. In each section, we also present how the implementation is applied to a real-world wiki engine. Finally, we investigate how different components communicate with each other and illustrate a scenario to describe the overall workflow of our system.

### 6.1 System Overview

The system architecture depicted in Figure 22 shows how the integration components are tied together and merged into the Semantic Assistants architecture. Before detailing the implementation of our system components, we have to justify the programming language that is used for our system implementation. The Wiki-NLP integration is realized as a Java-based web application. This is because, compared to server-side scripting languages, such as PHP<sup>1</sup> or ASP<sup>2</sup>, Java-based web applications have the following advantages:

**Portability.** Web applications written in the Java language run inside

---

<sup>1</sup>Hypertext Processor, <http://www.php.net/>

<sup>2</sup>Active Server Pages, <http://msdn.microsoft.com/en-us/library/aa286483.aspx>

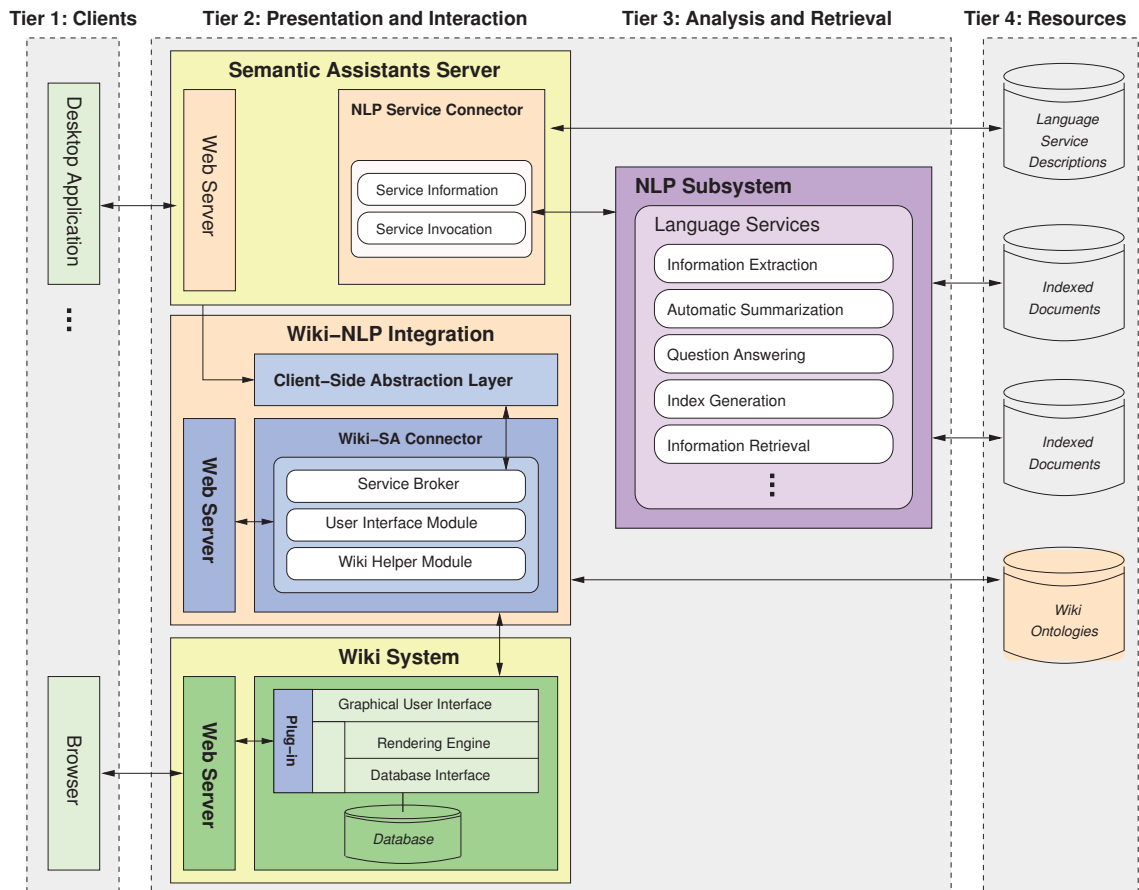


Figure 22: Wiki-NLP integration merged with the Semantic Assistants architecture

a Java Virtual Machine (JVM) on the server environment. Unlike server-side scripting languages that require specific compilers or interpreters to be installed on the server, Java applications are neither operating system nor browser dependent<sup>3</sup>.

**Efficiency.** Once a Java application is loaded into the JVM, it remains in the server’s memory as a single object instance and therefore, client requests are handled through simple and lightweight method calls. Also, In Java applications, concurrent requests are handled by separate threads with Java’s built-in constructs to support their “synchronization”, and thus, they are scalable as well.

<sup>3</sup>Except for Java Applets that require a Java plug-in to be installed on the browser.

In our system architecture, the Wiki-NLP integration web application, called the Wiki-SA Connector, is a Java-based HTTP proxy server that provides the means for system components to communicate and exchange data with each other. In order to use the proxy server, it first has to be deployed in an application container. A container provides the environment for the web application to run. It translates the request and response data between raw protocol formats to Java representations. The Semantic Assistants servlet can be deployed in the same web container as the Semantic Assistants server, or on a remote machine, provided that both machines are accessible over the HTTP protocol.

Once the connector is deployed, it constantly listens for incoming requests over the HTTP protocol. Since the proxy accepts both HTTP GET and POST requests, a user's request for a wiki page can be sent to the proxy directly from a browser's address bar, upon clicking on a hyperlink or as a result of an HTML form submission. When the request is validated, the proxy server will execute the command retrieved from the request's body. For example, when a client requests a wiki page through the proxy, the servlet fetches the content of the wiki page and generates a Java Server Pages (JSP) [CEJ<sup>+</sup>05] page containing the wiki's original content, as well as the Semantic Assistants user interface. The JSP page is then compiled into HTML code and returned to the browser, thus creating a seamless experience for users. Figure 23 presents the high-level system interactions, both when the wiki page is requested directly from the wiki web server (the dotted line) and through the Semantic Assistants proxy server (the solid line).

Similarly, when a service execution request is sent to the proxy server, following the request validation, it is translated to a Java call to the Semantic Assistants server, triggering the execution of an NLP service. The result of the service execution is sent back to the proxy server to be written to the destination wiki database and consequently, returned to the user's browser. In the following section, we will examine the details of our Semantic Assistants servlet component.

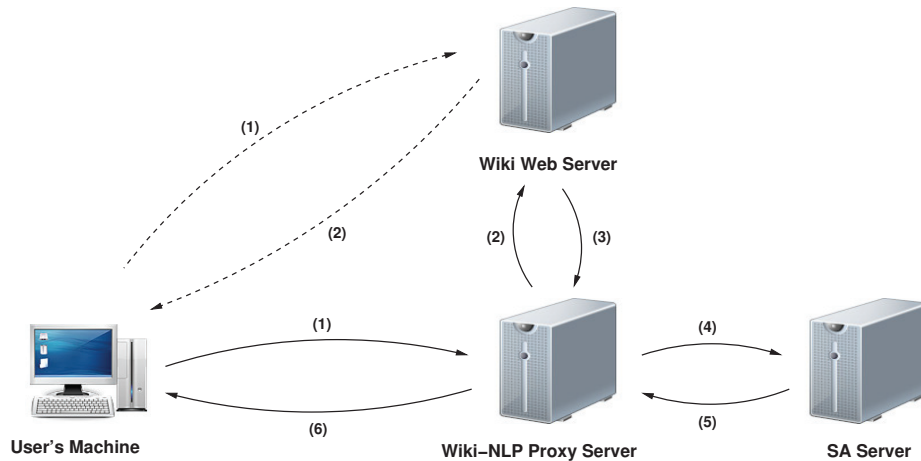


Figure 23: Communication flow between wiki system, wiki web server and the Semantic Assistants servlet

## 6.2 The Semantic Assistants Servlet

The Semantic Assistants Servlet component is a Java class that conforms to the Java Servlet API<sup>4</sup>, a protocol by which a Java class can respond to requests over the HTTP protocol. Servlets can be thought of as the Java counterpart to non-Java dynamic web content technologies. They typically offer the capability to process form data, provide dynamic content or manage state information. In our system architecture, the Semantic Assistants Servlet plays the role of an HTTP proxy server, acting as an intermediary between the Semantic Assistants server, the wiki system and the user's browser. The servlet is designed using the Front Controller Pattern [AMC03] and therefore provides a centralized entry point for handling application requests. In other words, all the requests from the user's browser are sent to the servlet and it will in turn provide the browser with the outcome of the demanded action, such as wiki content or the capability to inquire about and invoke available NLP services.

The Semantic Assistants servlet has four main responsibilities:

**1. Pre-processing of requests.** Every application request received by the servlet is first validated before being dispatched to the business logic.

<sup>4</sup>Java Servlet API, [http://download.oracle.com/docs/cd/E17802\\_01/products/products/servlet/2.5/docs/servlet-2\\_5-mr2/](http://download.oracle.com/docs/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/)

The servlet performs a check on all the necessary parameters to execute the demanded action by examining the request's parameters found in its query string (in case of a HTTP GET request) or the request header (in case of a HTTP POST request), as well as cookies embedded in the body of the request. A list of parameters found in each service request can be found in Table 6. Pre-processing the requests provides a preemptive behaviour against malicious or faulty requests to be sent to the Semantic Assistants server.

Table 6: List of parameters in HTTP service requests

<b>Parameter</b>	<b>Description</b>	<b>Possible Value</b>
Action	Action to be executed by servlet	"Proxy", "Invoke"
Wiki Engine	Name of wiki engine	"MediaWiki", "TWiki"
Wiki URL	URL of the wiki engine	any URL
Username	Bot username	any string
Password	Bot password	any string
Scope	Where the results should be written	"self", "other"
Target	The page name to write the results	any valid page name
ServiceName	Name of service	any string
Params	Service's Runtime Parameters	delimited list of strings
Input	List of wiki pages as service input	delimited list of URLs
Lang	Languages that the user know	"English", "Spanish"
Format	Desired response format	"XML", "Markup" (default)

**2. Dispatching requests to the business logic.** Following the pre-processing phase, the servlet then dispatches the request parameters to a wiki factory class, which decides whether the underlying wiki engine is supported by the integration. The factory class, as the name suggests, is a Java class using the Factory Method Pattern [AMC03] that will match the underlying wiki engine against the known wikis residing in the servlet's repository of wiki ontologies. Consequently, the request would continue to be processed in the business logic with the right type of wiki object or returned back to the servlet with an error message to be sent to the browser. Section 6.2.2 provides more details on the wiki object creation

process.

**3. Controlling the display flow.** Based on the status of the pre-processing phase or business logic outcome, the servlet maps the request to a chosen JSP page for the templating mechanism described in Section 6.4.1. For example, if an exception occurs during the pre-processing or service execution, the servlet can store the exception in the request object and forward the display to a JSP page, providing the user with detailed information about the exception.

**4. Maintaining the ontology model.** As we described earlier in Section 5.5, wikis are introduced to the Wiki-NLP integration by their ontologies. The Semantic Assistants servlet's `OntologyKeeper` class is specifically designed to load and query wiki ontologies. This class keeps the in-memory model of ontologies during the lifecycle of the servlet and can query the servlet's in-memory ontology model using SPARQL or Protégé libraries.

As mentioned earlier, the Semantic Assistants servlet acts as an intermediary between the Semantic Assistants server, the wiki system and the user's browser. For each of these system endpoints, there exists a sub-component 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.

### 6.2.1 The User Interface Module

This module is responsible for generating the Semantic Assistants user interface for wikis. Since wikis are accessible through Web browsers, this module is designed to generate an HTML representation of the Semantic Assistants user interface, allowing users to see available assistants and invoke arbitrary NLP services, as shown in Figure 24.

The user interface of the Wiki-NLP integration uses tabs to divide the interface content into separate panes, which can be viewed one at a time. This way, wiki users are not overwhelmed with a lot of options, and the

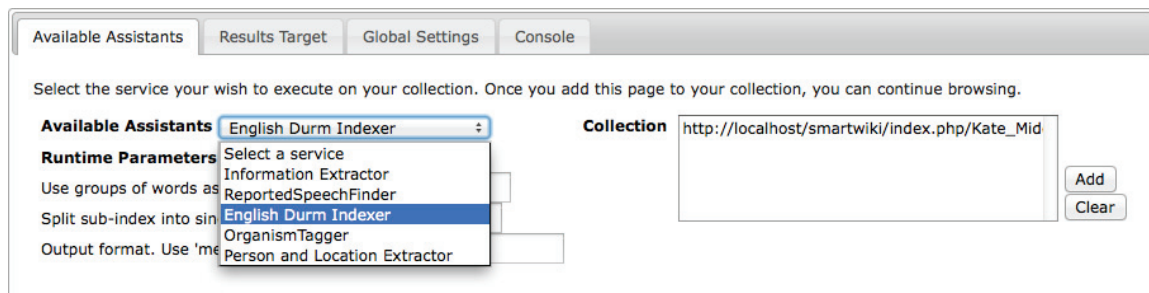


Figure 24: Semantic Assistants user interface generated by the servlet

design allows more features to be added to the user interface in additional tabs.

The first tab of the integration user interface allows users to inquire about available NLP services and customize them at runtime. It also lets users add or remove pages from their “collection”. Each page URL that is added to the collection is kept inside a cookie in the user’s browser. This way, a user can navigate to other pages and add them to the collection. Using the collection feature, users can execute an NLP pipeline on all of their selected pages at once. The list of available assistants is retrieved from the Semantic Assistants server every time the interface is requested by a user. Therefore, when a new service is added to the Semantic Assistants server, the list will be automatically updated.

The second tab gives users the chance to select the location where the service results should be written. The available options are provided to users by reasoning on the underlying wiki engine ontology. For example, as shown in Figure 25, when a user chooses to store the results in a separate page from the original article, the user interface module will ask the servlet’s `OntologyKeeper` class to return all the namespaces of the wiki by querying its ontology. Finally, by pressing the “Run Service” button, a service invocation request is sent to the servlet via AJAX technology, along with any required information, e.g., the name of the requested service or list of input pages.

The third tab provides users with the ability to change the Semantic Assistants server that they are connected to, by selecting from a pre-defined list of servers, or defining a custom one. This way, users can dynamically



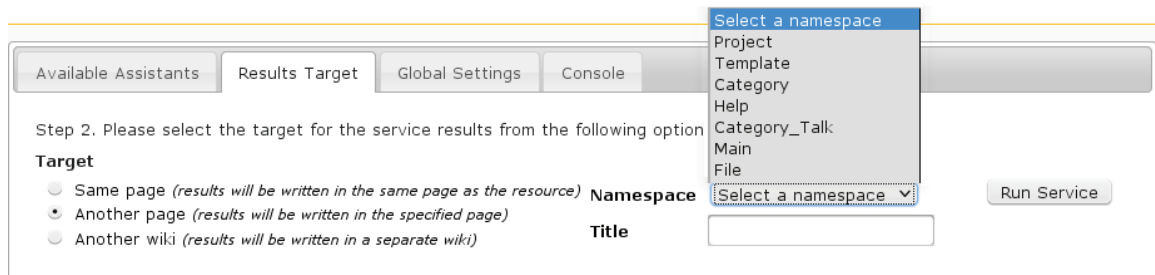


Figure 25: Semantic Assistants user interface second tab

change servers to have access to a variety of NLP services. The selected server address is also stored in the user’s browser and thus, will be remembered the next time he asks for the proxy page.

The fourth tab is designed as a place to provide users with log messages sent by the servlet on the progress of an NLP service being executed in the server. This tab is automatically activated when a service execution is finished. It informs the user on the status of the process and the place to find the results.

While the static elements of the user interface are pre-defined in the Semantic Assistants proxy JSP page, dynamic content is generated by the user interface module and embedded in the page. For example, Figure 26 shows an excerpt of the servlet’s proxy JSP page. It can be seen that the static parts of the JSP page, like the table structure, are literally placed in the page, whereas the dynamic parts, like the list of available assistants, are inserted by direct Java calls. Line 5 of the excerpt, starting with “<%” and ending with “%>”, asks the servlet to place a Java call to the designated class and replace the line with the method’s return value.

For on-the-fly modification of the user interface, intended divisions are coded inside the page using HTML *div* elements, in order to inform the servlet about the place to inject the generated code. For instance, line 15 of the excerpt defines a part of the page as “*saRTPParams*”, which will be used to embed service runtime parameters textfields. When a service is selected by a user from the list of available assistants, the page’s JavaScript code, as shown in Figure 27, sends a request to the servlet using AJAX technology, to inquire about its runtime parameters. The returned results

```

1 <table>
2 <tr>
3 <td>
4 <label for="semAssistServices" id="lblServices">Available Assistants</label>
5 <%= HTMLGenerator.servicesCombobox(SemAssistServlet.services) %>
6 </td>
7 </tr>
8 <tr>
9 <td>
10 <label for="saRTParams" id="lblParams">Runtime Parameters</label>
11 </td>
12 </tr>
13 <tr>
14 <td>
15 <span id="saRTParams"></span>
16 </td>
17 </tr>
18 </table>

```

Figure 26: The proxy JSP page code

from the request will be injected into the page's division identified as “*saRT-Params*”.

```

1 // Sending an XMLHttpRequest to the servlet via AJAX
2 var selectedService = $("#semAssistServices option:selected").val();
3 var req = proxyServer.concat("params&serviceName=").concat(selectedService);
4 xmlhttp.open("GET",req,true);
5 xmlhttp.send();
6
7 // Injecting the results to the page using jQuery
8 xmlhttp.onreadystatechange=function(){
9     if (xmlhttp.readyState==4 && xmlhttp.status==200){
10         $("#saRTParams").html(xmlhttp.responseText);
11     }
12 };

```

Figure 27: JavaScript code for on-the-fly user interface modification

## 6.2.2 The Wiki Helper Module

The wiki helper module encompasses the classes required for communicating with wiki engines. This module contains the wiki factory class that delegates the application requests to the designated wiki class. It works

closely with the Semantic Assistants Servlet’s wiki ontology repository (see Section 6.2.4). Each known wiki engine described in the repository must have its associated classes in this module that knows how to make use of the ontology: We call this class the `WikiOntoKeeper`. In addition, each wiki engine must have two other separate classes: (1) a class that can connect to the wiki engine and is responsible for reading from and writing to the wiki database: We call it the `WikiHelper`; (2) a class that can transform service result Java objects to the wiki’s specific markup, the `WikiParser`. On each servlet bootstrapping, the list of known wikis is created by the servlet’s `OntologyKeeper` class. This list is sent to the `WikiFactory` class when a request for a wiki comes in. The factory class then matches the wiki engine parameter against the list of known wikis, and if a match is found, the correct type of wiki engine object gets created. The wiki object itself will bear the responsibility of creating its three helper, parser and ontology keeper objects along with any other needed classes. Therefore, as far as the servlet is concerned, it can instantiate a wiki object, as shown in Figure 28, without worrying about the concrete type that will be defined at runtime.

```
1 // Concrete wiki type is determined at runtime by examining the input argument
2 WikiEngine wiki = WikiFactory.getWiki(wikiEngine);
```

Figure 28: Java code to instantiate a wiki object

Figure 29 illustrates the wiki factory pattern used to dynamically create a “MediaWiki” engine object.

### 6.2.3 The Semantic Assistants Broker

The Semantic Assistants Broker module is the connecting point of the servlet to the Semantic Assistants server. When a service execution request is received by the servlet, it is dispatched to this module after the pre-processing phase. This module contains a “broker” class that connects to the Semantic Assistants server defined in the request parameter and triggers the execution of an NLP service. Since the broker module

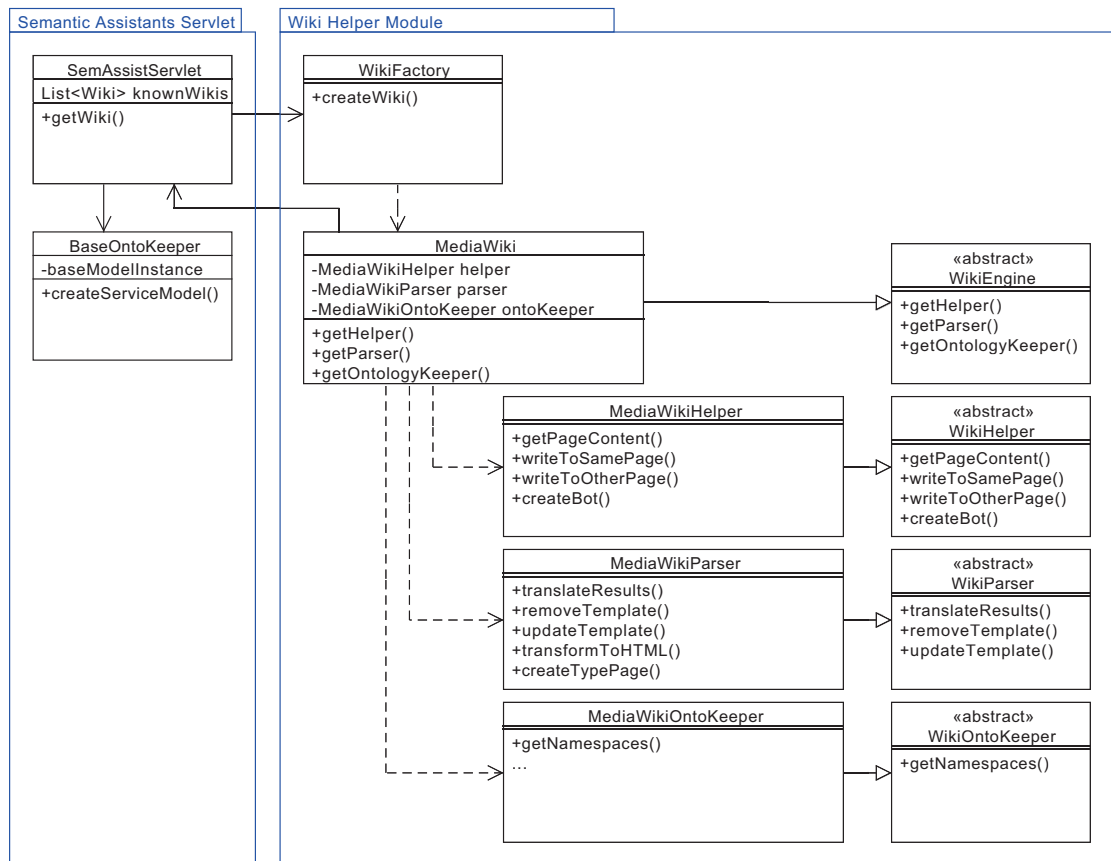


Figure 29: UML class diagram presenting the wiki factory pattern

classes are written in Java, they can directly use the Semantic Assistants CSAL libraries (see Section 2.5) that further facilitate the communication with the Semantic Assistants server, as well as transform service results to Java objects accessible by the wiki's helper and parser classes.

### 6.2.4 The Wiki Ontology Repository

The last sub-component of the Semantic Assistants Servlet to describe is the wiki ontology repository. The ontology repository resides inside the servlet's web application and contains the wiki upper ontology file as described in Section 5.4, along with formal descriptions of supported wiki engines in form of OWL files. On each servlet bootstrapping, the servlet's

ontology keeper class runs over the repository OWL files and creates an in-memory model of the wikis by parsing them using Protégé’s OWL libraries. The in-memory model is created only once, because parsing OWL code and constructing the appropriate data structures is a time-consuming process, despite the efficiency of the Protégé libraries. Therefore, the same consistent model that is created at start up is served to wiki ontology keeper classes faster and without having to parse and repeatedly process the same data.

In our implementation, we have defined a wiki ontology for the MediaWiki engine. The MediaWiki ontology imports the wiki upper ontology class and therefore inherits all its concepts such as *pages*, *namespaces* and *resources*. Specialized concepts, only applicable to MediaWiki, such as *Virtual namespaces*<sup>5</sup>, were added to the ontology. The complete MediaWiki OWL file is provided in Appendix B.

### 6.3 The Semantic Assistants Wiki Plug-in

The wiki plug-ins described in our architecture are typically implemented in the same language as their wiki engines. The main purpose of a plug-in is to provide functionalities that cannot be offered through the servlet, such as proactive service execution described in Requirement #17. For example, the plug-in can be designed in a way that creates pre-defined service requests to the Semantic Assistants Servlet on a time or event triggered basis. Moreover, since the plug-in is installed on the wiki and has direct access to the wiki database, it can patrol content changes in the wiki and create dynamic service execution requests to the servlet to analyze the new content or flag already existing results as outdated when the original content of the page changes.

For our MediaWiki integration, we have developed an extension written in PHP. Once installed, it introduces a new menu item to the wiki’s navigational menu and adds six templates to customize NLP result presentation in wiki pages. The Semantic Assistants MediaWiki plug-in is a light-weight

---

<sup>5</sup>MediaWiki Namespaces,<http://www.mediawiki.org/wiki/Help:Namespaces>

extension, in fact, its whole implementation code is shown in Figure 30.

```
1 <?php
2 # Not a valid entry point, skip unless MEDIAWIKI is defined
3 if ( !defined( 'MEDIAWIKI' ) ) {
4     exit( 1 );
5 }
6
7 $dir = dirname( __FILE__ ) . '/';
8
9 # Extension information to be displayed in the "Version" page
10 $wgExtensionCredits(' semantic') = array(
11     'path' => __FILE__,
12     'name' => 'Semantic Assistants',
13     'version' => '1.0',
14     'author' => array( 'Bahar Sateli' ),
15     'description' => 'Offers NLP services by connecting the Wiki to the Semantic
16         Assistants framework.',
17     'url' => 'http://www.semanticsoftware.info/semantic-assistants-project',
18 );
19
20 # Set up hook
21 $wgHooks('MonoBookTemplateToolboxEnd') = 'wfToolboxLink';
22
23 function wfToolboxLink(&$monobook) {
24     # Create a link in the menu pointing to the Wiki-NLP servlet
25     print("<li> <a href=\"http://loomp.cs.concordia.ca:8080/Wiki-NLP/SemAssistServlet
26         ?action=proxy\">Semantic Assistants</a></li>");
27     return true;
28 }
```

Figure 30: The Semantic Assistants MediaWiki plug-in code

Figure 31(a) presents the MediaWiki's Version page<sup>6</sup> that shows the Semantic Assistants Wiki plug-in installed on its engine. The Semantic Assistants menu item as shown in Figure 31(b) provides the users with the ability to request NLP services on any wiki page through the ease of one click. Clicking on the "Semantic Assistants" link causes a proxy request to be sent to the Semantic Assistants servlet. From the user's point of view, the page is simply reloaded, whereas the content of the wiki page is now served via the Semantic Assistants proxy server and has the Semantic Assistants user interface embedded in it.

<sup>6</sup>MediaWiki Version Page, <http://meta.wikimedia.org/wiki/Special:Version>

## Installed extensions

Semantic extensions	
<a href="#">Semantic Assistants</a> (Version 1.0)	Offers NLP services by connecting the Wiki to the Semantic Assistants framework.
<a href="#">Semantic MediaWiki</a> (Version 1.5.6)	Making your wiki more accessible - for machines <i>and</i> humans ( <a href="#">online documentation</a> )

(a)

toolbox
<ul style="list-style-type: none"><li>■ <a href="#">What links here</a></li><li>■ <a href="#">Related changes</a></li><li>■ <a href="#">Special pages</a></li><li>■ <a href="#">Printable version</a></li><li>■ <a href="#">Permanent link</a></li><li>■ <a href="#">Semantic Assistants</a></li><li>■ <a href="#">Browse properties</a></li></ul>

(b)

Figure 31: Semantic Assistants plug-in installed on MediaWiki

As mentioned above, the second feature of the Semantic Assistants MediaWiki plug-in is the addition of six new templates. These templates are used to present the service results once they are written back to the wiki's database. Provided that the structure of the templates is preserved, the wiki administrators can optionally add their desired stylesheets for the results' representation. We will talk more about the templates in Section 6.4.1.

## 6.4 Storing and Presenting Service 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 content is one of the most challenging parts of the integration, due to the fact that each wiki engine has its own proprietary markup and database schema as explained in Section 2.1.2. In this section, we explain how our integration manages this important task.

### 6.4.1 Templating Mechanism

Following a successful NLP service execution by the Semantic Assistants server, results are passed to the servlet's broker module to be refined for the wiki. The broker module interprets the server's XML response and transforms the message into an array of Java objects. This is the ultimate extent that our integration can stay abstract from a wiki engine. From this point on, service results are transformed to wiki-specific markup and prepared for the templating mechanism. The templating mechanism is the process of embedding service results into wiki-specific 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 servlet and the wiki plug-in. The wiki helper module prepares the markup by placing results within their corresponding templates and storing them in the wiki's database. Once a wiki page is viewed by the user, the templates installed on the wiki will render the template markup to generate appropriate HTML representation.

For our MediaWiki integration, the Semantic Assistants templates are designed using the wiki's built-in Templates feature<sup>7</sup>. A MediaWiki template can contain parameters by putting a parameter name in three right and left curly brackets `{{{ }}}`. On each template invocation inside a wiki page, the template call is replaced by the template content, where the parameters with matching names are replaced by their values or with defaults. The Semantic Assistants plug-in leverages this useful feature of MediaWiki to create the following templates:

**Semantic Assistants Template.** This template is specifically designed to point out the start and end position of the Semantic Assistants service results in a wiki page's markup. It is important to separate the NLP service results from the page's original markup to avoid user confusion over what already existed in the wiki and what has been developed through the help of NLP pipelines, as postulated in Requirement #3. The Semantic

---

<sup>7</sup>MediaWiki Templates, <http://meta.wikimedia.org/wiki/Help:Template>



Assistants template has three parameters: (1) the name of the NLP service executed on the page content, (2) the wiki page name, and (3) the absolute URL of the wiki page that has been analyzed. Having these parameters in the template helps the user to visually identify the service that was executed and its input wiki page. Moreover, when the same service is re-run on the document, the wiki helper module can accurately locate the service’s result, as postulated in Requirement #3, and update its content, instead of adding a set of new results to the page.

```
1 <!-- Semantic Assistants Results Begin -->
2 {{{serviceName}}} on {{{doc}}} {{{url}}} (View)
3 ...
4 {{{serviceName}}} {{{doc}}}
5 <!-- Semantic Assistants Results End -->
```

Figure 32: MediaWiki Semantic Assistants service template markup

**Semantic Assistants Table.** This template is designed to present the annotations retrieved from a Semantic Assistants NLP service. According to the Semantic Assistants framework, each “Annotation” has five distinctive parts: (1) a *content* attribute that holds the annotation’s string value, (2) a *type* attribute that represents the annotation type, (3) a *start* attribute that shows the start offset of the annotation in the text, (4) an *end* attribute that shows the end offset of the annotation in the text, and (5) a *feature* attribute that contains additional information about the annotated entity. For each of these annotation attributes, there exists a parameter in the Semantic Assistants table template as shown in Figure 33.

When an array of annotations are passed to the wiki helper module, it embeds the annotations’ content inside the template markup, so that they can be stored in the database. For example, Figure 34 shows the generated table from a service execution in MediaWiki. In this example, the Semantic Assistants “Person and Location Extractor” service has been run on a “Kate Middleton” wiki page. The MediaWiki wiki helper module places the annotations inside the Semantic Assistants Table template and duplicates the row markup (see line 7 of Figure 33) for each generated

```

1  {| class="wikitable" style="height:50px"
2  ! width="200" | Content
3  ! width="80" | Type
4  ! width="50" style="text-align: center;" | Start
5  ! width="50" style="text-align: center;" | End
6  ! Features
7  |- valign="top" |{{{content}}} | style="text-align: center;" | ((Property:{{{type}}}|{{{type}}})) |
   style="text-align: center;" | {{{start}}} | style="text-align: center;" | {{{end}}} | {{{
   features}}}
8  |}

```

Figure 33: MediaWiki Semantic Assistants Table template markup

annotation.

**Person and Location Extractor** on [Kate\\_Middleton](#) [\(View\)](#)

Content	Type	Start	End	Features
Elizabeth Middleton	<a href="#">Person</a>	236	255	■ gender: female
Michael Francis	<a href="#">Person</a>	397	412	■ gender: male
Buckinghamshire	<a href="#">Location</a>	631	646	■ locType: region
Leeds	<a href="#">Location</a>	1896	1901	■ locType: city

Figure 34: Semantic Assistants annotations view in MediaWiki

**Semantic Assistants Block.** This template is designed for the Semantic Assistants “Boundless Annotations” type of service results, as described in Section 2.5.3. According to the Semantic Assistants framework, boundless annotations are similar to regular annotations, except that they apply to a document as a whole and do not have specific “start” and “end” offsets. While this annotation type bears the same structure as regular annotations, it is not appropriate to generate a table structure to represent them. Therefore, the wiki helper module assembles boundless annotations content into one string value and places them inside the Semantic Assistants block template. Figure 35 shows the template markup designed for MediaWiki.

Figure 36 shows the MediaWiki Semantic Assistants Block template

```

1 <table style="border: 2px dotted; border-color: #545454;
2 background-color: #F0F0F0; padding: .5em 1em; float: left;
3 margin-bottom: 2em; color: #000;">
4 <tr>
5 <td>{{{content}}}</td>
6 </tr>
7 </table>

```

Figure 35: MediaWiki Semantic Assistants Block template markup

generated from running the Semantic Assistants “Simple Summarizer” service on the same wiki page as the previous example.

**Simple Summarizer** on [Kate\\_Middleton](#) ([View](#))

Summary: She is the eldest of three children born to Carole Elizabeth Middleton (née Goldsmith), a former flight attendant and now part-owner of Party Pieces, a private company with an estimated worth of £30 million, and Michael Francis Middleton, who also worked as a flight attendant prior to becoming a flight dispatcher for British Airways, currently also an owner of Party Pieces. Her parents married on 21 June 1980, at the Parish Church of Dorney, Buckinghamshire and in 1987, founded Party Pieces, a mail order company that sells party supplies and decorations. Catherine has a sister, Philippa Charlotte, known as "Pippa" (born 1983), and a brother, James Middleton (born 1987).

Figure 36: Semantic Assistants Block preview in MediaWiki, showing a summary generated from a wiki page

### 6.4.2 Storing the Markup

The next important task is to store the results in the wiki, so that they become persistent and can be reused later on. This is important because, according to Requirement #17, the service execution can occur proactively and does not require user interaction. In such cases, service results, such as generated content, has to be stored in the wiki so that users can access them later. Since storing markup needs direct interaction with the wiki database, and each wiki has its own schema, this task is handled by the wiki helper module inside the servlet. The wiki helper class can use the wiki’s Java API, if offered by its engine, or rely on third party libraries to connect to the wiki database.

For the MediaWiki integration, our implementation reuses the Java Wiki Bot Framework<sup>8</sup>, an open-source third party library that provides methods to connect, modify and read collections of articles. This framework also provides the means to create wiki bots for batch processing of wiki content. Figure 37 shows how the bot is used to save NLP results to a wiki page.

```
1  /** Stores the markup to the specified wiki page.
2  * @param content content to write
3  * @param pageName name of the wiki page
4  * */
5  @Override
6  public void writeToPage(final String content, final String pageName) {
7      MediaWikiBot bot = new MediaWikiBot(iWikiAddress);
8      bot.login(iWikiUser, iWikiPass);
9      Article article = new Article(bot, pageName);
10     article.addText(content);
11     article.save();
12 }
```

Figure 37: Java Wiki Bot Framework used to write content to a wiki page

**Storing Semantic Metadata.** In Section 5.3.1, we described various ways of representing semantic metadata generated by NLP pipelines. We chose the semantic wiki markup approach for our MediaWiki integration, since the Semantic MediaWiki extension installed on its engine allows us to represent the semantic entities using a simple wiki markup.

In Semantic MediaWiki, a semantic annotation is defined in form of a “subject, predicate, object” triple, using `[[property::object]]` syntax. Using a wiki helper module, semantic results can be easily transformed to semantic markup using this syntax. For example, when an NLP service generates annotations, our wiki helper module will use the results objects to create semantic metadata: each entity found by the pipeline is annotated with its associated type. Therefore, if a “Person and Location Extractor” pipeline finds “John” as a person in a wiki page, the wiki helper module will transform it into `[[hasType::Person|John]]`. This markup

<sup>8</sup>Java Wiki Bot Framework, <http://jwbf.sourceforge.net/>

is then stored in the wiki to semantically annotate the page. Once the page is saved in the wiki's database, the Semantic MediaWiki will parse the provided semantic markup and transform it to triples using the RDF language. For example, Figure 38 shows how our previous example is exported to RDF by the Semantic MediaWiki engine.

```
1 <owl:ObjectProperty rdf:about="http://localhost/Wiki-Sandbox/index.php/  
   Special:URIResolverProperty-3AHasType">  
2 <rdfs:label>HasType</rdfs:label>  
3 <swivt:page rdf:resource="http://loompa.cs.concordia.ca/Wiki-Sandbox/index.php/  
   Property:HasType"/>  
4 <rdfs:isDefinedBy rdf:resource="http://loompa.cs.concordia.ca/Wiki-Sandbox/index.php/  
   Special:ExportRDF/Property:HasType"/>  
5 <swivt:wikiNamespace rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">102</  
   swivt:wikiNamespace>  
6 </owl:ObjectProperty>  
7  
8 <swivt:Subject rdf:about="http://localhost/Wiki-Sandbox/index.php/Special:URIResolver/  
   Person">  
9 <rdfs:label>Person</rdfs:label>  
10 <swivt:page rdf:resource="http://loompa.cs.concordia.ca/Wiki-Sandbox/index.php/Person"/>  
11 <rdfs:isDefinedBy rdf:resource="http://loompa.cs.concordia.ca/Wiki-Sandbox/index.php/  
   Special:ExportRDF/Person"/>  
12 <swivt:wikiNamespace rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">0</  
   swivt:wikiNamespace>  
13 </swivt:Subject>
```

Figure 38: RDF representation of semantic metadata generated by Semantic MediaWiki

## 6.5 Service Execution Flow

We have clarified the connection between system components and how results are presented to wiki users. In this section, we describe a scenario to illustrate the execution flow of NLP services and how users will eventually benefit from the system.

In this scenario, we have a user who wishes to extract all the named entities of a certain type from a wiki article. Our user's first step is to ask for the Semantic Assistants user interface by clicking on the plug-in menu as shown in Figure 31(b). Upon clicking, the browser creates an HTTP

request object and sends it to the Semantic Assistants Servlet. The servlet receives the request and asks the wiki helper module to create the right type of wiki object to connect to its database and retrieve the content of the article. At this point, the servlet still needs to get the Semantic Assistants user interface and embed it in the result JSP page before sending it back to the browser. Therefore, the servlet tells the user interface module to ask for available services from the broker module and dynamically generates a list of all available assistants, as depicted in Figure 24. Once the HTML representation of the list is ready, the servlet will assemble the JSP page and send it back to the browser. This sequence is visualized in Figure 39.

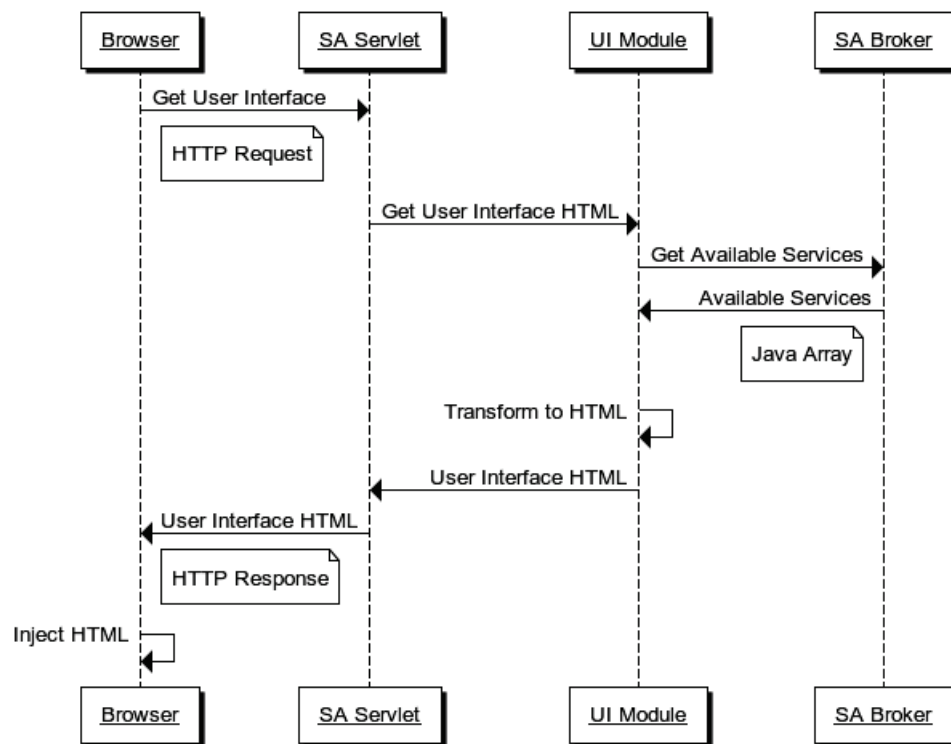


Figure 39: Communication flow for user interface generation

From the user's point of view, the proxy page presents the exact structure of the original wiki page, in addition to the Semantic Assistants user interface. Using the list of available services, our user can now decide which service to run by reading each service description, adjusting the input and eventually requesting its execution. Again, the browser sends another request to the servlet, asking for the execution of the user-selected

service on the provided input. The servlet then asks the wiki helper module to retrieve the content of the wiki page and prepares it for service execution. In this step, all the noise from a wiki page can be discarded before sending the textual content to the NLP pipeline. The broker module is the next component to receive the wiki content and make the actual execution method call to the Semantic Assistants server. Once the execution is finished, the broker module will transform the Semantic Assistants Server XML response to an array of Java objects with the help of the Semantic Assistants CSAL libraries. The array of results is then sent to the wiki helper module to be transformed to a language understandable by the wiki engine, namely, wiki markup. The helper module stores the markup in the wiki database and informs the servlet that the execution is finished. Finally, the servlet embeds a “completion of service” message inside an HTTP response object and sends it to the browser to be displayed to the user. The communication flow for a service execution is illustrated in Figure 40.

## 6.6 Résumé

This chapter described the implementation process of the system design described in the previous chapter. In Sections 6.1 and 6.2, we looked at the essential components of our architecture and defined how their design was translated into concrete implementation decisions. Then we elaborated the implementation details of the Semantic Assistants Wiki plug-in and how the results from NLP services are presented to a wiki. We finally closed the chapter with an example scenario of a service execution and illustrated the communication flow of our system components. In the next chapter, we will put our system into practice and evaluate its practicability in various domains.

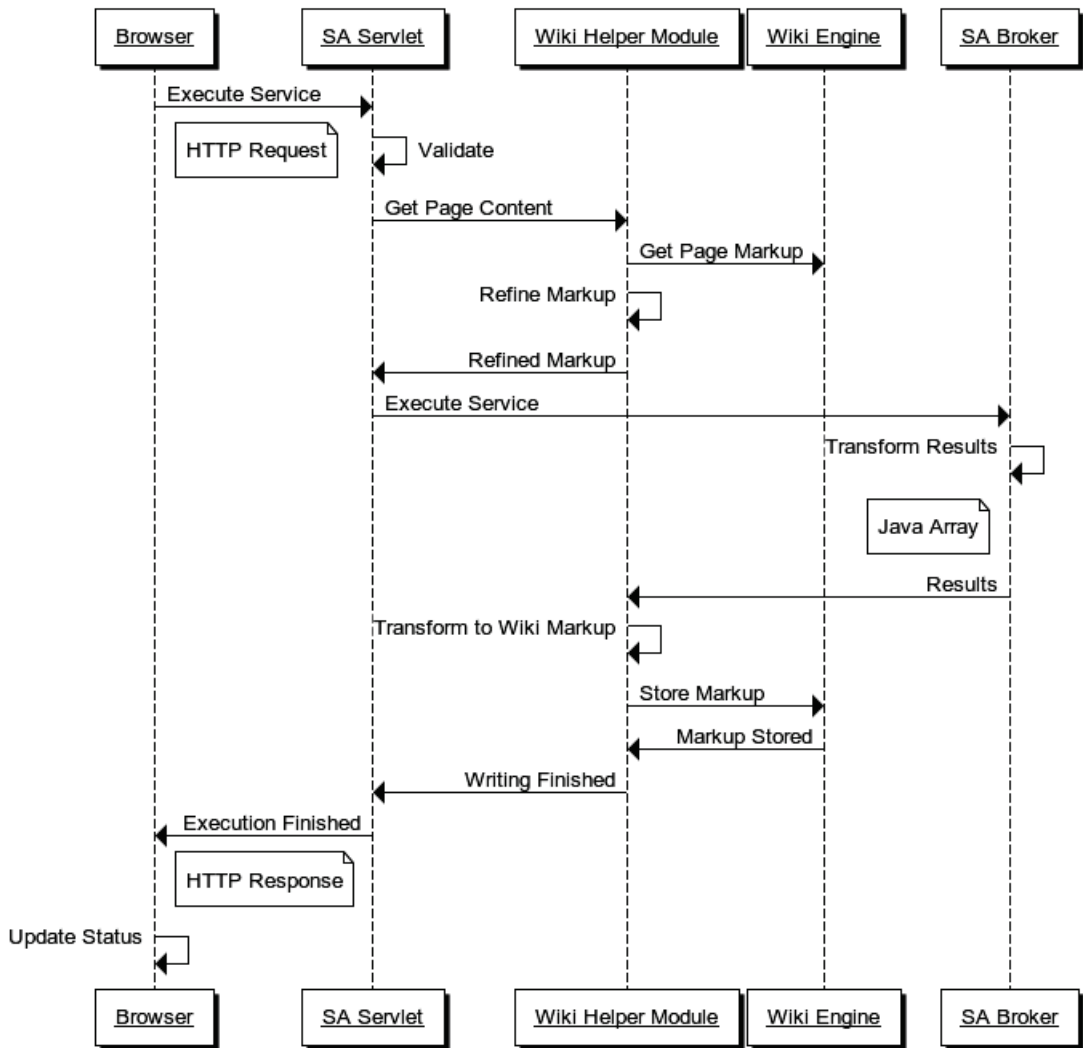


Figure 40: Communication flow for service invocation



# Chapter 7

## Evaluation

Every scientific work needs to be evaluated to prove its feasibility and usefulness. In this research work, we proposed an architecture to integrate wikis with NLP techniques, claiming that enriching wiki content with metadata derived from NLP techniques can aid its users with content development, organization and retrieval. In this chapter, we evaluate our Wiki-NLP integration in the light of our goals by applying it to real-world projects from various domains.

### 7.1 Methodology

In our evaluation process, we will assess our architecture along three dimensions:

**Practicability.** In this dimension, we want to evaluate whether the Wiki-NLP architecture developed throughout this thesis can be applied to a concrete scenario and all the analysis workflow phases described in Section 5.2 can be successfully carried out to analyze the content of a wiki. Towards this end, in Scenario 1, described in Section 7.2, we will use our integration on a wiki that contains a digitized version of an encyclopedia. The goal in this process is to invoke an NLP service on the wiki content and have the results retrieved and stored in the wiki's database. In addition, by invoking different NLP services on the wiki, we will demonstrate

the NLP-independence feature of our architecture as postulated in Requirement #2.

**Usability.** Following our first scenario, in Section 7.3, we will evaluate whether the Wiki-NLP architecture can be used by wiki users who do not have a profound knowledge of NLP concepts. For this, in Scenario 2, we asked the students of one undergraduate and one graduate level software engineering class to use the wiki for a specific task. Then, at the end of the experiment, we inquired about their level of knowledge in the NLP domain and the ease-of-use of our integration.

**Effectiveness.** In Section 7.3, we also examine whether the integration of NLP services into a wiki system will ultimately improve the quality of its content. Towards this end, in Scenario 2 we asked the students to develop wiki content with and without the help of NLP services. At the end, we will compare the number of defects found in each revision of the wiki's content, which allows us to estimate the impact of NLP support on the wiki content's quality.

**Efficiency.** In our last scenario, described in Section 7.4, we evaluate whether the integration of NLP techniques in wikis via our architecture is efficient, in terms of the time needed by wiki users to fulfill their information needs. For this, we asked two graduate researchers to collaboratively use a wiki for the purpose of biomedical literature curation. During the experiment, we will keep track of the time spent on the curation process and compare it to the time needed to do the same task without the help of NLP services.

Table 7 shows the mapping of our evaluation scenarios to the requirements defined in Chapter 4. The only requirement that is not explicitly evaluated in this chapter is the proactive behaviour of the system. That is because since the Wiki-NLP integration is implemented as a RESTful web application, irrespective of how the HTTP request is formed – user-generated or by a cron job – it is naturally able to respond accordingly.

Table 7: Mapping of requirements to evaluation scenarios

Requirement	Scenario 1	Scenario 2	Scenario 3
Seamless Integration	✗	✓	✓
Change Visibility	✗	✓	✓
Organizing Wiki Content	✗	✓	✓
Finding Wiki Content	✓	✓	✓
Low Learning Curve	✗	✓	✓
Collection-based Analysis	✓	✓	✓
Easy Deployment	✗	✓	✗
NLP Service Independence	✓	✓	✗
Wiki System Independence	✓	✗	✗
Flexible Request Handling	✓	✓	✓
Read Content from Wiki	✓	✗	✗
Write Content to Wiki	✓	✗	✗
Proactive Service Execution	✗	✗	✗

## 7.2 Wiki-based Cultural Heritage Data Management

In Section 2.2.5, we described how wikis are used as cultural heritage knowledge bases. The Durm [WKKL11] project is an example of this category, carried out from 2004 to 2006 at the University of Karlsruhe with the goal of investigating the use of semantic technologies for cultural heritage data management. The project uses a MediaWiki instance that contains a complete digitized version of one volume of the *Handbuch der Architektur*<sup>1</sup> – a 100-year old historical encyclopedia for architecture. The Durm wiki offers three NLP services through the use of a bot written using the Python Wikipedia Robot Framework<sup>2</sup>: (1) a German Index Generation service [WKKL10] that creates a classical back-of-the-book index of the wiki content by grouping extracted terms into groups of noun phrases, while

<sup>1</sup>Handbook on Architecture

<sup>2</sup>Python Wikipedia Robot Framework, <http://pywikipediabot.st.net>

keeping the track of their corresponding wiki page, (2) an Automatic Summarization service that provides contrastive or focused summaries of arbitrary length to users, and (3) an Ontology Population service that annotates text tokens found in the wiki content with a corresponding ontology class, thus making the wiki content machine-accessible and available for semantic queries. Figure 41 shows the workflow between document storage, retrieval and NLP analysis in the Durm wiki.

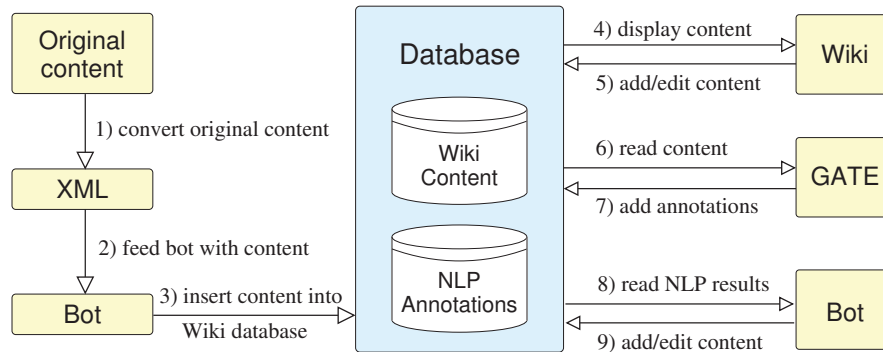


Figure 41: The Durm wiki workflow [WKKL11]

In this scenario, we aim to prove that the same NLP capabilities can be provided to Durm wiki users through our general architecture, independent of its concrete engine, rather than hard-coding such capabilities in the wiki itself. We also aim to demonstrate that by using our Wiki-NLP integration, the Durm wiki can benefit from a wider range of services, without the need to modify the wiki engine or the bot used in the original approach.

### 7.2.1 Evaluation Scenario

The corpus developed during the Durm project is released under the GNU Free Documentation license and is available for download<sup>3</sup>. We acquired the corpus and imported it in a MediaWiki instance that was set up for this evaluation purpose. Then, we installed the Semantic Assistants plugin on the wiki and invoked the same German Index Generation service described above on a sample wiki page. In addition to this service, we also

<sup>3</sup>The Durm Corpus, <http://www.semanticsoftware.info/durm-corpus>

invoked a simple named entity recognition service on a sample wiki page that extracts entities of type *Person* and *Location*. Figure 42 shows the results as a table of annotations written into the page.

The screenshot shows a Wiki-NLP page titled "Wände aus Eisen und Mörtel". The page content includes a table of annotations for the entities "Person" and "Location". The table has columns for Content, Type, Start, End, and Features. The extracted entities are G. Luther (Person), Berlin (Location), and München (Location).

Content	Type	Start	End	Features
G. Luther	Person	431	440	gender: -
Berlin	Location	444	450	locType: city
Berlin	Location	698	704	locType: city
München	Location	915	922	locType: city

Figure 42: Person and Location Extractor service results in DurmWiki

## 7.2.2 Results

The results of the index generation service invoked during the evaluation are stored in our online demo wiki as shown in Figure 43, and resembles the same page that had been created by their bot, on the original Durm wiki public version [WKKL11]. Therefore, it proves that the same NLP functionality is achieved through our generic architecture, without any modifications on the wiki engine. Also, Figure 42 shows that, in addition to the index generation service, we successfully invoked a named entity recognition service. This supports our claim of having an NLP-independent integration

architecture, where new NLP services can be added and discovered dynamically through the Semantic Assistants service-oriented architecture.



Figure 43: Output of German Durm Indexer pipeline in DurmWiki

## 7.3 Wiki-based Collaborative Software Requirements Engineering

Previously, we described in Section 2.2.3 how wikis are used in the software requirements engineering domain. In this scenario, we aim at evaluating the usability of our system by testing the Wiki-NLP integration in the context of a collaborative software requirements engineering process. For this purpose, a MediaWiki instance, called ReqWiki, was set up as a collaborative platform for the documentation of a hypothetical Android<sup>4</sup>

<sup>4</sup>Android, <http://developer.android.com/guide/basics/what-is-android.html>

application requirements specifications. To customize the wiki for this scenario, the Software Requirements Specification (SRS) templates document provided to students in the course material, based on the Unified Process (UP) [Lar04], were divided into three parts and placed in separate wiki pages: (1) a “Vision” page to define the product position, stakeholders, assumptions, dependencies, needs and features, (2) a “Use Case” page to define actors, goals, use cases, and (3) a “Supplementary Specification” page to define functional and non-functional requirements, standards, legal notes, test cases and traceability links. For each of these pages, a “discussion” page – provided by the MediaWiki engine – was also available for students to discuss contradictory ideas and leave notes for other stakeholders.

To reduce the learning curve of using the wiki, we installed the Semantic Forms<sup>5</sup> extension on its engine to allow students entering and editing wiki content using HTML forms, instead of working with raw markup. Figure 44 shows a wiki form used to create a problem statement, similar to the table found in the document templates provided to the other group of students using traditional word processor applications. Then we asked a graduate student from a Software Engineering Case Study (SOEN 6951) course to transform the document templates, e.g., product position or use case tables, to MediaWiki template markups for a user-friendly presentation of the system entities. In addition, an ontology for requirements specifications documents was designed and reflected in the wiki by creating Semantic MediaWiki-style relationships between domain entities in the wiki, e.g., “Goal” *belongs to* “Actor”. Using this ontology, we were able to embed pre-defined semantic queries to create traceability links between various entities of the system in the three wiki pages described above, thus helping the students with the integrity of their SRS documents.

---

<sup>5</sup>Semantic Forms extension for MediaWiki, [http://www.mediawiki.org/wiki/Extension:Semantic\\_Forms](http://www.mediawiki.org/wiki/Extension:Semantic_Forms)



The image shows a screenshot of a web-based form for editing a problem statement in ReqWiki. At the top, there is a navigation bar with buttons for 'page', 'discussion', 'edit with form' (which is highlighted), 'edit', 'history', 'delete', 'move', 'protect', 'watch', and 'refresh'. Below the navigation bar, the title of the form is 'Edit FormProblem: Difficulty comparing nutrition values of similar aliment-products'. The form is divided into three main sections: 'Affects:', 'Impact:', and 'Successful Solution:'. The 'Affects:' section contains a text input field with the value 'System Users'. The 'Impact:' section contains a list of three items: '- Inaccurate or raw-estimated food consumptions', '- Unable to make balanced food purchases to optimize healthy eating habits.', and '- Poor nutrition'. The 'Successful Solution:' section contains a list of two items: '- Clarify nutrition-fact-labels' and '- Food classification'.

Figure 44: A sample form in ReqWiki

### 7.3.1 Evaluation Scenario

The ReqWiki system was eventually introduced to one undergraduate level (SOEN 342) and one graduate level (SOEN 6481) software engineering class. Students were asked to voluntarily use the wiki system for their course assignment, i.e., developing an Android application SRS. Several instances of ReqWiki were set up for a total of 22 students, teamed up in groups of one or two. The Semantic Assistants plug-in was also installed on each wiki and students were asked to perform a quality analysis on their third assignment before submitting it. For this task, we provided various domain-specific NLP services, as well as general Information Extraction services as follows:

**Writing Quality Assessment**, which performs grammar and spell checking on the content and provides suggestions for improvements. This service provides the capabilities of the After The Deadline [Mud10] tool and helps students to find spelling and grammatical mistakes, as well as passive voice, in their requirements specifications.

**Readability Assessment**, which measures the readability of a given text based on standard readability metrics, like Flesch and Kincaid [DuB06]. This service provides the students with an overall readability score of their assignment. The result score indicates how hard to read and comprehend their assignment is for other stakeholders, e.g., their teammates and markers.

**Requirements Quality Assurance**, which is a service developed based on



the NASA requirements quality metrics [Lap09]. It detects SRS defects like *Options*, *Directives* or *Weak Phrases* in a document. By using this service, students have the chance to find these defects in their assignment and correct them, resulting in a higher quality SRS document.

**English Durm Indexer**, which creates a noun-phrase index of the wiki content. This service uses MuNPEX<sup>6</sup>, an open-source tool that groups words into noun phrases. Students can compare the result of this service to their “Glossary” section and check its completeness.

Figure 45 shows how the results of the NLP services are presented to the students in ReqWiki. A sample use case table is shown on top of the picture. Two NLP services, namely Readability Assessment and Writing Quality, have been invoked on this use case and the results are presented at the bottom of the page.

### 7.3.2 Results

Nielsen and Landaur mathematically prove in [NL93] that the detection of usability problems as a function of the number of tested users is well-modeled as a Poisson process. They suggest that for a medium-size project, at least 16 evaluations are needed, at which optimal cost-benefit ratios are obtained. Therefore, at the end of the course, all the ReqWiki users were provided with a questionnaire (see Appendix C) to evaluate their experience using the ReqWiki system for their assignments in terms of its usability, as well as the quality of the NLP services provided. We explicitly asked them about the user-friendliness of the Semantic Assistants user interface and its features in detail, e.g., forms, templates, pre-defined queries.

At the end of the questionnaire, we asked the students whether, given the experience of using ReqWiki, they would use ReqWiki-like systems with semantic support for requirements engineering tasks in the future

---

<sup>6</sup>Multi-Lingual Noun Phrase Extractor <http://www.semanticsoftware.info/munpex>

## UC/Manage Tasks

<b>Description</b>	The manager receives a customer service request. The manager directs the operation for creating, updating, deleting and querying a task. Some operations use either the automatic or manual task assignation functionality that were defined in the Supplementary Specification document.
<b>Level</b>	user-goal
<b>Primary Actor</b>	A / Manager
<b>StakeHolders</b>	Manager, Senior technician, Junior technician
<b>Interests</b>	The manager wants to create, update, delete, query task and assigns it to a number of technicians. The senior and junior technician need to be notified every time that a new task is assigned to them.
<b>Pre-Conditions</b>	The manager must be identified and authenticated in the application
<b>Success end condition</b>	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	<ul style="list-style-type: none"> <li>problem: The sentence has been detected as passive, and can be improved by changing the verb phrase (is created)</li> </ul>

Writing Quality on UC/Manage\_Tasks ([View](#))

Content	Type	Start	End	Features
The tasks is	Grammar	686	698	<ul style="list-style-type: none"> <li>problem: Wrong Auxiliary Verb</li> <li>suggestion: The task is</li> </ul>

Figure 45: Presentation of SRS Defects in ReqWiki

or if they would resort to traditional wikis or word processors. Figure 47 presents the results gathered from the questionnaire feedback.

The two pie charts on the right side of Figure 47 show that an average of 50% percent of the students voted the Semantic Assistants user interface to be “Very Easy” or “Easy” to use, while the two pie charts in the middle show that an average of 80% of the students had no or mere

**What is your level of experience in the area of Natural Language Processing?**

*Choose one of the following answers*

- Previous academic experience (e.g., you have taken related courses)
- Previous industrial experience (e.g., you have worked in this area)
- Both academic and industrial experience
- None

Figure 46: A sample question from the students questionnaire

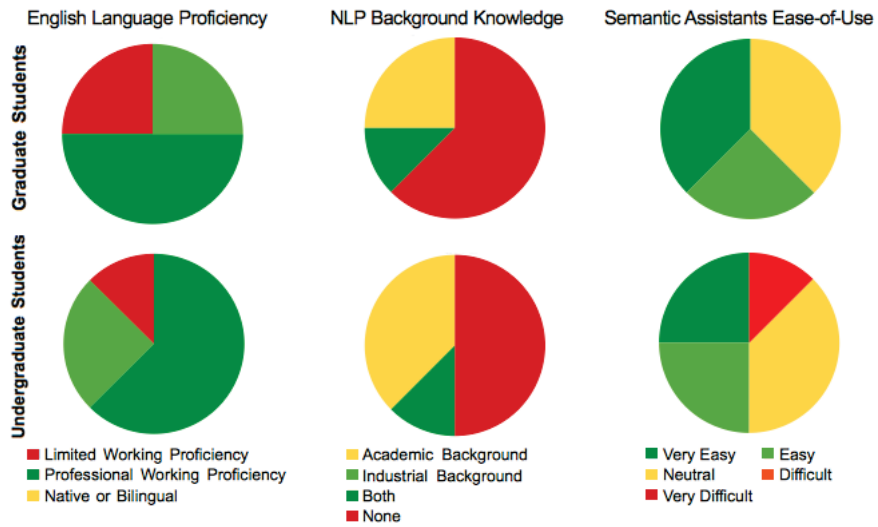


Figure 47: Feedback statistics from students questionnaire

textbook knowledge in the NLP domain. Figure 48 presents the correlation between the students' level of NLP knowledge and their choice on the system's usability. Among the feedback data, only one student rated the Semantic Assistants user interface as "Very Difficult" to use and indicated that the reason for his or her choice was a browser incompatibility issue, meaning that he or she was not able to view or use the integration functionality. Nevertheless, we can conclude that the seamless integration of NLP services inside the wiki provided the chance for wiki users to benefit from NLP techniques, without having a profound knowledge in this area.

Finally, all of the students who used ReqWiki during the course indicated that in the future, they are likely to use a wiki system enhanced with

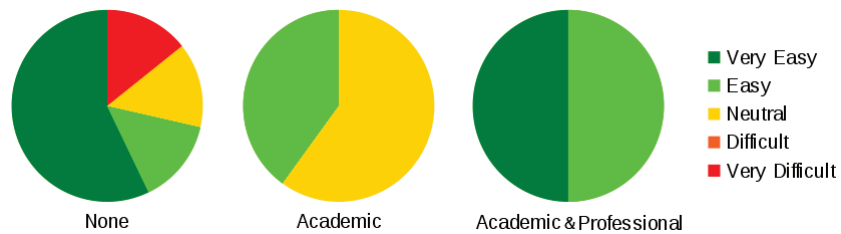


Figure 48: System usability feedback based on the students NLP knowledge level

semantic support and NLP techniques for similar efforts, rather than a traditional wiki or word processors. Therefore, an acceptance rate of 100% for future use further proves the usability and helpfulness of our Wiki-NLP integration in real-world software requirements engineering tasks.

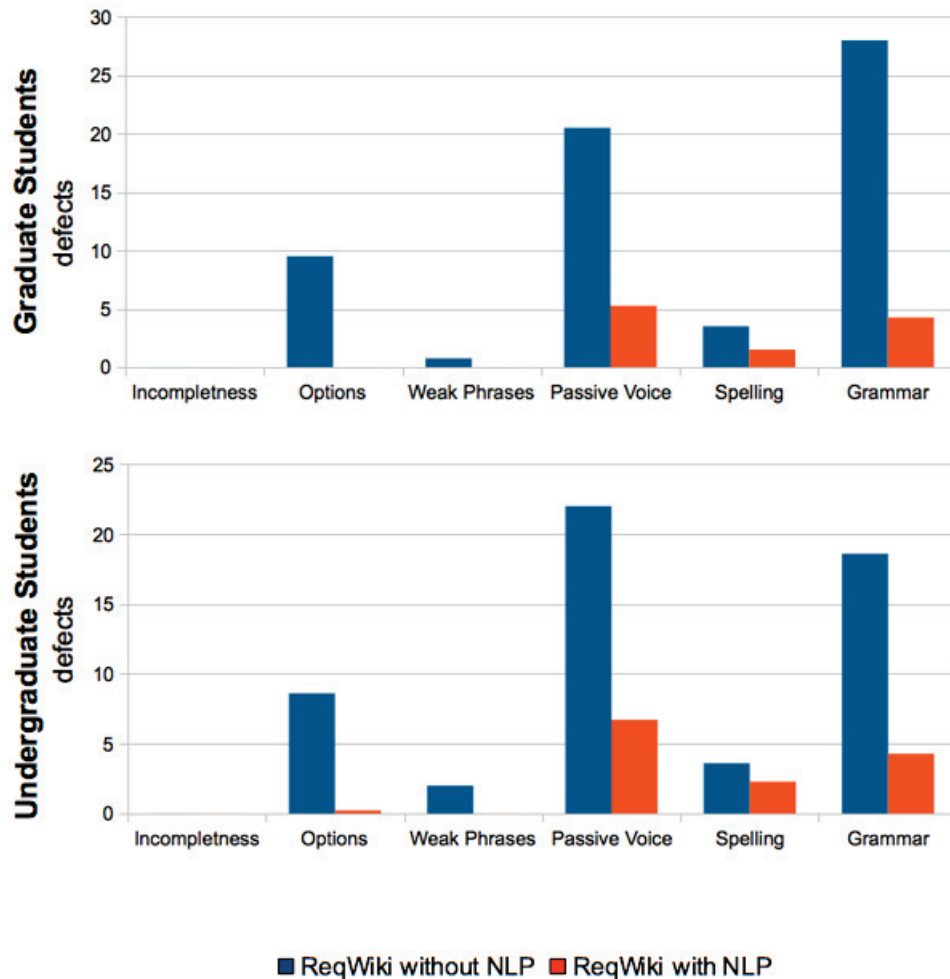


Figure 49: Average number of defects found in assignments

As for evaluating the effectiveness of our integration, we calculated the number of defects found in the students' Use Case documents, when they had no NLP support on the ReqWiki (Assignment #2) and performed a similar calculation for the same documents after they had been analyzed with ReqWiki's available NLP services for quality assessment (Assignment #3). Figure 49 shows the average number of defects found in assignments, both before and after the help of NLP services. It can be seen that the use of NLP

services significantly decreased the number of defects found in the second revision of the Use Case documents.

From our findings in the gathered data, we can conclude that: (1) despite the students' low level of knowledge in the NLP domain, all the students were able to use our seamless integration and almost half of the students ranked the Semantic Assistants user interface to be easy-to-use, and (2) by comparing the number of defects found in SRS documents before and after using NLP services, we proved that the integration of NLP services into an ordinary software engineering task carried out via a wiki interface can improve the quality of its content.

## 7.4 Wiki-based 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<sup>7</sup> and provided to *curators* – domain experts – who will manually browse through the data and extract domain knowledge from the literature.

Our third evaluation scenario took place within Concordia's Centre for Structural and Functional Genomics<sup>8</sup> in the context of the Genozymes project<sup>9</sup>. The Genozymes project comprises a team of multi-disciplinary scientists, including biologists, biochemists and bioinformaticians, with the ultimate goal of producing breakthroughs in genomics research that will transform green waste into renewable and alternative chemicals and fuels. Among them, a team of biologists and bioinformaticians are currently working on the curation of characterized glycoside hydrolases<sup>10</sup> of fungal origin from the domain literature [MPW<sup>+</sup>11]. The curators use BRENDA [CSG<sup>+</sup>09] – a comprehensive enzyme information system – and

---

<sup>7</sup>PubMed, <http://www.ncbi.nlm.nih.gov/pubmed/>

<sup>8</sup>Concordia's Centre for Structural and Functional Genomics, <http://genomics.concordia.ca/>

<sup>9</sup>The Genozymes Project, <http://www.fungalgenomics.ca/>

<sup>10</sup>family of enzymes used to break down plant cell walls

the PubMed website to find related literature references. Once the literature is acquired, curators evaluate the papers for curation, typically by reading the abstract. Then, for all the selected papers, curators read their full text and extract entities of interest. Each extracted entity is then inserted into a spreadsheet as a new record. Further related data, such as gene names, gene IDs and species, are gathered by curators from published articles that meet their criteria of characterized glycoside hydrolases. Finally, the extracted entities in the spreadsheet are organized in a searchable database called mycoCLAP [MPW<sup>+</sup>11], which has an online query interface<sup>11</sup>. Figure 50 illustrates the curation workflow practiced in this project.

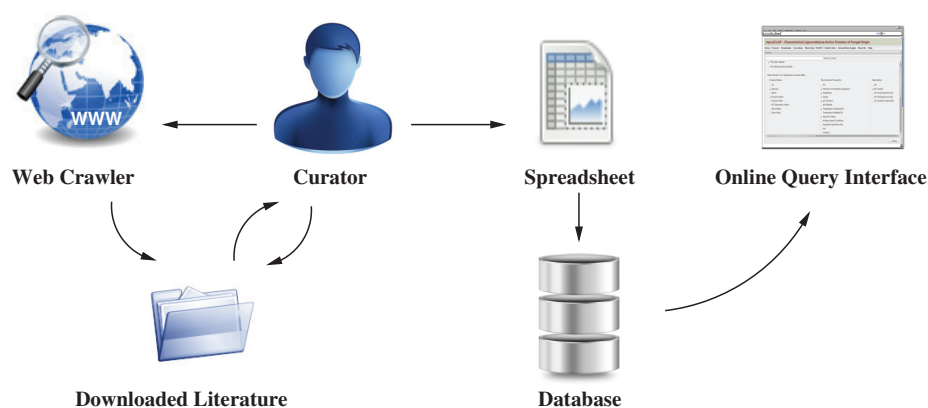


Figure 50: Manual curation workflow for biomedical literature

### 7.4.1 Evaluation Scenario

The manual curation approach practiced in CSFG is an expensive and time-consuming task. In addition, resource management, e.g., managing downloaded literature and removing duplicate files, is frequently reported as a problem. Recently, ontological NLP analysis pipelines have been developed to help curators spend less time on mining the literature, while providing richer and semantically related results: *mycoMINE* [MMM<sup>+</sup>11] is

<sup>11</sup>Characterized Lignocellulose-Active Proteins of fungal origin, <http://mycoclap.fungalgenomics.ca>

such a pipeline developed in the Semantic Software Lab that provides information about lignocellulose entities found in the domain literature and detects entities such as pH, Temperature and Kinetic Assay Conditions, Enzymes and Substrates, as well as Organisms through its OrganismTagger [NKBW11] component. For our scenario, a MediaWiki instance, called GenWiki, was set up for the curators with the goal of helping them spend less time on the selection and curation of papers. GenWiki was then pre-filled with literature related to characterized glycoside hydrolases of fungal origin. For each paper, we put the full text as well as its abstract into individual wiki pages, as shown in Figure 51.

The screenshot shows a MediaWiki page for PMID: 20709852. The page title is "Characterization of a Cellobiohydrolase (MoCel6A) Produced by *Magnaporthe oryzae*". The authors listed are Machiko Takahashi, Hideyuki Takahashi, Yuki Nakano, Teruko Konishi, Ryohei Terauchi, and Takumi Takeda. The affiliations include Iwate Biotechnology Research Center, Kitakami, Iwate 024-0003, Japan, and University of the Ryukyus, Department of Bioscience and Biotechnology, Faculty of Agriculture, 1 Senbaru Nishihara, Okinawa 903-0213, Japan. The abstract text describes the characterization of MoCel6A, a GH-6 family cellobiohydrolase, and its activity on various substrates like phosphoric acid-swollen cellulose (PSC),  $\beta$ -glucan, and cellooligosaccharide derivatives. It also mentions the effect of a histidine tag and the deletion of the cellulose binding domain (CBD) on the enzyme's activity.

Figure 51: A wiki page containing a full-text paper

The Semantic Assistants plug-in described in Section 6.3 was also installed on the GenWiki engine and the functionality of the system was introduced to the curators during a meeting. For the evaluation, the curators were assigned credentials on the wiki and asked to keep track of the time spent on selection and annotation of wiki pages with the help of



the mycoMINE pipeline that was accessible through the Semantic Assistants interface. Figure 52 shows how the manual curation workflow was changed by our Wiki-NLP integration. In this approach, literature is provided to the curators via a wiki interface that allows them to annotate it with NLP pipelines within the wiki, as shown in Figure 53.

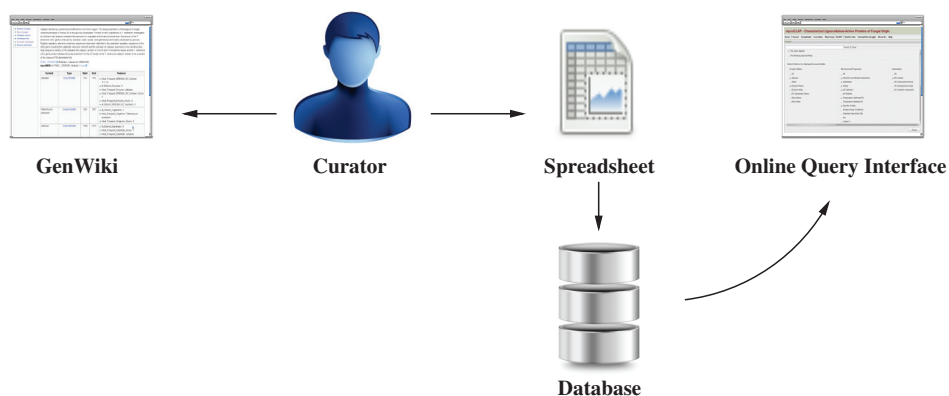


Figure 52: GenWiki-assisted curation workflow for biomedical literature

activities using MoCel6A and *Trichoderma reesei* cellobiohydrolase (TrCel6A), which were prepared in *Aspergillus oryzae*. MoCel6A showed increased hydrolysis of cellopentaose used as a substrate in the presence of 292 mM cellobiose at pH 4.5 and pH 6.0, and enhanced activity disappeared at pH 9.0. In contrast, TrCel6A exhibited slightly increased hydrolysis at pH 4.5, and hydrolysis was 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
In contrast, TrCel6A exhibited slightly increased hydrolysis at pH 4.5, and hydrolysis was severely inhibited at pH 9.0.	pH	1847	1967	<ul style="list-style-type: none"> <li>pH_alias: In contrast, TrCel6A exhibited slightly increased hydrolysis at pH 4.5, and hydrolysis was severely inhibited at pH 9.0.</li> </ul>
cellobiohydrolase	Enzyme	89	106	<ul style="list-style-type: none"> <li>enzyme_alias: cellobiohydrolase</li> <li>BRENDA_SystematicName: 4-beta-D-glucan cellobiohydrolase</li> <li>BRENDA_ECNumber: 3.2.1.91</li> <li>abbreviation_alias: -</li> <li>google_search: <a href="http://www.google.com/search?q=cellobiohydrolase">http://www.google.com/search?q=cellobiohydrolase</a></li> <li>BRENDA_RecommendedName: cellulose 1,4-beta-cellobiosidase</li> <li>SwissProt_ID: O68438</li> <li>BRENDA's page: <a href="http://www.brenda-enzymes.org/php/result_flat.php4?ecno=3.2.1.91">http://www.brenda-enzymes.org/php/result_flat.php4?ecno=3.2.1.91</a></li> </ul>

Figure 53: Presentation of NLP-generated annotations in GenWiki



## 7.4.2 Results

As described earlier in this chapter, the ultimate goal of this evaluation scenario is to assess the efficiency of the Wiki-NLP integration in terms of the time that the curators need to extract knowledge from the literature and import it to the mycoCLAP database.

In [MMM<sup>+</sup>11] a similar research has been conducted that provides an average time for a full paper curation process. This publication uses the same curators as our scenario, as well as a comparable dataset (a corpus of 10 papers) and therefore, is a valid dataset to be compared to the results of our evaluation. Table 8 shows the results reported by our curators using ReqWiki for literature curation.

Table 8: GenWiki-assisted literature curation time

Paper	Abstract Selection	Full Paper Curation
PMID: 12565856	10 sec.	Rejected
PMID: 12763033	10 sec.	Rejected
PMID: 12567807	15 sec.	31 min.
PMID: 15006424	30 sec.	56 min.
PMID: 15294290	30 sec.	21 min.
PMID: 15555935	30 sec.	21 min.
PMID: 15716038	15 sec.	34 min.
PMID: 19590866	15 sec.	21 min.
PMID: 20143777	15 sec.	22 min.
PMID: 20591661	15 sec.	24 min.
PMID: 20709852	15 sec.	41 min.
PMID: 21626020	15 sec.	41 min.
PMID: 21948841	15 sec.	71 min.
DOI: j.procbio*	15 sec.	Rejected
DOI: j.enzmictec**	30 sec.	25 min.
Median	15 sec.	28 min.
Average	18.3 sec.	34 min.

\* Complete reference ID is DOI: 10.1016/j.procbio.2007.01.007

\*\* Complete reference ID is DOI: 10.1016/j.enzmictec.2006.03.017

The “Abstract Selection” column in Table 8 contains the time that is needed for the curator to read a paper abstract to decide whether the paper

should be considered for curation and the “Full Paper Curation” column states whether the paper was selected, and if so, how much time was spent on the curation process. We used the times in this table to calculate the time needed for both selection and curation of a paper and compare it to the results found in [MMM<sup>+</sup>11] in Table 9. It can be seen that the time for selection and curation of papers in the wiki, with the help of the mycoMINE NLP pipeline, was reduced by 50% and 9.33%, respectively.

However, in the data shown in Table 8, the curation time for *PMID: 21948841* is 71 minutes and deviates markedly from other members of the sample. Therefore, if we exclude this outlier from our sample data, the average curation time will be 28 minutes and thus the manual curation time will be reduced by 18.33% – almost twice the first average.

Therefore, Table 9 supports our evaluation hypothesis that the seamless integration of NLP services inside a wiki is indeed efficient, in terms of the time needed to fulfill the wiki users’ information needs.

Table 9: Curation time of papers with different levels of semantic support

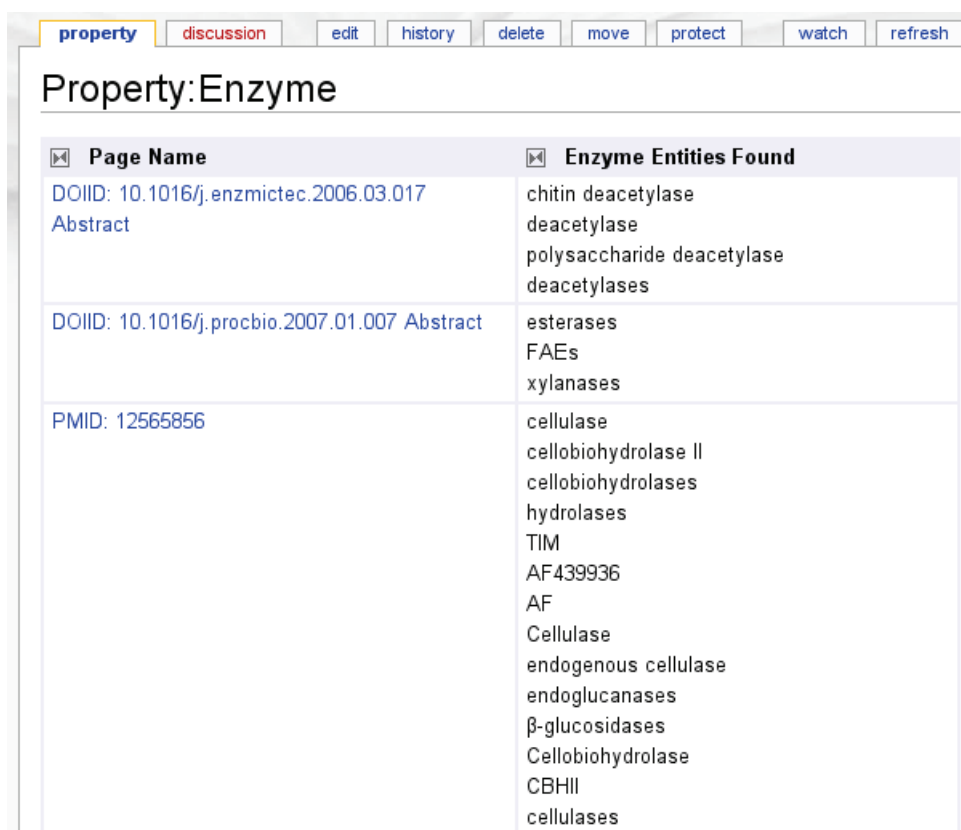
<b>Abstract Selection</b>		<b>Full Paper Curation</b>	
Manual	GenWiki	Manual	GenWiki
1 min.	20 sec.	37.5 min.	30.63 min.

**Semantic Metadata.** Our Wiki-NLP integration in GenWiki is also able to produce semantic metadata from the annotations found by the mycoMINE pipeline. The semantic results further help curators with querying and finding entities in the wiki – a feature that is missing from the CSFG workflow. Currently, the only possible way of querying the knowledge extracted from literature is to use the mycoCLAP website in order to perform a keyword-based search. However, in GenWiki, since annotations are also represented with semantic markup, they can be queried using Semantic MediaWiki inline queries, like the one showed in Figure 54, or exported as RDF triples to be used by external applications. Figure 55 shows how curators see the results of a semantic query for all the entities of type “Enzyme” in the wiki. They can directly navigate to an enzyme’s wiki page by

clicking on its title in the table.

```
1  {{#ask: ((hasType::Enzyme))
2    |?Enzyme=Enzyme Entities Found
3    |format=table
4    |headers=plain
5    |default=No pages found!
6    |mainlabel=Page Name
7  }}
```

Figure 54: The Semantic MediaWiki inline query for all enzyme entities in GenWiki



Page Name	Enzyme Entities Found
<a href="#">DOIID: 10.1016/j.enzmictec.2006.03.017 Abstract</a>	chitin deacetylase deacetylase polysaccharide deacetylase deacetylases
<a href="#">DOIID: 10.1016/j.procbio.2007.01.007 Abstract</a>	esterases FAEs xylanases
<a href="#">PMID: 12565856</a>	cellulase cellobiohydrolase II cellobiohydrolases hydrolases TIM AF439936 AF Cellulase endogenous cellulase endoglucanases $\beta$ -glucosidases Cellobiohydrolase CBHII cellulases

Figure 55: Semantic query results for all enzyme entities in GenWiki, generated by NLP services

## 7.5 Résumé

This chapter described the evaluation process of our Wiki-NLP integration. We evaluated our architecture along four dimensions: Practicability, Usability, Effectiveness and Efficiency. We started each scenario by briefly describing the domain and putting the integration in context. The MediaWiki engine, enhanced with our Wiki-NLP architecture that was described in Chapter 6, was used in our scenarios. Finally, the results gathered from our studies proved not only the practicability of the Wiki-NLP integration, but that it indeed brings a measurable value to wiki end-users. In the next chapter, we will summarize this research work and provide some thoughts for future work.

# Chapter 8

## Conclusions and Future Work

In this chapter, we provide a summary and the conclusion of our research work, by describing the progress made towards the goal of providing wiki users with the benefit of NLP techniques. Also, we will suggest some research directions to be undertaken in the near future.

### 8.1 Summary

Wikis are popular web-based applications, whose users can collaboratively add, edit, or delete content via a Web browser, using a simplified markup language. They have been widely adopted in various domains as a light-weight and easy-to-use information management tool. The aim of this thesis was to develop an architecture for aiding wiki users in time-consuming and labor-intensive tasks, through the help of automatic text mining services. We first performed a literature survey on related existing work in this area, followed by an elaborated effort on investigating typical use cases of wikis in real-world scenarios. During this investigation, we gathered a comprehensive list of reported problems in working with wikis and chose the most prominent ones to derive our initial system requirements. We performed a detailed requirements analysis from the perspectives of wiki end-users, the system as a whole, as well as wiki developers, i.e., wiki administrators or engine developers who wish to enhance their wiki systems with NLP capabilities.

The NLP services of our solution are provided by the Semantic Assistants project, an open source service-oriented architecture that brokers NLP pipelines, developed based on the GATE framework, as web services. Using this architecture provided us with the advantage of service independence, and automatically fulfilled a number of our integration requirements. The remaining ones were then translated to concrete design decisions in Chapter 5.

One of the main challenges of this research work was the lack of a standard architecture and markup syntax for wikis. Therefore, no definite architecture could be easily developed that would encompass the variety of every existing wiki engine. Consequently, in our design chapter, we thoroughly analyzed how various juxtapositions of our system components can realize the ultimate goal of the integration, and examined each design alternative against the system requirements. Eventually, we chose a collaborative approach, combining the advantages of our design alternatives into a cohesive architecture that provides wiki systems with NLP services, while keeping the wiki dependency as small as possible. Our contribution was implemented as an abstraction layer between the Semantic Assistants architecture and the wiki system component, realized as a proxy server using J2EE Servlets and a number of Web 2.0 technologies on the client-side wiki. The integration 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, i.e., a formal description of wiki domain concepts and their relationships. 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.

When semantic capabilities are enabled in a wiki system, our architecture can also generate semantic metadata from the results of semantic NLP pipelines and transform them in a way that can be stored in a wiki system. Our work is the first to attempt to automatically generate semantic metadata from wiki content, thus making it machine-accessible. Eventually, the semantic metadata that has been added to the wiki can be exploited to

organize its content, enhance its search features or exported for external application use.

Finally, the integration was applied to MediaWiki – a widely-used wiki engine best known from the Wikimedia projects – to prove its feasibility. We developed a MediaWiki ontology, as well as helper modules, to integrate it in our Wiki-NLP architecture. The integration was ultimately introduced to the wiki engine via a light-weight plug-in.

The NLP-enhanced MediaWiki instance was used in the evaluations process during this work. First, we demonstrated the practicability of our work by applying NLP services on an existing heritage data wiki. In this context, we automatically generated a back-of-the-book index through an NLP service call – a feature that had previously been implemented in the wiki itself. This way, we showed that the same capabilities can be provided to the wiki through our general wiki- and service-independent architecture.

Second, as the first to perform an extrinsic evaluation of using NLP techniques in wikis, we demonstrated the usability, effectiveness and efficiency of our integration within a number of real-world projects. As part of this, we used our NLP-enhanced MediaWiki instance in two software engineering courses as a collaborative platform, where students could use it to develop their assignments and ultimately use the provided NLP services to detect defects in their documents. Our survey results, gathered from the students, showed the usability of our integration, despite their low knowledge level in the NLP domain. Also, by examining the number of defects found in their assignments before and after using NLP services, we concluded that the NLP integration can indeed help to improve document quality.

Third, to evaluate the efficiency of our approach, we integrated our architecture into the workflow of a functional genomics project, where biologists manually extract knowledge from the biomedical literature and

Table 10: Comparison of Wiki-NLP integration against architecture requirements and similar wikis

Requirement	Wikulu	SMW	IkeWiki	SweetWiki	AceWiki	Wiki-NLP
Seamless Integration	✓	~	✗	✗	✗	✓
NLP Service Recommendation	✓	✗	✗	✗	✗	✓
Change Visibility	✗	~	✗	✓	✓	✓
Low Learning Curve	✓	✗	✗	✓	✓	✓
Easy Deployment	✓	✓	✓	✓	✓	✓
Facilitate Client Integration	✗	✓	✗	✗	✗	✓
Wiki System Independence	✓	✗	✗	✗	✗	✓
Flexible Response Handling	✓	✗	✗	✗	✗	✓
Read Content from Wiki	✓	✓	✓	✓	✓	✓
Write Content to Wiki	✓	✓	✓	✓	✓	✓
External Data Access	~	✓	✗	✓	✗	✓
Proactive Service Execution	✗	✓	✗	✗	✗	✓

✓ = fully satisfied

~ = partially satisfied or not available in literature

✗ = not satisfied

curate them for a database. Using our integration, knowledge was automatically extracted by NLP pipelines, providing additional semantic metadata for each extracted entity. Employing NLP analysis on the wiki content not only proved to reduce the curation time up to almost 20%, but also enriched the wiki with semantic metadata, thus further facilitating knowledge management for end-users. Table 10 presents our Wiki-NLP integration in comparison with the wikis described in Chapter 4. The table shows that our Wiki-NLP integration is able to fulfill all the requirements for employing NLP techniques in various wikis.

## 8.2 Suggestions for Future Work

A number of our system aspects remain open for further investigation:

First, one of the powerful features of our architecture is its modularity, meaning that building blocks can be gradually added as they are needed. As new wiki engines and human-computer interaction technologies emerge, the current architecture components can be replaced by more up-to-date



technologies, as well as adding new components responsible for the newer functionalities.

The Wiki-NLP integration architecture developed in this work, although designed to be general, has only been evaluated on one wiki engine, namely MediaWiki. Therefore, it is suggested that the integration is tested on more wiki engines, as their different requirements will improve the compatibility of the existing solution.

Regarding the integration user interface, since our solution is mostly browser-based, more tests on different browsers need to be performed to assess the compatibility of the system interface. Major parts of our integration use client-side scripting languages, such as JavaScript, which are infamous for browser incompatibility issues. Also, as new technologies emerge and become adopted, these implementations will need to be replaced to provide a more interactive, intuitive and user-friendly user interface.

We developed a wiki ontology with a set of primitive concepts and relationships. The wiki ontology can also be extended to allow the integration to perform semantic reasoning on the wiki engine description, thus providing users with more convenience, e.g., reasoning on where and how to present the results, based on the wiki capabilities described in its ontology – and better service results, e.g., reasoning on where to look for input data in the wiki.

### **8.3 Conclusion**

Augmenting wikis with NLP capabilities has not attracted a lot of research attention yet and this research work is among the first. Unlike other existing related work, our thesis provides a general architecture to offer NLP capabilities to wikis, without the need for hard-coding such features or modifying the wiki engine. Our initial evaluations of the system proved the usefulness of the Wiki-NLP integration architecture in real-world projects carried out within a wiki environment. By providing a direct and seamless

integration of NLP capabilities, we are able help wiki users to overcome common problems of using wikis, such as, information overload or poor organization. This way, the organized structures 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.

Wikis were invented to change the role of end-users from being consumers of websites to “prosumers”<sup>1</sup> – they can read, develop or modify content of wikis via a simple browser interface. In this work, we introduced yet another party to the wiki community users: natural language processing services, performing text mining techniques on wiki content to develop primary and complementary content, organize the wiki structure, as well as enriching it with semantic metadata, so that it becomes accessible to machines.

The Wiki-NLP integration allows wiki users to benefit from the collaboration between the artificial intelligence domain and wiki systems by using various generic or domain-specific semantic assistants, seamlessly helping them with their tasks in a wiki, e.g., by creating focused summaries, extracting entities or answering questions based on a wiki’s available knowledge.

Although this thesis focused on the integration of NLP capabilities in wiki systems, the core idea can also be applied to other web information systems, such as Content Management Systems, due to the similar nature of web-based applications.

Finally, the integration of wikis and NLP systems also provides NLP pipeline developers with a chance to reach a new target audience and a body of content that grows over time. They can now directly access wiki content to develop and train more NLP pipelines to improve wiki users’ experience, thus, persuade them to use wikis to develop more content.

---

<sup>1</sup>This term is the portmanteau of “producer” and “consumer”, coined by Alvin Toffler in *The Third Wave*, ISBN 0517327198.

# Bibliography

- [AMC03] Deepak Alur, Dan Malks, and John Crupi. *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall, 2nd edition, 2003.
- [BEK<sup>+</sup>00] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple Object Access Protocol (SOAP) 1.1. Technical report, World Wide Web Consortium, May 2000.
- [BGE<sup>+</sup>08] Michel Buffa, Fabien Gandon, Guillaume Ereteo, Peter Sander, and Catherine Faron. SweetWiki: A Semantic Wiki. *Web Semantics*, 6(1):84–97, 2008.
- [Bro08] John Broughton. *Wikipedia: the missing manual*. O'Reilly, 1st edition, 2008.
- [Buf06] Michel Buffa. Intranet Wikis. In *Proceedings of IntraWeb Workshop 2006 at the 15th International World Wide Web Conference*, Edinburgh, Scotland, 23-26 May 2006.
- [CCMW01] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, World Wide Web Consortium, 2001.
- [CEJ<sup>+</sup>05] Vivek Chopra, Jon Eaves, Rupert Jones, Sing Li, and John T. Bell. *Beginning JavaServer Pages*. Wrox Press Ltd., Birmingham, UK, 2005.

- [CMB<sup>+</sup>11] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. *Text Processing with GATE (Version 6)*. University of Sheffield, Department of Computer Science, 2011.
- [CSG<sup>+</sup>09] Antje Chang, Maurice Scheer, Andreas Grote, Ida Schomburg, and Dietmar Schomburg. BRENDA, AMENDA and FRENDA the enzyme information system: new content and tools in 2009. *Nucleic Acids Research*, 37(Database-Issue):588–592, 2009.
- [Dey01] Anind K. Dey. Understanding and Using Context. *Personal Ubiquitous Computing*, 5:4–7, January 2001.
- [DRR<sup>+</sup>07] Björn Decker, Eric Ras, Jörg Rech, Pascal Jaubert, and Marco Rieth. Wiki-Based Stakeholder Participation in Requirements Engineering. *IEEE Software*, 24(2):28–35, March/April 2007.
- [DuB06] William H. DuBay. *Smart language: Readers, Readability, and the Grading of Text*. Impact Information, 2006.
- [EGHW08] Anja Ebersbach, Markus Glaser, Richard Heigl, and Alexander Warta. *Wiki: Web Collaboration*. Springer, 2nd edition, 2008.
- [FSS98] Norbert E. Fuchs, Uta Schwertel, and Rolf Schwitter. Attempto Controlled English - Not Just Another Logic Specification Language. In Pierre Flener, editor, *Logic-Based Program Synthesis and Transformation*, volume 1559, pages 1–20. Springer, 1998.
- [Gar05] Jesse James Garrett. Ajax: A New Approach to Web Applications. <http://adaptivepath.com/ideas/essays/archives/000385.php>, February 2005.
- [GHJV95] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

- [GW08] Thomas Gitzinger and René Witte. Enhancing the OpenOffice.org Word Processor with Natural Language Processing Capabilities. In *Natural Language Processing resources, algorithms and tools for authoring aids*, Marrakech, Morocco, 1 June 2008.
- [Hea97] Marti A. Hearst. TextTiling: Segmenting Text into Multiparagraph Subtopic Passages. *Computational Linguistics*, 23(1):33–64, 1997.
- [HZG09] Johannes Hoffart, Torsten Zesch, and Iryna Gurevych. An architecture to support intelligent user interfaces for Wikis by means of Natural Language Processing. In Dirk Riehle and Amy Bruckman, editors, *International Symposium on Wikis (WikiSym2009)*, Orlando, Florida, USA, 25–27 October 2009. ACM.
- [KBD<sup>+</sup>09] Hak Lae Kim, John G. Breslin, Stefan Decker, Jaehwa Choi, and Hong-Gee Kim. Personal knowledge management for knowledge workers using social semantic technologies. *International Journal of Intelligent Information and Database Systems*, 3(1):28–43, 2009.
- [KC04] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210, February 2004.
- [KSB<sup>+</sup>10] Thomas Kurz, Sebastian Schaffert, Tobias Buerger, Stephanie Stroka, Rolf Sint, Mihai Radulescu, and Szabolcs Grunwald. KiWi – A Platform for building Semantic Social Media Applications. In *9th International Semantic Web Conference (ISWC2010)*, Shanghai, China, 7–11 November 2010.
- [Kuh08] Tobias Kuhn. AceWiki: A Natural and Expressive Semantic Wiki. In *Proceedings of Workshop Semantic Web User Interaction*

at CHI 2008: *Exploring HCI Challenges*, Florence, Italy, 5–10 April 2008.

- [KVV06] Markus Krötzsch, Denny Vrandečić, and Max Völkel. Semantic MediaWiki. In *The Semantic Web*, volume 4273 of *Lecture Notes in Computer Science*. Springer, 2006.
- [Lap09] Phillip A. Laplante. *Requirements Engineering for Software and Systems*. Auerbach Publications, Boston, MA, USA, 1st edition, 2009.
- [Lar04] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall PTR, 3rd edition, 2004.
- [LC01] Bo Leuf and Ward Cunningham. *The Wiki Way. Quick Collaboration on the Web*. Addison-Wesley Longman, 2001.
- [Lih04] Andrew Lih. Wikipedia as Participatory journalism: reliable sources? metrics for evaluating collaborative media as a news resource. In *Proceedings of the 5th International Symposium on Online Journalism*, pages 16–17, Texas, USA, 16–17 April 2004.
- [Mad08] Stewart Mader. *Wikipatterns – A practical guide to improving productivity and collaboration in your organization*. Wiley Publication, 2008.
- [MMM<sup>+</sup>11] Marie-Jean Meurs, Caitlin Murphy, Ingo Morgenstern, Nona Naderi, Greg Butler, Justin Powlowski, Adrian Tsang, and René Witte. Semantic Text Mining for Lignocellulose Research. In *The ACM Fifth International Workshop on Data and Text Mining in Biomedical Informatics in conjunction with CIKM*, Glasgow, UK, 24–28 October 2011.
- [MPW<sup>+</sup>11] Caitlin Murphy, Justin Powlowski, Min Wu, Greg Butler, and Adrian Tsang. Curation of characterized glycoside hydrolases of Fungal origin. *Database*, Volume2011, 2011.

- [MT04] R. Mihalcea and P. Tarau. TextRank: Bringing Order into Texts. In *Proceedings of EMNLP-04 and the 2004 Conference on Empirical Methods in Natural Language Processing*, 2004.
- [Mud10] Raphael Mudge. The Design of a Proofreading Software Service. In *Workshop on Computational Linguistics and Writing: Writing Processes and Authoring Aids, CL&W*, 2010.
- [NKBW11] Nona Naderi, Thomas Kappler, Christopher J.O. Baker, and René Witte. OrganismTagger: Detection, normalization, and grounding of organism entities in biomedical documents. *Bioinformatics*, 2011.
- [NL93] Jakob Nielsen and Thomas K Landauer. *A mathematical model of the finding of usability problems*, volume 206, pages 206–213. ACM, 1993.
- [OBD06] Eyal Oren, John G. Breslin, and Stefan Decker. How semantics make better wikis. In *Proceedings of the 15th international conference on the World Wide Web (WWW2006)*, Edinburgh, Scotland, 23–26 May 2006. ACM.
- [QL08] Bastian Quilitz and Ulf Leser. Querying distributed RDF data sources with SPARQL. In *Proceedings of the 5th European Conference on The Semantic Web: research and applications, ESWC'08*, pages 524–538. Springer-Verlag, 2008.
- [Sah07] Goutam Kumar Saha. Web ontology language (OWL) and semantic web. *Ubiquity*, 2007, September 2007.
- [Sch06] Sebastian Schaffert. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In *IEEE International Conference on Collaboration Technologies and Infrastructures*, pages 388–396. IEEE Computer Society, 2006.



- [SFAV05] Clara Silveira, João Pascoal Faria, Ademar Aguiar, and Raul Vidal. Wiki-Based Requirements Documentation of Generic Software Products. In *Proceedings of the 10th Australian Workshop on Requirements Engineering*, 2005.
- [Sim71] Herbert A. Simon. Designing organizations for an information rich world. In Martin Greenberger, editor, *Computers, communications, and the public interest*, pages 37–72. The Johns Hopkins Press, 1971.
- [UG96] M. Uschold and M. Gruninger. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.
- [W3C07] W3C. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). W3c recommendation, World Wide Web Consortium (W3C), 2007.
- [Wat07] Neil L. Waters. Why you can't cite Wikipedia in my class. *Commun. ACM*, 50:15–17, September 2007.
- [WG07] René Witte and Thomas 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.
- [WG09] René Witte and Thomas 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, Pathumthani, Thailand, 2–5 February 2009. Springer.
- [WGKK08] René Witte, Thomas Gitzinger, Thomas Kappler, and Ralf Krestel. A Semantic Wiki Approach to Cultural Heritage Data Management. In *Language Technology for Cultural Heritage Data (LaTeCH 2008)*, June 1st 2008.



- [WKKL10] René Witte, Ralf Krestel, Thomas Kappler, and Peter C. Lockemann. Converting a Historical Architecture Encyclopedia into a Semantic Knowledge Base. *IEEE Intelligent Systems*, 25(1):58–66, January/February 2010.
- [WKKL11] René Witte, Thomas Kappler, Ralf Krestel, and Peter C. Lockemann. *Integrating Wiki Systems, Natural Language Processing, and Semantic Technologies for Cultural Heritage Data Management*, pages 213–230. *Theory and Applications of Natural Language Processing*. Springer, 2011.
- [WPF<sup>+</sup>04] Ian Witten, Gordon Paynter, Eibe Frank, Carl Gutwin, and Craig Nevill-Manning. Kea: Practical automatic keyphrase extraction. pages 314–326, 2004.

# Appendix A

## Wiki Upper Ontology Description

```
1 | @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
2 | @prefix swrlb: <http://www.w3.org/2003/11/swrlb#> .
3 | @prefix owl: <http://www.w3.org/2002/07/owl#> .
4 | @prefix protege: <http://protege.stanford.edu/plugins/owl/protege#> .
5 | @prefix cu: <http://localhost/ConceptUpper.owl#> .
6 | @prefix xsp: <http://www.owl-ontologies.com/2005/08/07/xsp.owl#> .
7 | @prefix : <http://localhost/WikiOntology.owl#> .
8 | @prefix xml: <http://www.w3.org/XML/1998/namespace> .
9 | @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
10 | @prefix swrl: <http://www.w3.org/2003/11/swrl#> .
11 | @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
12 | @base <http://localhost/WikiOntology.owl> .
13 |
14 | <http://localhost/WikiOntology.owl> rdf:type owl:Ontology ;
15 |     owl:imports <http://localhost/ConceptUpper.owl> .
16 |
17 | #####
18 | #   Object Properties
19 | #####
20 |
21 | ### http://localhost/WikiOntology.owl#belongsToNS
22 | :belongsToNS rdf:type owl:ObjectProperty ;
23 |     rdfs:range :Namespace ;
24 |     rdfs:domain :Page .
25 |
26 | ### http://localhost/WikiOntology.owl#hasContent
27 | :hasContent rdf:type owl:ObjectProperty ;
28 |     rdfs:domain :Page ;
29 |     rdfs:range [ rdf:type owl:Class ;
30 |                 owl:unionOf ( cu:Format
31 |                                 :WikiMarkup
32 |                             )
33 |                 ] .
34 |
```

```

35 ### http://localhost/WikiOntology.owl#hasMetadata
36 :hasMetadata rdf:type owl:ObjectProperty ;
37           rdfs:range :Metadata .
38
39 ### http://localhost/WikiOntology.owl#hasTalkpage
40 :hasTalkpage rdf:type owl:ObjectProperty ;
41           rdfs:domain :Content_Page ;
42           rdfs:range :Talk_Page .
43
44 #####
45 #   Data properties
46 #####
47
48 ### http://localhost/WikiOntology.owl#hasEngine
49 :hasEngine rdf:type owl:DatatypeProperty ;
50           rdfs:domain :Wiki ;
51           rdfs:range xsd:string .
52
53 ### http://localhost/WikiOntology.owl#hasVersion
54 :hasVersion rdf:type owl:DatatypeProperty ;
55           rdfs:domain :Wiki .
56
57 #####
58 #   Classes
59 #####
60
61 ### http://localhost/ConceptUpper.owl#Artifact
62 cu:Artifact rdf:type owl:Class .
63
64 ### http://localhost/ConceptUpper.owl#ArtificialLanguage
65 cu:ArtificialLanguage rdf:type owl:Class .
66
67 ### http://localhost/ConceptUpper.owl#Format
68 cu:Format rdf:type owl:Class .
69
70 ### http://localhost/ConceptUpper.owl#Tool
71 cu:Tool rdf:type owl:Class .
72
73 ### http://localhost/WikiOntology.owl#Content_Page
74 :Content_Page rdf:type owl:Class ;
75           rdfs:subClassOf :Page .
76
77 ### http://localhost/WikiOntology.owl#History
78 :History rdf:type owl:Class ;
79           rdfs:subClassOf :Metadata .
80
81 ### http://localhost/WikiOntology.owl#Metadata
82 :Metadata rdf:type owl:Class .
83
84 ### http://localhost/WikiOntology.owl#Namespace
85 :Namespace rdf:type owl:Class .

```

```
86
87 ### http://localhost/WikiOntology.owl#Page
88 :Page rdf:type owl:Class ;
89       rdfs:subClassOf cu:Artifact .
90
91 ### http://localhost/WikiOntology.owl#Talk_Page
92 :Talk_Page rdf:type owl:Class ;
93           rdfs:subClassOf :Page .
94
95 ### http://localhost/WikiOntology.owl#Virtual_Namespace
96 :Virtual_Namespace rdf:type owl:Class ;
97                   rdfs:subClassOf :Namespace .
98
99 ### http://localhost/WikiOntology.owl#Wiki
100 :Wiki rdf:type owl:Class ;
101       rdfs:subClassOf cu:Tool .
102
103 ### http://localhost/WikiOntology.owl#WikiMarkup
104 :WikiMarkup rdf:type owl:Class ;
105           rdfs:subClassOf cu:ArtificialLanguage .
106
107 ### Generated by the OWL API (version 3.2.3.1824) http://owlapi.sourceforge.net
```

# Appendix B

## MediaWiki Ontology Description

```
1 | @prefix wo: <http://localhost/WikiOntology.owl#> .
2 | @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
3 | @prefix swrlb: <http://www.w3.org/2003/11/swrlb#> .
4 | @prefix owl: <http://www.w3.org/2002/07/owl#> .
5 | @prefix protege: <http://protege.stanford.edu/plugins/owl/protege#> .
6 | @prefix xsp: <http://www.owl-ontologies.com/2005/08/07/xsp.owl#> .
7 | @prefix : <http://localhost/MediaWiki.owl#> .
8 | @prefix xml: <http://www.w3.org/XML/1998/namespace> .
9 | @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
10 | @prefix swrl: <http://www.w3.org/2003/11/swrl#> .
11 | @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
12 | @base <http://localhost/MediaWiki.owl> .
13 |
14 | <http://localhost/MediaWiki.owl> rdf:type owl:Ontology ;
15 |     owl:imports <http://localhost/WikiOntology.owl> .
16 |
17 | #####
18 | #   Object Properties
19 | #####
20 |
21 | ### http://localhost/WikiOntology.owl#hasMetadata
22 | wo:hasMetadata rdfs:domain wo:Page .
23 |
24 | #####
25 | #   Data properties
26 | #####
27 |
28 | ### http://localhost/MediaWiki.owl#NS.Value
29 | :NS.Value rdf:type owl:DatatypeProperty ;
30 |     rdfs:domain wo:Namespace ;
31 |     rdfs:range xsd:string .
32 |
33 | #####
34 | #   Classes
```

```

35 #####
36
37 ### http://localhost/MediaWiki.owl#P_Article
38 :P_Article rdf:type owl:Class ;
39     rdfs:subClassOf wo:Content_Page ,
40         [ rdf:type owl:Restriction ;
41           owl:onProperty wo:belongsToNS ;
42           owl:hasValue :NS_Main
43         ] ,
44         [ rdf:type owl:Restriction ;
45           owl:onProperty wo:hasTalkpage ;
46           owl:someValuesFrom :P_Article_Talk
47         ] .
48
49 ### http://localhost/MediaWiki.owl#P_Article_Talk
50 :P_Article_Talk rdf:type owl:Class ;
51     rdfs:subClassOf wo:Talk_Page ,
52         [ rdf:type owl:Restriction ;
53           owl:onProperty wo:belongsToNS ;
54           owl:hasValue :NS_Talk
55         ] .
56
57 ### http://localhost/MediaWiki.owl#P_Category
58 :P_Category rdf:type owl:Class ;
59     rdfs:subClassOf wo:Content_Page ,
60         [ rdf:type owl:Restriction ;
61           owl:onProperty wo:hasTalkpage ;
62           owl:someValuesFrom :P_Category_Talk
63         ] ,
64         [ rdf:type owl:Restriction ;
65           owl:onProperty wo:belongsToNS ;
66           owl:hasValue :NS_Category
67         ] .
68
69 ### http://localhost/MediaWiki.owl#P_Category_Talk
70 :P_Category_Talk rdf:type owl:Class ;
71     rdfs:subClassOf wo:Talk_Page ,
72         [ rdf:type owl:Restriction ;
73           owl:onProperty wo:belongsToNS ;
74           owl:hasValue :NS_Category_Talk
75         ] .
76
77 ### http://localhost/MediaWiki.owl#P_File
78 :P_File rdf:type owl:Class ;
79     rdfs:subClassOf wo:Content_Page ,
80         [ rdf:type owl:Restriction ;
81           owl:onProperty wo:hasTalkpage ;
82           owl:someValuesFrom :P_File_Talk
83         ] ,
84         [ rdf:type owl:Restriction ;
85           owl:onProperty wo:belongsToNS ;

```

```

86         owl:hasValue :NS_File
87     ] .
88
89 ### http://localhost/MediaWiki.owl#P_File_Talk
90 :P_File_Talk rdf:type owl:Class ;
91     rdfs:subClassOf wo:Talk_Page ,
92         [ rdf:type owl:Restriction ;
93           owl:onProperty wo:belongsToNS ;
94           owl:hasValue :NS_File_Talk
95         ] .
96
97 ### http://localhost/MediaWiki.owl#P_Help
98 :P_Help rdf:type owl:Class ;
99     rdfs:subClassOf wo:Content_Page ,
100         [ rdf:type owl:Restriction ;
101           owl:onProperty wo:hasTalkpage ;
102           owl:someValuesFrom :P_Help_Talk
103         ] ,
104         [ rdf:type owl:Restriction ;
105           owl:onProperty wo:belongsToNS ;
106           owl:hasValue :NS_Help
107         ] .
108
109 ### http://localhost/MediaWiki.owl#P_Help_Talk
110 :P_Help_Talk rdf:type owl:Class ;
111     rdfs:subClassOf wo:Talk_Page ,
112         [ rdf:type owl:Restriction ;
113           owl:onProperty wo:belongsToNS ;
114           owl:hasValue :NS_Help
115         ] .
116
117 ### http://localhost/MediaWiki.owl#P_Project
118 :P_Project rdf:type owl:Class ;
119     rdfs:subClassOf wo:Content_Page ,
120         [ rdf:type owl:Restriction ;
121           owl:onProperty wo:hasTalkpage ;
122           owl:someValuesFrom :P_Project_Talk
123         ] ,
124         [ rdf:type owl:Restriction ;
125           owl:onProperty wo:belongsToNS ;
126           owl:hasValue :NS_Project
127         ] .
128
129 ### http://localhost/MediaWiki.owl#P_Project_Talk
130 :P_Project_Talk rdf:type owl:Class ;
131     rdfs:subClassOf wo:Talk_Page ,
132         [ rdf:type owl:Restriction ;
133           owl:onProperty wo:belongsToNS ;
134           owl:hasValue :NS_Project_Talk
135         ] .
136

```

```

137 ### http://localhost/MediaWiki.owl#P_Template
138 :P_Template rdf:type owl:Class ;
139         rdfs:subClassOf wo:Content_Page ,
140                 [ rdf:type owl:Restriction ;
141                   owl:onProperty wo:hasTalkpage ;
142                   owl:someValuesFrom :P_Template_Talk
143                 ] ,
144                 [ rdf:type owl:Restriction ;
145                   owl:onProperty wo:belongsToNS ;
146                   owl:hasValue :NS_Template
147                 ] .
148
149 ### http://localhost/MediaWiki.owl#P_Template_Talk
150 :P_Template_Talk rdf:type owl:Class ;
151         rdfs:subClassOf wo:Talk_Page ,
152                 [ rdf:type owl:Restriction ;
153                   owl:onProperty wo:belongsToNS ;
154                   owl:hasValue :NS_Template_Talk
155                 ] .
156
157 ### http://localhost/MediaWiki.owl#P_User_Page
158 :P_User_Page rdf:type owl:Class ;
159         rdfs:subClassOf wo:Content_Page ,
160                 [ rdf:type owl:Restriction ;
161                   owl:onProperty wo:belongsToNS ;
162                   owl:hasValue :NS_User
163                 ] ,
164                 [ rdf:type owl:Restriction ;
165                   owl:onProperty wo:hasTalkpage ;
166                   owl:someValuesFrom :P_User_Talk
167                 ] .
168
169 ### http://localhost/MediaWiki.owl#P_User_Talk
170 :P_User_Talk rdf:type owl:Class ;
171         rdfs:subClassOf wo:Talk_Page ,
172                 [ rdf:type owl:Restriction ;
173                   owl:onProperty wo:belongsToNS ;
174                   owl:hasValue :NS_User_Talk
175                 ] .
176
177 ### http://localhost/WikiOntology.owl#Page
178 wo:Page rdfs:subClassOf [ rdf:type owl:Restriction ;
179                           owl:onProperty wo:hasMetadata ;
180                           owl:someValuesFrom wo:History
181                         ] .
182
183 #####
184 #   Individuals
185 #####
186
187 ### http://localhost/MediaWiki.owl#Media

```



```

188 :Media rdf:type wo:VirtualNamespace ,
189         owl:NamedIndividual ;
190
191     :NS_Value "Media"^^xsd:string .
192
193 ### http://localhost/MediaWiki.owl#MediaWiki
194 :MediaWiki rdf:type wo:Wiki ,
195             owl:NamedIndividual ;
196             wo:hasVersion "1.16"^^xsd:string ;
197             wo:hasEngine "MediaWiki"@en .
198
199 ### http://localhost/MediaWiki.owl#MediaWiki_Markup
200 :MediaWiki_Markup rdf:type wo:WikiMarkup ,
201                       owl:NamedIndividual .
202
203 ### http://localhost/MediaWiki.owl#NS_Category
204 :NS_Category rdf:type wo:Namespace ,
205              owl:NamedIndividual ;
206              :NS_Value "Category"^^xsd:string .
207
208 ### http://localhost/MediaWiki.owl#NS_Category_Talk
209 :NS_Category_Talk rdf:type wo:Namespace ,
210                   owl:NamedIndividual ;
211                   :NS_Value "Category_Talk"^^xsd:string .
212
213 ### http://localhost/MediaWiki.owl#NS_File
214 :NS_File rdf:type wo:Namespace ,
215          owl:NamedIndividual ;
216          :NS_Value "File"^^xsd:string .
217
218 ### http://localhost/MediaWiki.owl#NS_File_Talk
219 :NS_File_Talk rdf:type wo:Namespace ,
220              owl:NamedIndividual ;
221              :NS_Value "File_Talk"^^xsd:string .
222
223 ### http://localhost/MediaWiki.owl#NS_Help
224 :NS_Help rdf:type wo:Namespace ,
225         owl:NamedIndividual ;
226         :NS_Value "Help"^^xsd:string .
227
228 ### http://localhost/MediaWiki.owl#NS_Help_Talk
229 :NS_Help_Talk rdf:type wo:Namespace ,
230              owl:NamedIndividual ;
231              :NS_Value "Help_Talk"^^xsd:string .
232
233 ### http://localhost/MediaWiki.owl#NS_Main
234 :NS_Main rdf:type wo:Namespace ,
235         owl:NamedIndividual ;
236         :NS_Value "Main"^^xsd:string .
237
238 ### http://localhost/MediaWiki.owl#NS_MediaWiki

```

```

239
240 :NS_MediaWiki rdf:type wo:Namespace ,
241             owl:NamedIndividual ;
242             :NS_Value "MediaWiki"^^xsd:string .
243
244 ### http://localhost/MediaWiki.owl#NS_MediaWiki_Talk
245 :NS_MediaWiki_Talk rdf:type wo:Namespace ,
246                  owl:NamedIndividual ;
247                  :NS_Value "MediaWiki_Talk"^^xsd:string .
248
249 ### http://localhost/MediaWiki.owl#NS_Project
250 :NS_Project rdf:type wo:Namespace ,
251            owl:NamedIndividual ;
252            :NS_Value "Project"^^xsd:string .
253
254 ### http://localhost/MediaWiki.owl#NS_Project_Talk
255 :NS_Project_Talk rdf:type wo:Namespace ,
256                 owl:NamedIndividual ;
257                 :NS_Value "Project_Talk"^^xsd:string .
258
259 ### http://localhost/MediaWiki.owl#NS_Talk
260 :NS_Talk rdf:type wo:Namespace ,
261         owl:NamedIndividual ;
262         :NS_Value "Talk"^^xsd:string .
263
264 ### http://localhost/MediaWiki.owl#NS_Template
265 :NS_Template rdf:type wo:Namespace ,
266             owl:NamedIndividual ;
267             :NS_Value "Template"^^xsd:string .
268
269 ### http://localhost/MediaWiki.owl#NS_Template_Talk
270 :NS_Template_Talk rdf:type wo:Namespace ,
271                  owl:NamedIndividual ;
272                  :NS_Value "Template_Talk"^^xsd:string .
273
274 ### http://localhost/MediaWiki.owl#NS_User
275 :NS_User rdf:type wo:Namespace ,
276         owl:NamedIndividual ;
277         :NS_Value "User"^^xsd:string .
278
279 ### http://localhost/MediaWiki.owl#NS_User_Talk
280 :NS_User_Talk rdf:type wo:Namespace ,
281              owl:NamedIndividual ;
282              :NS_Value "User_Talk"^^xsd:string .
283
284 ### http://localhost/MediaWiki.owl#Special
285 :Special rdf:type wo:VirtualNamespace ,
286          owl:NamedIndividual ;
287          :NS_Value "Special"^^xsd:string .
288
289 ### Generated by the OWL API (version 3.2.3.1824) http://owlapi.sourceforge.net

```

# Appendix C

## ReqWiki Questionnaire

The questionnaire provided to ReqWiki users in the evaluation scenario described is Section 7.3 in presented in this section. The following questionnaire was designed using LimeSurvey<sup>1</sup>, an open-source survey generator written in PHP. The logic embedded in the questionnaire uses conditions to show or hide questions. Therefore, ReqWiki users were only asked wiki related questions when answered “ReqWiki” to question 6.

It should be noted that this questionnaire also contains the Semantic Assistants OpenOffice.org Writer plug-in [GW08], as well as NLP service specific questions, in addition to the ReqWiki related questions. However, the demographic and ReqWiki-related results were isolated and only considered during the analysis. The raw data gathered from the questionnaire is presented in Appendix D.

---

<sup>1</sup><http://www.limesurvey.org/>

This is a questionnaire designed to gather feedback from the students of Software Requirements Specifications course, instructed by Dr. René Witte in Fall 2011. The goal is to improve tools, in the form of ReqWiki and Semantic Assistants, to support both students and professionals in software requirements engineering. Results from this research will be made freely available under open source licenses.

Please note that this is an anonymous survey, meaning that your personal information (e.g., your name or student number) is not stored along with your answers. Therefore, candid and honest feedback is encouraged. There is **no** connection with your course marks or grade whatsoever - this questionnaire is strictly for research and tool development purposes.

Thank you,

The Semantic Software Lab Research Team

There are 19 questions in this survey

## Demographic General Questions

### 1 [D1]How do you rate your level of proficiency in English language? \*

Please choose **only one** of the following:

- Limited Working Proficiency (can handle limited work requirements)
- Professional Working Proficiency (sufficient structural accuracy and vocabulary)
- Native or Bilingual (equivalent to that of an educated native speaker)

### 2 [D3]What is your level of experience in the area of Natural Language Processing? \*

Please choose **only one** of the following:

- Previous academic experience (e.g., you have taken related courses)
- Previous industrial experience (e.g., you have worked in this area)
- Both academic and industrial experience
- None

### 3 [D4]Rank your level of expertise in the following areas: \*

Please choose the appropriate response for each item:

	Novice	Advanced Beginner	Competent	Proficient	Expert
--	--------	----------------------	-----------	------------	--------

	Novice	Advanced Beginner	Competent	Proficient	Expert
Software Programming	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Software Testing	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Software Traceability	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
World Wide Web Concepts	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Semantic Web Concepts	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Definitions:

**Novice:** Minimal, or 'textbook' knowledge without connecting it to practice  
**Advanced Beginner:** Working knowledge of key aspects of practice  
**Competent:** Good working and background knowledge of area of practice  
**Proficient:** Depth of understanding of discipline and area of practice  
**Expert:** Authoritative knowledge of discipline and deep tacit understanding across area of practice

**4 [D5] Rank your level of expertise in the following areas BEFOR taking this course: \***

Please choose the appropriate response for each item:

	Novice	Advanced Beginner	Competent	Proficient	Expert
Software Development Methodologies	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Requirements Elicitation and Evaluation	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Requirements Specification (documentation)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Requirements Quality Assurance	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Domain Modeling (UML, RDF, Ontologies)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Definitions:

**Novice:** Minimal, or 'textbook' knowledge without connecting it to practice  
**Advanced Beginner:** Working knowledge of key aspects of practice  
**Competent:** Good working and background knowledge of area of practice  
**Proficient:** Depth of understanding of discipline and area of practice  
**Expert:** Authoritative knowledge of discipline and deep tacit understanding across area of practice

**5 [D6]**

**Rank your level of expertise in the following areas AFTER taking this course:**

\*

Please choose the appropriate response for each item:

	Novice	Advanced Beginner	Competent	Proficient	Expert
Software Development Methodologies	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Requirements Elicitation and Evaluation	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Requirements Specification (documentation)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Requirements Quality Assurance	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Domain Modeling (UML, RDF, Ontologies)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Definitions:

**Novice:** Minimal, or 'textbook' knowledge without connecting it to practice

**Advanced Beginner:** Working knowledge of key aspects of practice

**Competent:** Good working and background knowledge of area of practice

**Proficient:** Depth of understanding of discipline and area of practice

**Expert:** Authoritative knowledge of discipline and deep tacit understanding across area of practice

## Semantic Assistants

### 6 [SA1]Which one of the following applications did you use for the documentation of your assignment? \*

Please choose **only one** of the following:

OpenOffice.org Writer

ReqWiki

Other

### 7 [SA2]Which one of the following approaches did you use for the quality assessment of your SRS documents? \*

Please choose **only one** of the following:

The Semantic Assistants services

I did it manually

### 8 [SA3]Why did you choose to manually assess the quality of your SRS documents? \*

**Only answer this question if the following conditions are met:**

° Answer was SA12'I did it manually' at question '7 [SA2]' (Which one of the following approaches did you use for the quality assessment of your SRS documents?)

Please write your answer here:

### 9 [SA4]How easy-to-use did you find the Semantic Assistants user interface in the wiki? \*

**Only answer this question if the following conditions are met:**

° Answer was SA12'ReqWiki' at question '6 [SA1]' (Which one of the following applications did you use for the documentation of your assignment?) *and* Answer was SA11'The Semantic Assistants services' at question '7 [SA2]' (Which one of the following approaches did you use for the quality assessment of your SRS documents?)

Please choose **only one** of the following:

- Very Easy
- Easy
- Neutral
- Difficult
- Very Difficult

**10 [SA5]How easy-to-use did you find the Semantic Assistants OpenOffice plug-in user interface? \*****Only answer this question if the following conditions are met:**

° Answer was SA11'OpenOffice.org Writer' at question '6 [SA1]' (Which one of the following applications did you use for the documentation of your assignment?) *and* Answer was SA11'The Semantic Assistants services' at question '7 [SA2]' (Which one of the following approaches did you use for the quality assessment of your SRS documents?)

Please choose **only one** of the following:

- Very Easy
- Easy
- Neutral
- Difficult
- Very Difficult

**11 [SA6]Check all the difficulties you encountered while using the Semantic Assistants plug-in: \*****Only answer this question if the following conditions are met:**

° Answer was SA12'ReqWiki' at question '6 [SA1]' (Which one of the following applications did you use for the documentation of your assignment?) *and* Answer was SA11'The Semantic Assistants services' at question '7 [SA2]' (Which one of the following approaches did you use for the quality assessment of your SRS documents?)

Please choose **all** that apply :

- Browser Incompatibility
- Slow Response Time
- Results were hard to read/understand



None of above

Other:

**12 [SA7]Check all the difficulties you encountered while using the Semantic Assistants plug-in: \***

**Only answer this question if the following conditions are met:**

° Answer was SA11'OpenOffice.org Writer' at question '6 [SA1]' (Which one of the following applications did you use for the documentation of your assignment?) and Answer was SA11'The Semantic Assistants services' at question '7 [SA2]' (Which one of the following approaches did you use for the quality assessment of your SRS documents?)

Please choose **all** that apply:

- It was hard to install
- Slow Response Time
- Results were hard to read/understand
- None of above
- Other:

**13 [SA8]Rank which one of the ReqWiki features were the most beneficial to you: \***

**Only answer this question if the following conditions are met:**

° Answer was SA12'ReqWiki' at question '6 [SA1]' (Which one of the following applications did you use for the documentation of your assignment?) and Answer was SA11'The Semantic Assistants services' at question '7 [SA2]' (Which one of the following approaches did you use for the quality assessment of your SRS documents?)

Please choose the appropriate response for each item:

	Most beneficial	Beneficial	Neutral	Least beneficial
Collaborative environment	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Automatic versioning of pages	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Pre-defined Queries	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Pre-defined Traceability Links	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
SRS Forms (data entry)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
SRS Templates (data presentation)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

	Most beneficial	Beneficial	Neutral	Least beneficial
Ease of navigation between entities	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Ease of finding inconsistent/missing entities	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**14 [SA9] Given the experience of using ReqWiki, which application would you use for Software Requirements Engineering in future? \***

Only answer this question if the following conditions are met:

° Answer was SA12'ReqWiki' at question '6 [SA1]' (Which one of the following applications did you use for the documentation of your assignment?)

Please choose **only one** of the following:

- ReqWiki-like System with semantic support
- Traditional Wiki
- Traditional Word Processor (e.g., OpenOffice Writer, Microsoft Word)
- Other

**15 [SA10] Rank which one of the Semantic Assistants plug-in were the most beneficial to you: \***

Only answer this question if the following conditions are met:

° Answer was SA11'OpenOffice.org Writer' at question '6 [SA1]' (Which one of the following applications did you use for the documentation of your assignment?) and Answer was SA11'The Semantic Assistants services' at question '7 [SA2]' (Which one of the following approaches did you use for the quality assessment of your SRS documents?)

Please choose the appropriate response for each item:

	Most beneficial	Beneficial	Neutral	Least beneficial
Annotation Highlighting	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Interactive Annotations	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Customizing User Interface (font size)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Show/Hide Annotation content or features	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**16 [SA11] Given the experience of using the Natural Language Processing, would you use it again for Software Requirements Engineering tasks in future? \***

**Only answer this question if the following conditions are met:**

° Answer was SA11'OpenOffice.org Writer' at question '6 [SA1]' (Which one of the following applications did you use for the documentation of your assignment?) *and* Answer was SA11'The Semantic Assistants services' at question '7 [SA2]' (Which one of the following approaches did you use for the quality assessment of your SRS documents?)

Please choose **only one** of the following:

- Yes
- No (specify the reason in the box)

Make a comment on your choice here:

## NLP Services

### 17 [NLP1]Rate which services were the most beneficial to you: \*

Please choose the appropriate response for each item:

	Most Beneficial	Beneficial	Neutral	Least Beneficial	Did not use it
Information Extractor	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Writing Quality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
English Dum Indexer	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Readability Metrics (including the statistics)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Person and Location Extractor	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Requirements QA Defects	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Requirements QA Statistics	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

### 18 [NLP2]What other services do you wish to see in the context of Software Requirements Engineering?

Please write your answer here:

# **Appendix D**

## **Questionnaire Responses**

The data gathered from ReqWiki questionnaire feedbacks is provided in this section.

### **D.1 Graduate Students Responses**

## Results

**Number of records in this query:** 31  
**Total records in survey:** 31  
**Percentage of total:** 100.00%

### Field summary for D1

#### How do you rate your level of proficiency in English language?

Answer	Count	Percentage
Limited Working Proficiency (can handle limited work requirements) (D11)	9	29.03%
Professional Working Proficiency (sufficient structural accuracy and vocabulary) (D12)	15	48.39%
Native or Bilingual (equivalent to that of an educated native speaker) (D13)	7	22.58%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for D3

#### What is your level of experience in the area of Natural Language Processing?

Answer	Count	Percentage
Previous academic experience (e.g., you have taken related courses) (D31)	11	35.48%
Previous industrial experience (e.g., you have worked in this area) (D32)	4	12.90%
Both academic and industrial experience (D33)	2	6.45%
None (D34)	14	45.16%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for D4(D41)

#### Rank your level of expertise in the following areas:

##### [Software Programming]

Answer	Count	Percentage
Novice (D4L1)	1	3.23%
Advanced Beginner (D4L2)	6	19.35%
Competent (D4L3)	9	29.03%
Proficient (D4L4)	14	45.16%
Expert (D4L5)	1	3.23%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D4(D42)****Rank your level of expertise in the following areas:****[Software Testing]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D4L1)	4	12.90%
Advanced Beginner (D4L2)	5	16.13%
Competent (D4L3)	14	45.16%
Proficient (D4L4)	7	22.58%
Expert (D4L5)	1	3.23%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D4(D43)****Rank your level of expertise in the following areas:****[Software Traceability]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D4L1)	10	32.26%
Advanced Beginner (D4L2)	12	38.71%
Competent (D4L3)	5	16.13%
Proficient (D4L4)	4	12.90%
Expert (D4L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D4(D44)****Rank your level of expertise in the following areas:****[World Wide Web Concepts]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D4L1)	4	12.90%
Advanced Beginner (D4L2)	7	22.58%
Competent (D4L3)	16	51.61%
Proficient (D4L4)	3	9.68%
Expert (D4L5)	1	3.23%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D4(D45)****Rank your level of expertise in the following areas:****[Semantic Web Concepts]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D4L1)	18	58.06%
Advanced Beginner (D4L2)	6	19.35%

160

**Field summary for D4(D45)****Rank your level of expertise in the following areas:****[Semantic Web Concepts]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Competent (D4L3)	6	19.35%
Proficient (D4L4)	1	3.23%
Expert (D4L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D5(D51)****Rank your level of expertise in the following areas BEFOR taking this course:****[Software Development Methodologies]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D5L1)	2	6.45%
Advanced Beginner (D5L2)	14	45.16%
Competent (D5L3)	10	32.26%
Proficient (D5L4)	5	16.13%
Expert (D5L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D5(D52)****Rank your level of expertise in the following areas BEFOR taking this course:****[Requirements Elicitation and Evaluation]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D5L1)	7	22.58%
Advanced Beginner (D5L2)	14	45.16%
Competent (D5L3)	7	22.58%
Proficient (D5L4)	2	6.45%
Expert (D5L5)	1	3.23%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D5(D53)****Rank your level of expertise in the following areas BEFOR taking this course:****[Requirements Specification (documentation)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D5L1)	6	19.35%
Advanced Beginner (D5L2)	13	41.94%
Competent (D5L3)	9	29.03%
Proficient (D5L4)	2	6.45%



**Field summary for D5(D53)**

**Rank your level of expertise in the following areas BEFOR taking this course:  
[Requirements Specification (documentation)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Expert (D5L5)	1	3.23%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D5(D54)**

**Rank your level of expertise in the following areas BEFOR taking this course:  
[Requirements Quality Assurance]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D5L1)	9	29.03%
Advanced Beginner (D5L2)	14	45.16%
Competent (D5L3)	7	22.58%
Proficient (D5L4)	1	3.23%
Expert (D5L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D5(D55)**

**Rank your level of expertise in the following areas BEFOR taking this course:  
[Domain Modeling (UML, RDF, Ontologies)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D5L1)	5	16.13%
Advanced Beginner (D5L2)	13	41.94%
Competent (D5L3)	9	29.03%
Proficient (D5L4)	3	9.68%
Expert (D5L5)	1	3.23%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D6(D61)**

**Rank your level of expertise in the following areas AFTER taking this course:  
[Software Development Methodologies]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D6L1)	0	0.00%
Advanced Beginner (D6L2)	6	19.35%
Competent (D6L3)	13	41.94%
Proficient (D6L4)	11	35.48%
Expert (D6L5)	1	3.23%
No answer	0	0.00%

162

**Field summary for D6(D61)**

**Rank your level of expertise in the following areas AFTER taking this course:  
[Software Development Methodologies]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Not displayed	0	0.00%

**Field summary for D6(D62)**

**Rank your level of expertise in the following areas AFTER taking this course:  
[Requirements Elicitation and Evaluation]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D6L1)	0	0.00%
Advanced Beginner (D6L2)	5	16.13%
Competent (D6L3)	17	54.84%
Proficient (D6L4)	8	25.81%
Expert (D6L5)	1	3.23%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D6(D63)**

**Rank your level of expertise in the following areas AFTER taking this course:  
[Requirements Specification (documentation)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D6L1)	0	0.00%
Advanced Beginner (D6L2)	4	12.90%
Competent (D6L3)	19	61.29%
Proficient (D6L4)	7	22.58%
Expert (D6L5)	1	3.23%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D6(D64)**

**Rank your level of expertise in the following areas AFTER taking this course:  
[Requirements Quality Assurance]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D6L1)	0	0.00%
Advanced Beginner (D6L2)	6	19.35%
Competent (D6L3)	19	61.29%
Proficient (D6L4)	5	16.13%
Expert (D6L5)	1	3.23%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for D6(D65)

Rank your level of expertise in the following areas AFTER taking this course:  
[Domain Modeling (UML, RDF, Ontologies)]

Answer	Count	Percentage
Novice (D6L1)	0	0.00%
Advanced Beginner (D6L2)	6	19.35%
Competent (D6L3)	17	54.84%
Proficient (D6L4)	7	22.58%
Expert (D6L5)	1	3.23%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for SA1

Which one of the following applications did you use for the documentation of your assignment?

Answer	Count	Percentage
OpenOffice.org Writer (SA11)	18	58.06%
ReqWiki (SA12)	8	25.81%
Other	5	16.13%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for SA2

Which one of the following approaches did you use for the quality assessment of your SRS documents?

Answer	Count	Percentage
The Semantic Assistants services (SA11)	28	90.32%
I did it manually (SA12)	3	9.68%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for SA3

Why did you choose to manually assess the quality of your SRS documents?

Answer	Count	Percentage
Answer	3	9.68%
No answer	0	0.00%
Not displayed	28	90.32%

### Field summary for SA4

How easy-to-use did you find the Semantic Assistants user interface in the wiki?

Answer	Count	Percentage
Very Easy (SA41)	3	9.68%

#### Field summary for SA4

**How easy-to-use did you find the Semantic Assistants user interface in the wiki?**

Answer	Count	Percentage
Easy (SA42)	2	6.45%
Neutral (SA43)	3	9.68%
Difficult (SA44)	0	0.00%
Very Difficult (SA45)	0	0.00%
No answer	0	0.00%
Not displayed	23	74.19%

#### Field summary for SA5

**How easy-to-use did you find the Semantic Assistants OpenOffice plug-in user interface?**

Answer	Count	Percentage
Very Easy (SA51)	2	6.45%
Easy (SA52)	9	29.03%
Neutral (SA53)	4	12.90%
Difficult (SA54)	0	0.00%
Very Difficult (SA55)	0	0.00%
No answer	0	0.00%
Not displayed	16	51.61%

#### Field summary for SA6

**Check all the difficulties you encountered while using the Semantic Assistants plug-in:**

Answer	Count	Percentage
Browser Incompatibility (SA61)	1	3.23%
Slow Response Time (SA62)	1	3.23%
Results were hard to read/understand (SA63)	3	9.68%
None of above (SA64)	3	9.68%
Other	3	9.68%

#### Field summary for SA7

**Check all the difficulties you encountered while using the Semantic Assistants plug-in:**

Answer	Count	Percentage
It was hard to install (SA71)	1	3.23%
Slow Response Time (SA72)	3	9.68%
Results were hard to read/understand (SA73)	8	25.81%
None of above (SA74)	4	12.90%
Other	4	12.90%

**Field summary for SA8(SA81)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Collaborative environment]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	5	16.13%
Beneficial (SAL82)	3	9.68%
Neutral (SAL83)	0	0.00%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	23	74.19%

**Field summary for SA8(SA82)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Automatic versioning of pages]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	4	12.90%
Beneficial (SAL82)	2	6.45%
Neutral (SAL83)	2	6.45%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	23	74.19%

**Field summary for SA8(SA83)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Pre-defined Queries]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	4	12.90%
Beneficial (SAL82)	3	9.68%
Neutral (SAL83)	0	0.00%
Least beneficial (SAL84)	1	3.23%
No answer	0	0.00%
Not displayed	23	74.19%

**Field summary for SA8(SA84)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Pre-defined Traceability Links]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	7	22.58%
Beneficial (SAL82)	1	3.23%
Neutral (SAL83)	0	0.00%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
	166	

**Field summary for SA8(SA84)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Pre-defined Traceability Links]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Not displayed	23	74.19%

**Field summary for SA8(SA85)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[SRS Forms (data entry)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	3	9.68%
Beneficial (SAL82)	4	12.90%
Neutral (SAL83)	1	3.23%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	23	74.19%

**Field summary for SA8(SA86)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[SRS Templates (data presentation)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	3	9.68%
Beneficial (SAL82)	4	12.90%
Neutral (SAL83)	1	3.23%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	23	74.19%

**Field summary for SA8(SA87)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Ease of navigation between entities]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	5	16.13%
Beneficial (SAL82)	2	6.45%
Neutral (SAL83)	1	3.23%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	23	74.19%

### Field summary for SA8(SA88)

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Ease of finding inconsistent/missing entities]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	4	12.90%
Beneficial (SAL82)	3	9.68%
Neutral (SAL83)	1	3.23%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	23	74.19%

### Field summary for SA9

**Given the experience of using ReqWiki, which application would you use for Software Requirements Engineering in future?**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
ReqWiki-like System with semantic support (SA91)	8	25.81%
Traditional Wiki (SA92)	0	0.00%
Traditional Word Processor (e.g., OpenOffice Writer, Microsoft Word) (SA93)	0	0.00%
Other	0	0.00%
No answer	0	0.00%
Not displayed	23	74.19%

### Field summary for SA10(SA101)

**Rank which one of the Semantic Assistants plug-in were the most beneficial to you:  
[Annotation Highlighting]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (S10L1)	5	16.13%
Beneficial (S10L2)	6	19.35%
Neutral (S10L3)	3	9.68%
Least beneficial (S10L4)	1	3.23%
No answer	0	0.00%
Not displayed	16	51.61%

### Field summary for SA10(SA102)

**Rank which one of the Semantic Assistants plug-in were the most beneficial to you:  
[Interactive Annotations]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (S10L1)	2	6.45%
Beneficial (S10L2)	7	22.58%
Neutral (S10L3)	6	19.35%

168

**Field summary for SA10(SA102)**

**Rank which one of the Semantic Assistants plug-in were the most beneficial to you:  
[Interactive Annotations]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Least beneficial (S10L4)	0	0.00%
No answer	0	0.00%
Not displayed	16	51.61%

**Field summary for SA10(SA103)**

**Rank which one of the Semantic Assistants plug-in were the most beneficial to you:  
[Customizing User Interface (font size)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (S10L1)	2	6.45%
Beneficial (S10L2)	3	9.68%
Neutral (S10L3)	9	29.03%
Least beneficial (S10L4)	1	3.23%
No answer	0	0.00%
Not displayed	16	51.61%

**Field summary for SA10(SA104)**

**Rank which one of the Semantic Assistants plug-in were the most beneficial to you:  
[Show/Hide Annotation content or features]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (S10L1)	2	6.45%
Beneficial (S10L2)	4	12.90%
Neutral (S10L3)	8	25.81%
Least beneficial (S10L4)	1	3.23%
No answer	0	0.00%
Not displayed	16	51.61%

**Field summary for SA11**

**Given the experience of using the Natural Language Processing, would you use it again for  
Software Requirements Engineering tasks in future?**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Yes (SA111)	14	45.16%
No (specify the reason in the box) (SA112)	1	3.23%
Comments	0	0.00%
No answer	0	0.00%
Not displayed	16	51.61%



**Field summary for NLP1(NLPA1)****Rate which services were the most beneficial to you:****[Information Extractor]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most Beneficial (NLP11)	4	12.90%
Beneficial (NLP12)	7	22.58%
Neutral (NLP13)	7	22.58%
Least Beneficial (NLP15)	0	0.00%
Did not use it (NLP16)	13	41.94%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for NLP1(NLPA2)****Rate which services were the most beneficial to you:****[Writing Quality]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most Beneficial (NLP11)	9	29.03%
Beneficial (NLP12)	13	41.94%
Neutral (NLP13)	5	16.13%
Least Beneficial (NLP15)	2	6.45%
Did not use it (NLP16)	2	6.45%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for NLP1(NLPA3)****Rate which services were the most beneficial to you:****[English Durm Indexer]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most Beneficial (NLP11)	3	9.68%
Beneficial (NLP12)	7	22.58%
Neutral (NLP13)	11	35.48%
Least Beneficial (NLP15)	2	6.45%
Did not use it (NLP16)	8	25.81%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for NLP1(NLPA4)****Rate which services were the most beneficial to you:****[Readability Metrics (including the statistics)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most Beneficial (NLP11)	2	6.45%
Beneficial (NLP12)	11	35.48% <sup>170</sup>

**Field summary for NLP1(NLPA4)**

**Rate which services were the most beneficial to you:  
[Readability Metrics (including the statistics)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Neutral (NLP13)	9	29.03%
Least Beneficial (NLP15)	1	3.23%
Did not use it (NLP16)	8	25.81%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for NLP1(NLPA6)**

**Rate which services were the most beneficial to you:  
[Person and Location Extractor]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most Beneficial (NLP11)	3	9.68%
Beneficial (NLP12)	4	12.90%
Neutral (NLP13)	10	32.26%
Least Beneficial (NLP15)	3	9.68%
Did not use it (NLP16)	11	35.48%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for NLP1(NLPA7)**

**Rate which services were the most beneficial to you:  
[Requirements QA Defects]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most Beneficial (NLP11)	9	29.03%
Beneficial (NLP12)	13	41.94%
Neutral (NLP13)	5	16.13%
Least Beneficial (NLP15)	3	9.68%
Did not use it (NLP16)	1	3.23%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for NLP1(NLPA8)**

**Rate which services were the most beneficial to you:  
[Requirements QA Statistics]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most Beneficial (NLP11)	7	22.58%
Beneficial (NLP12)	5	16.13%
Neutral (NLP13)	10	32.26%
Least Beneficial (NLP15)	3	9.68%

**Field summary for NLP1(NLPA8)**

**Rate which services were the most beneficial to you:**

**[Requirements QA Statistics]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Did not use it (NLP16)	6	19.35%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for NLP2**

**What other services do you wish to see in the context of Software Requirements Engineering?**

	<b>Count</b>	<b>Percentage</b>
Answer	3	9.68%
No answer	28	90.32%
Not displayed	0	0.00%

## **D.2 Undergraduate Students Responses**

## Results

**Number of records in this query:** 18

**Total records in survey:** 18

**Percentage of total:** 100.00%

### Field summary for D1

**How do you rate your level of proficiency in English language?**

Answer	Count	Percentage
Limited Working Proficiency (can handle limited work requirements) (D11)	2	11.11%
Professional Working Proficiency (sufficient structural accuracy and vocabulary) (D12)	4	22.22%
Native or Bilingual (equivalent to that of an educated native speaker) (D13)	12	66.67%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for D3

**What is your level of experience in the area of Natural Language Processing?**

Answer	Count	Percentage
Previous academic experience (e.g., you have taken related courses) (D31)	8	44.44%
Previous industrial experience (e.g., you have worked in this area) (D32)	0	0.00%
Both academic and industrial experience (D33)	2	11.11%
None (D34)	8	44.44%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for D4(D41)

**Rank your level of expertise in the following areas:**

**[Software Programming]**

Answer	Count	Percentage
Novice (D4L1)	1	5.56%
Advanced Beginner (D4L2)	3	16.67%
Competent (D4L3)	7	38.89%
Proficient (D4L4)	7	38.89%
Expert (D4L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D4(D42)****Rank your level of expertise in the following areas:****[Software Testing]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D4L1)	3	16.67%
Advanced Beginner (D4L2)	8	44.44%
Competent (D4L3)	5	27.78%
Proficient (D4L4)	2	11.11%
Expert (D4L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D4(D43)****Rank your level of expertise in the following areas:****[Software Traceability]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D4L1)	7	38.89%
Advanced Beginner (D4L2)	8	44.44%
Competent (D4L3)	3	16.67%
Proficient (D4L4)	0	0.00%
Expert (D4L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D4(D44)****Rank your level of expertise in the following areas:****[World Wide Web Concepts]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D4L1)	2	11.11%
Advanced Beginner (D4L2)	7	38.89%
Competent (D4L3)	6	33.33%
Proficient (D4L4)	3	16.67%
Expert (D4L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D4(D45)****Rank your level of expertise in the following areas:****[Semantic Web Concepts]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D4L1)	9	50.00%
Advanced Beginner (D4L2)	7	38.89% <sup>175</sup>

**Field summary for D4(D45)****Rank your level of expertise in the following areas:****[Semantic Web Concepts]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Competent (D4L3)	2	11.11%
Proficient (D4L4)	0	0.00%
Expert (D4L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D5(D51)****Rank your level of expertise in the following areas BEFOR taking this course:****[Software Development Methodologies]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D5L1)	4	22.22%
Advanced Beginner (D5L2)	6	33.33%
Competent (D5L3)	7	38.89%
Proficient (D5L4)	1	5.56%
Expert (D5L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D5(D52)****Rank your level of expertise in the following areas BEFOR taking this course:****[Requirements Elicitation and Evaluation]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D5L1)	7	38.89%
Advanced Beginner (D5L2)	9	50.00%
Competent (D5L3)	2	11.11%
Proficient (D5L4)	0	0.00%
Expert (D5L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D5(D53)****Rank your level of expertise in the following areas BEFOR taking this course:****[Requirements Specification (documentation)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D5L1)	8	44.44%
Advanced Beginner (D5L2)	7	38.89%
Competent (D5L3)	2	11.11%
Proficient (D5L4)	1	5.56%

176

**Field summary for D5(D53)****Rank your level of expertise in the following areas BEFOR taking this course:  
[Requirements Specification (documentation)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Expert (D5L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D5(D54)****Rank your level of expertise in the following areas BEFOR taking this course:  
[Requirements Quality Assurance]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D5L1)	8	44.44%
Advanced Beginner (D5L2)	9	50.00%
Competent (D5L3)	1	5.56%
Proficient (D5L4)	0	0.00%
Expert (D5L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D5(D55)****Rank your level of expertise in the following areas BEFOR taking this course:  
[Domain Modeling (UML, RDF, Ontologies)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D5L1)	4	22.22%
Advanced Beginner (D5L2)	8	44.44%
Competent (D5L3)	5	27.78%
Proficient (D5L4)	1	5.56%
Expert (D5L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D6(D61)****Rank your level of expertise in the following areas AFTER taking this course:  
[Software Development Methodologies]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D6L1)	3	16.67%
Advanced Beginner (D6L2)	6	33.33%
Competent (D6L3)	5	27.78%
Proficient (D6L4)	4	22.22%
Expert (D6L5)	0	0.00%
No answer	0	0.00%

177



**Field summary for D6(D61)**

**Rank your level of expertise in the following areas AFTER taking this course:  
[Software Development Methodologies]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Not displayed	0	0.00%

**Field summary for D6(D62)**

**Rank your level of expertise in the following areas AFTER taking this course:  
[Requirements Elicitation and Evaluation]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D6L1)	3	16.67%
Advanced Beginner (D6L2)	4	22.22%
Competent (D6L3)	9	50.00%
Proficient (D6L4)	2	11.11%
Expert (D6L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D6(D63)**

**Rank your level of expertise in the following areas AFTER taking this course:  
[Requirements Specification (documentation)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D6L1)	3	16.67%
Advanced Beginner (D6L2)	5	27.78%
Competent (D6L3)	7	38.89%
Proficient (D6L4)	3	16.67%
Expert (D6L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for D6(D64)**

**Rank your level of expertise in the following areas AFTER taking this course:  
[Requirements Quality Assurance]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Novice (D6L1)	3	16.67%
Advanced Beginner (D6L2)	5	27.78%
Competent (D6L3)	8	44.44%
Proficient (D6L4)	2	11.11%
Expert (D6L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for D6(D65)

Rank your level of expertise in the following areas AFTER taking this course:  
[Domain Modeling (UML, RDF, Ontologies)]

Answer	Count	Percentage
Novice (D6L1)	2	11.11%
Advanced Beginner (D6L2)	3	16.67%
Competent (D6L3)	10	55.56%
Proficient (D6L4)	3	16.67%
Expert (D6L5)	0	0.00%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for SA1

Which one of the following applications did you use for the documentation of your assignment?

Answer	Count	Percentage
OpenOffice.org Writer (SA11)	7	38.89%
ReqWiki (SA12)	8	44.44%
Other	3	16.67%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for SA2

Which one of the following approaches did you use for the quality assessment of your SRS documents?

Answer	Count	Percentage
The Semantic Assistants services (SA11)	9	50.00%
I did it manually (SA12)	9	50.00%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for SA3

Why did you choose to manually assess the quality of your SRS documents?

Answer	Count	Percentage
Answer	9	50.00%
No answer	0	0.00%
Not displayed	9	50.00%

### Field summary for SA4

How easy-to-use did you find the Semantic Assistants user interface in the wiki?

Answer	Count	Percentage
Very Easy (SA41)	2	11.11%

#### Field summary for SA4

##### How easy-to-use did you find the Semantic Assistants user interface in the wiki?

Answer	Count	Percentage
Easy (SA42)	2	11.11%
Neutral (SA43)	1	5.56%
Difficult (SA44)	0	0.00%
Very Difficult (SA45)	1	5.56%
No answer	0	0.00%
Not displayed	12	66.67%

#### Field summary for SA5

##### How easy-to-use did you find the Semantic Assistants OpenOffice plug-in user interface?

Answer	Count	Percentage
Very Easy (SA51)	1	5.56%
Easy (SA52)	2	11.11%
Neutral (SA53)	0	0.00%
Difficult (SA54)	0	0.00%
Very Difficult (SA55)	0	0.00%
No answer	0	0.00%
Not displayed	15	83.33%

#### Field summary for SA6

##### Check all the difficulties you encountered while using the Semantic Assistants plug-in:

Answer	Count	Percentage
Browser Incompatibility (SA61)	2	11.11%
Slow Response Time (SA62)	0	0.00%
Results were hard to read/understand (SA63)	2	11.11%
None of above (SA64)	2	11.11%
Other	4	22.22%

#### Field summary for SA7

##### Check all the difficulties you encountered while using the Semantic Assistants plug-in:

Answer	Count	Percentage
It was hard to install (SA71)	0	0.00%
Slow Response Time (SA72)	0	0.00%
Results were hard to read/understand (SA73)	1	5.56%
None of above (SA74)	2	11.11%
Other	1	5.56%

**Field summary for SA8(SA81)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Collaborative environment]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	1	5.56%
Beneficial (SAL82)	5	27.78%
Neutral (SAL83)	0	0.00%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	12	66.67%

**Field summary for SA8(SA82)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Automatic versioning of pages]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	1	5.56%
Beneficial (SAL82)	4	22.22%
Neutral (SAL83)	1	5.56%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	12	66.67%

**Field summary for SA8(SA83)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Pre-defined Queries]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	0	0.00%
Beneficial (SAL82)	5	27.78%
Neutral (SAL83)	1	5.56%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	12	66.67%

**Field summary for SA8(SA84)**

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Pre-defined Traceability Links]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	5	27.78%
Beneficial (SAL82)	1	5.56%
Neutral (SAL83)	0	0.00%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
	181	

**Field summary for SA8(SA84)****Rank which one of the ReqWiki features were the most beneficial to you:  
[Pre-defined Traceability Links]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Not displayed	12	66.67%

**Field summary for SA8(SA85)****Rank which one of the ReqWiki features were the most beneficial to you:  
[SRS Forms (data entry)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	3	16.67%
Beneficial (SAL82)	3	16.67%
Neutral (SAL83)	0	0.00%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	12	66.67%

**Field summary for SA8(SA86)****Rank which one of the ReqWiki features were the most beneficial to you:  
[SRS Templates (data presentation)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	4	22.22%
Beneficial (SAL82)	0	0.00%
Neutral (SAL83)	2	11.11%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	12	66.67%

**Field summary for SA8(SA87)****Rank which one of the ReqWiki features were the most beneficial to you:  
[Ease of navigation between entities]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (SAL81)	1	5.56%
Beneficial (SAL82)	3	16.67%
Neutral (SAL83)	2	11.11%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	12	66.67%

### Field summary for SA8(SA88)

**Rank which one of the ReqWiki features were the most beneficial to you:  
[Ease of finding inconsistent/missing entities]**

Answer	Count	Percentage
Most beneficial (SAL81)	2	11.11%
Beneficial (SAL82)	3	16.67%
Neutral (SAL83)	1	5.56%
Least beneficial (SAL84)	0	0.00%
No answer	0	0.00%
Not displayed	12	66.67%

### Field summary for SA9

**Given the experience of using ReqWiki, which application would you use for Software Requirements Engineering in future?**

Answer	Count	Percentage
ReqWiki-like System with semantic support (SA91)	8	44.44%
Traditional Wiki (SA92)	0	0.00%
Traditional Word Processor (e.g., OpenOffice Writer, Microsoft Word) (SA93)	0	0.00%
Other	0	0.00%
No answer	0	0.00%
Not displayed	10	55.56%

### Field summary for SA10(SA101)

**Rank which one of the Semantic Assistants plug-in were the most beneficial to you:  
[Annotation Highlighting]**

Answer	Count	Percentage
Most beneficial (S10L1)	1	5.56%
Beneficial (S10L2)	2	11.11%
Neutral (S10L3)	0	0.00%
Least beneficial (S10L4)	0	0.00%
No answer	0	0.00%
Not displayed	15	83.33%

### Field summary for SA10(SA102)

**Rank which one of the Semantic Assistants plug-in were the most beneficial to you:  
[Interactive Annotations]**

Answer	Count	Percentage
Most beneficial (S10L1)	0	0.00%
Beneficial (S10L2)	2	11.11%
Neutral (S10L3)	0	0.00%

183

**Field summary for SA10(SA102)**

**Rank which one of the Semantic Assistants plug-in were the most beneficial to you:  
[Interactive Annotations]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Least beneficial (S10L4)	1	5.56%
No answer	0	0.00%
Not displayed	15	83.33%

**Field summary for SA10(SA103)**

**Rank which one of the Semantic Assistants plug-in were the most beneficial to you:  
[Customizing User Interface (font size)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (S10L1)	0	0.00%
Beneficial (S10L2)	1	5.56%
Neutral (S10L3)	2	11.11%
Least beneficial (S10L4)	0	0.00%
No answer	0	0.00%
Not displayed	15	83.33%

**Field summary for SA10(SA104)**

**Rank which one of the Semantic Assistants plug-in were the most beneficial to you:  
[Show/Hide Annotation content or features]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most beneficial (S10L1)	1	5.56%
Beneficial (S10L2)	1	5.56%
Neutral (S10L3)	1	5.56%
Least beneficial (S10L4)	0	0.00%
No answer	0	0.00%
Not displayed	15	83.33%

**Field summary for SA11**

**Given the experience of using the Natural Language Processing, would you use it again for  
Software Requirements Engineering tasks in future?**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Yes (SA111)	3	16.67%
No (specify the reason in the box) (SA112)	0	0.00%
Comments	0	0.00%
No answer	0	0.00%
Not displayed	15	83.33%

### Field summary for NLP1(NLPA1)

Rate which services were the most beneficial to you:

[Information Extractor]

Answer	Count	Percentage
Most Beneficial (NLP11)	2	11.11%
Beneficial (NLP12)	3	16.67%
Neutral (NLP13)	2	11.11%
Least Beneficial (NLP15)	0	0.00%
Did not use it (NLP16)	11	61.11%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for NLP1(NLPA2)

Rate which services were the most beneficial to you:

[Writing Quality]

Answer	Count	Percentage
Most Beneficial (NLP11)	5	27.78%
Beneficial (NLP12)	5	27.78%
Neutral (NLP13)	2	11.11%
Least Beneficial (NLP15)	0	0.00%
Did not use it (NLP16)	6	33.33%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for NLP1(NLPA3)

Rate which services were the most beneficial to you:

[English Durm Indexer]

Answer	Count	Percentage
Most Beneficial (NLP11)	0	0.00%
Beneficial (NLP12)	3	16.67%
Neutral (NLP13)	4	22.22%
Least Beneficial (NLP15)	1	5.56%
Did not use it (NLP16)	10	55.56%
No answer	0	0.00%
Not displayed	0	0.00%

### Field summary for NLP1(NLPA4)

Rate which services were the most beneficial to you:

[Readability Metrics (including the statistics)]

Answer	Count	Percentage
Most Beneficial (NLP11)	0	0.00%
Beneficial (NLP12)	5	27.78% <sup>185</sup>



**Field summary for NLP1(NLPA4)**

**Rate which services were the most beneficial to you:  
[Readability Metrics (including the statistics)]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Neutral (NLP13)	2	11.11%
Least Beneficial (NLP15)	0	0.00%
Did not use it (NLP16)	11	61.11%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for NLP1(NLPA6)**

**Rate which services were the most beneficial to you:  
[Person and Location Extractor]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most Beneficial (NLP11)	0	0.00%
Beneficial (NLP12)	4	22.22%
Neutral (NLP13)	3	16.67%
Least Beneficial (NLP15)	1	5.56%
Did not use it (NLP16)	10	55.56%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for NLP1(NLPA7)**

**Rate which services were the most beneficial to you:  
[Requirements QA Defects]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most Beneficial (NLP11)	5	27.78%
Beneficial (NLP12)	4	22.22%
Neutral (NLP13)	1	5.56%
Least Beneficial (NLP15)	0	0.00%
Did not use it (NLP16)	8	44.44%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for NLP1(NLPA8)**

**Rate which services were the most beneficial to you:  
[Requirements QA Statistics]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Most Beneficial (NLP11)	1	5.56%
Beneficial (NLP12)	3	16.67%
Neutral (NLP13)	3	16.67%
Least Beneficial (NLP15)	0	0.00%

**Field summary for NLP1(NLPA8)**

**Rate which services were the most beneficial to you:**

**[Requirements QA Statistics]**

<b>Answer</b>	<b>Count</b>	<b>Percentage</b>
Did not use it (NLP16)	11	61.11%
No answer	0	0.00%
Not displayed	0	0.00%

**Field summary for NLP2**

**What other services do you wish to see in the context of Software Requirements Engineering?**

	<b>Count</b>	<b>Percentage</b>
Answer	7	38.89%
No answer	11	61.11%
Not displayed	0	0.00%