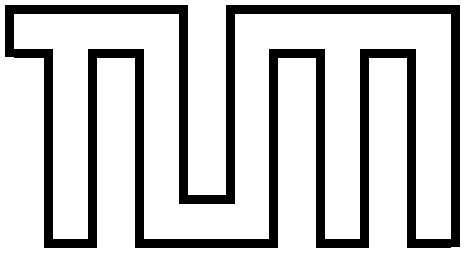
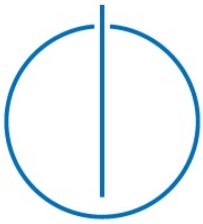


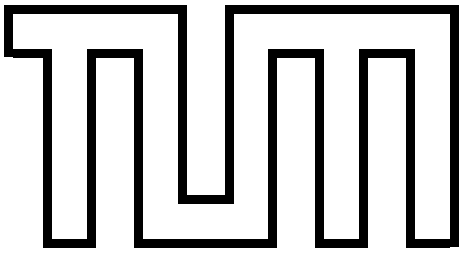
Fakultät für Informatik
der Technische Universität München

Master's Thesis in Informatics

Management of Complex Product Ontologies Using a Web-Based Natural Language Processing Interface

A B M Junaed





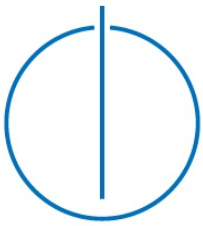
Fakultät für Informatik
der Technische Universität München

Master's Thesis in Informatics

Management komplexer Produktontologien mittels einer web-basierten Natürliche-Sprachverarbeitungs-Oberfläche

Management of Complex Product Ontologies Using a Web-Based Natural Language Processing Interface

Author: A B M Junaed
Supervisor: Prof. Dr. Florian Matthes
1st Advisor: Jörg Landthaler, M.Sc.
2nd Advisor: Tim Lochow, Dipl.-Ing.
Submission: 15.06.2016



I assure the single handed composition of this master's thesis only supported by declared resources.

München, 15.06.2016

(A B M Junaed)

Acknowledgment

First and above all, I praise and thank Almighty ALLAH for everything.

I would like to thank Prof. Dr. Florian Matthes for giving me the opportunity to write my Master thesis in the Software Engineering for Business Information Systems (SEBIS) chair at TU Munich.

I earnestly thank my two advisors, Jörg Landthaler and Tim Lochow (Airbus Group Innovations) for their continuous motivation, extensive advice and great support throughout this research work. In addition, I would like to thank Airbus Group Innovations TX4 team members. I enjoyed working in such a friendly place with nice colleagues.

Also, special thanks to each and every person who was somehow involved in the development of this work. To all my interviewees, to all the people I consulted: all your valuable feedback and inputs were of great help, I really appreciate the time you took to help me out.

Last but not least, I thank my parents and my family for always supporting me towards my dream.

Abstract

Complex products such as commercial aircraft and the associated development activities require complex engineering processes which are supported by IT tools. The IT landscape is characterized by a set of very heterogeneous engineering tools providing data in different formats, e.g., relational databases, XML, CSV, XLS, and alike. However, there is no single API to access the data. A key approach at Airbus Group Innovations (AGI[1]) to tackle this problem is the application of Linked Data and Semantic Web technology. Web Ontology Language (OWL) is one of the core components of the Semantic Web stack, which is used for knowledge representation (KR). But, OWL also comes with a learning curve for the domain experts who have little or no knowledge in ontology engineering. To minimize the learning curve to access such data, the primary purpose of this thesis is to provide a solution consisting of a web-based natural language interface (NLI) to manage complex product ontologies. The purpose of the NLI is to guide the domain experts to create and update OWL ontologies, as well as to search the ontology to find out the inconsistencies or missing concepts. An additional contribution of this thesis is to provide a mechanism to reuse existing ontologies through the NLI.

We apply a design science methodology to tackle the challenges. A state of the art study was conducted to detect a useful basis for our solution. As part of the solution, a web-based prototype is presented, which uses an NLI. This NLI is based on a CNL and a Semantic Wiki. Moreover, our prototype can reuse existing ontologies by importing them into the system and by creating domain specific lexicons automatically.

The prototype was evaluated in several ways. Qualitative interviews with Airbus stakeholders yielded valuable feedback. Moreover, several real ontologies of an aircraft were used to carry out functional tests and these tests showed promising results. The prototype is also highly portable to different OWL ontologies, since it does not require any customization.

[1] Airbus Group Innovations is the corporate research and technology center of Airbus Group.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

1.1 Motivation

Airbus Group programs are characterized by complex engineering and development activities across several disciplines, lifecycle phases, and geographical sites. From an IT perspective, it is challenging to support collaborative tasks across disciplines and international teams. The IT landscape is characterized by a set of very heterogeneous engineering tools providing data in different formats, e.g., relational databases, XML, CSV, XLS, etc. As a result, there is no unique API to access the data. Getting a global view of the actual status of individual engineering and manufacturing tasks, creating efficient automated data flows or simply gathering highly interlinked information in one common view is difficult. Searching and reasoning across the data to detect inconsistencies in the design can provide a major benefit to the quality of the data being transmitted between engineering and manufacturing stakeholders. Furthermore, complex products such as commercial aircraft and the associated development processes impose a significant amount of terminology and semantic knowledge, which can hardly be managed by individual engineers. This is a fact which causes recursive rework of inherited inconsistencies due to the lack of official product references across (all) disciplines in the development or manufacturing process.

A Key approach at Airbus Group Innovations (AGI[1]) to solve this problem is directed towards Linked Data and Semantic web technology: apply semantic web technologies to publish data in universal formats and to draw connections between data sources which gives linked data. This data is accessible via the same kind of API. Web Ontology Language (OWL) [WMS04] is one of the core components of semantic web stack which is used for knowledge representation (KR), and it includes descriptions of classes, properties,

[1] Airbus Group Innovations is the corporate research and technology Centre of Airbus Group.

and their instances. Moreover, OWL brings reasoning capability since it is based on description logic. But OWL also adds a learning curve for the domain experts who are knowledgeable about a particular area but have little or no knowledge in ontology engineering.

Many user-friendly interfaces have been developed to minimize the learning curve to access and use OWL ontologies. Some of them solve the problem by providing a graphical user interface through which users can browse the data (e.g., Protégé [No01]), others developed a form-based interface to provide search functionality, e.g., KIM Platform [Ki03]. TopBraid Composer[2] is another front-end tool to work with OWL ontologies. However, Dzbor et al. compared between TopBraid Composer and Protégé (version 3.x) in [Dz06]. They gave a general conclusion that the quality of current tools is acceptable when used by experts in logic, but there is a high learning curve with newcomers. Moreover, an interface evaluation study done by Kaufmann and Bernstein shows that the most acceptable systems by the end users are the systems which support Natural Language Interfaces (NLI) [KB07]. Therefore, an appropriate NLI is required which will provide domain experts an easy to use interface to access OWL ontologies. In this thesis, we are going to focus on this topic.

1.2 Problem Description

Here we will conduct our research to provide a solution consisting of a web-based natural language interface (NLI), which will guide domain experts to create and update OWL ontologies, as well as to search the ontology to find out the inconsistencies or missing concepts. Moreover, our research will also focus on importing existing ontologies into the NLI, which will be beneficial to reuse existing ontologies. As an exemplary test case of real life data, an Airbus cabin ontology will be used for development; then the proposed solution will be evaluated with other real life data set also. Since natural language itself inherently possesses some challenges which are discussed in Section 1.3, those also have to be tackled carefully.

1.3 Technical Challenges for NLI

Building NLIs to structured data requires handling various challenges. The major challenges are:

1. **Guiding/usability:** Guide the user to write natural language query without any error to search or to add knowledge into the knowledge base, while providing

[2]http://www.topbraidcomposer.com

minimum training for the user and keeping the supported language intuitive.

2. **Ambiguity:** Understanding Natural Language inherently possesses challenges, e.g. ambiguity [CP82]. One sentence can give multiple interpretations and selecting the correct one is challenging.

3. **Portability:** Portability means to be able to use an NLI system easily with different domains, that means domain independence. Even if portable NLIs are much more useful than domain specific systems, constructing transportable systems causes various technical and theoretical problems [Gr87]. In addition, portability affects retrieval performance: "the more a system is tailored to a domain, the better its retrieval performance is", as stated by Kaufmann and Bernstein[KB07].

4. **Hiding complexities of the knowledge structure:** Hiding the underlying complexities of the structured knowledge from the end user while showing results is a real challenge for NLIs.

1.4 Research Questions

The research questions which will be tackled in this study are grouped into two different categories:

1. **Major research questions:** Primary focus of this thesis will be given towards the following research questions:

   1.1. How to create an OWL ontology using a web-based NLI?

   1.2. How to search in OWL ontology using a web-based NLI?

   1.3. How to incorporate existing ontologies into the proposed NLI?

   1.4. How to create domain specific lexicon automatically from existing ontologies?

2. **Derived research questions:** Following research questions are derived from the challenges associated with NLIs:

   2.1. How to guide the user to formulate queries, to add and edit data into knowledge base by providing minimum training while keeping the supported language intuitive?

   2.2. How to resolve the ambiguity of natural language?

   2.3. How to keep the NLI portable?

   2.4. How to hide the underlying complexities of the structured knowledge from the end user?

1.5 Research Methodology

Hevner et al. presented an information systems (IS) research framework accompanied by a set of seven research guidelines [Al04]. These guidelines are followed in this thesis and presented in Figure 1.1. At first, the problem relevance is explained in Section 1.2,

Figure 1.1: Research method of this thesis following the guidelines provided by Hevner et al. in [Al04]

which describes the importance of the NLI for managing complex ontologies. According to the design as a search process guideline, design alternatives were generated and tested against our requirements to discover an effective solution for the defined problem space. A prototype is developed as an artifact, which will be presented in Section 7.1. The artifact is evaluated by domain experts and by functionality test, which will be discussed in Chapter 9. The contribution of this research is the artifact itself, and the communication of the research refers to the finalization of the written report, i.e. this thesis.

1.6 Outline

This thesis consists of five parts. Part I contains this Chapter and Chapter 2. Chapter 2 entitled “Scientific Background” describes the background of the fields of Semantic Web and Linked Data.

Part II discusses relevant prior and related work. NLIs to KBs are discussed in Chapter 3. Chapter 4 describes Semantic Wikis.

Part III contains our scientific contribution which consists of four Chapters, from Chapter 5 to Chapter 8. Existing NLIs to KBs are compared in Chapter 5. Conceptual design of the prototype is presented in Chapter 6. Chapter 7 demonstrates the design and architecture of the final system. After that, the prototypical implementation is presented in Chapter 8.

Evaluation of the prototypical implementation is presented in Part IV.

Part V presents future work and conclusion. Chapter 10 presents ideas for future work and Chapter 11 concludes this thesis.

Finally, the appendices of this thesis are presented. Appendix A contains the list of abbreviations used in this thesis. Appendix B demonstrates the user guide and finally, Appendix C contains the questionnaire which is used for the expert interviews

Chapter 2

Scientific Background

2.1 Semantic Web

Tim Berners-Lee, the inventor of the World Wide Web (WWW), proposed a new generation of the Web [BL99], called the Semantic Web: where all information on the Web will be inter-operable and understandable by computers. As discussed by Guha et al., the idea is to define and link data on the Web in such a way that applications can use this data for more efficient discovery, integration and automation [GMM03]. Furthermore, the Semantic Web will define structured relations among different types of resources. Moreover, each resource can have metadata attached to it.

2.1.1 Semantic Web Basics

Figure 2.1 shows the Semantic Web stack where each language is represented by a block. The blocks are presented hierarchically, and each block uses capabilities of the layers below. Here we will discuss the most important basics of Semantic Web, which are related to our study.

RDF: The Resource Description Framework (RDF) is a formal language that defines the basic unit of the Semantic Web as a triple. Each triple has three parts:

$$Subject \quad Predicate \quad Object$$

RDF schema is presented by the shorthand form RDFs.

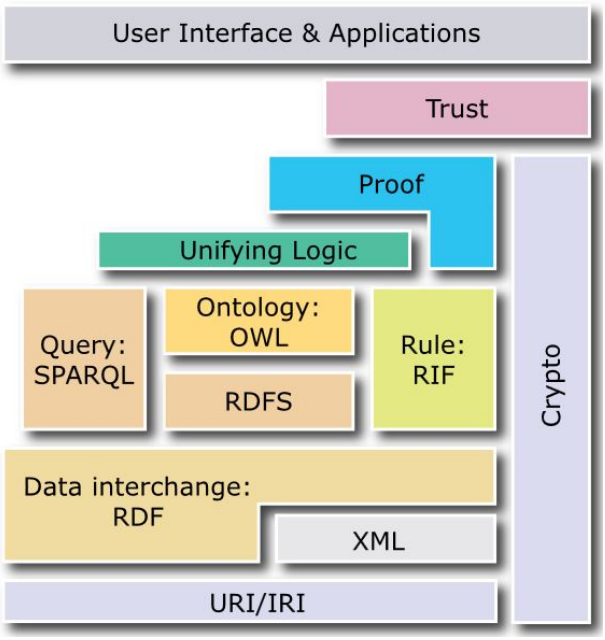


Figure 2.1: Semantic Web stack as visualized by the World Wide Web Consortium (http://w3.org). This image originates from http: //www.w3.org/2007/03/layerCake.svg

OWL: "An ontology is a formal explicit description of concepts in a domain of discourse (classes (sometimes called concepts)), properties of each concept describing various features and attributes of the concept (slots (sometimes called roles or properties)), and restrictions on slots (facets (sometimes called role restrictions)). An ontology together with a set of individual instances of classes constitutes a knowledge base (KB)."[1] ABox refers to the actual data such as instances or individuals that are generated based on the definitions in the ontology. OWL (Web Ontology Language) is recommended by W3C organisation for publishing and sharing ontologies on the World Wide Web [2].

SPARQL: SPARQL (pronounced "sparkle"), is an SQL-like query format which is designed to query the underlying triples of the Semantic Web.

OWL Editors

In order to work with OWL ontologies, there are several front-end tools, such as — TopBraid Composer[3] , Protégé[4] , SWOOP[5] , OntoStudio[6] . Dzbor et al. compared

[1]http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html
[2]http://www.w3.org/TR/owl-ref/
[3]http://www.topbraidcomposer.com
[4]http://protege.stanford.edu
[5]http://www.mindswap.org/2004/SWOOP/
[6]http://www.ontoprise.de

TopBraid Composer and Protégé (version 3.x) in [Dz06], where users were selected from both novices and experts. Several problems were reported in the study: the users encountered difficulties to get an overview of the usage of classes/properties, ontology visualization was not helpful for the users, etc. The authors gave a general conclusion that the quality of the current tools is acceptable when used by experts in logic, but there is a high learning curve with newcomers.

2.2 Linked Data

The term Linked Data gained in popularity after the initiative of the Linked Open Data[7] project. According to the Web site of Linked Open Data project, the term Linked Data refers to "a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF." Described by Tim Berners-Lee, Linked Data is "the Semantic Web done right".

Tim Berners-Lee stated four rules for Linked Data[8], which are stated below:

1. "Use URIs as names for things"
2. "Use HTTP URIs so that people can look up those names"
3. "When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)"
4. "Include links to other URIs. so that they can discover more things"

These rules are not mandatory, but important to make data interconnected.

[7]http://linkeddata.org/
[8]https://www.w3.org/DesignIssues/LinkedData.html

Part II

Related Work

Chapter 3

Natural Language Interfaces to Knowledge Bases

This Chapter will focus on different NLI systems available to work with ontologies, which can be grouped into two broad categories. The first category is Question Answering (QA) systems which translate a Natural Language query into the formal query language e.g. SPARQL to retrieve data. The second category uses CNL to work with OWL ontologies. After discussing these two types in Section 3.1 and 3.2 respectively, we will discuss about semantic wikis in Section 4, which can be useful to represent data to the end user and to hide the underlying complexity at the same time.

3.1 Question-Answering Systems

As argued by Unger et al. in [Un12], Question Answering approaches allow users to express queries in natural language without being aware of the underlying schema or query language, which is a good compromise between intuitiveness and expressivity. Several question answering systems have been proposed in the past to work with OWL ontologies or RDF data, for example, Aqualog [LM04], NLP-Reduce [KBF07], FREyA [LB11] and AutoSparql [LB11].

AquaLog [LM04] is a portable question-answering system which mainly supports factual queries beginning with what, which, who and the like. Customization of AquaLog can increase the performance of the system though it is not mandatory. Customization includes associating certain words with relevant concepts from the ontology, e.g., where needs to be associated with ontology classes which represent a location, similarly, who needs to be associated with classes which represent a person or organization. To properly handle, AquaLog needs the ontology to have a simpler hierarchy structure; and the terms in a

query can only refer to ontology concepts between which the path length is not more than two. To evaluate portability, AquaLog was tested against the wine ontology[1] . As customization process, words like where, when, and who were associated with relevant ontology classes; then synonyms of several ontology resources were added manually. As reported by Lopez et al. [Lo07], this step was not mandatory, but it increases the recall because of the limitations of WordNet[2] coverage. Only 17.64% of questions were correctly handled by this system.

NLP-Reduce [KBF07] uses a reduced set of NLP operators(such as synonym expansion and stemming), hence its name is NLP-Reduce. This system is robust to deficient input, ungrammatical input and allows users to enter keywords, sentence fragments, or full English sentences. It tries to link the words of a query and their synonyms to the expressions in a KB. SPARQL query generator is the core component of NLP-Reduce which attempts to match the query words to the synonym-enhanced triples stored in the lexicon to generate SPARQL queries. Since it abandons any complex NLP techniques, it cannot answer queries which require a dependency structure of the sentence elements. (e.g., "Which restaurants are closer to Munich?") As pointed out by Damljanovic in [Da], the relaxation of supported queries seems to have a negative impact on the performance. Its performance was lower than that of similar systems when evaluated with geography and restaurants datasets provided by Mooney[3].

AutoSPARQL [LB11] carried out another possibility to translate NL to SPARQL by using SPARQL templates. At first, the input is processed by the part of speech (POS) tagger. Based on the POS tags, lexical entries are created using a set of heuristics. These lexical entries and predefined domain independent lexical entries are used for parsing. Then the SPARQL templates are generated with slots. To fill these slots, entities (i.e., classes, instances or properties) are retrieved from knowledge base using string similarities and NLP. For each slot and a possible entity, different query candidates are found and ranked. Finally, the best answer is returned to the user. Since the generated SPARQL templates capture the semantic structure of the natural language input, questions containing quantifiers (e.g., the most, more than), comparatives (e.g., higher than) and superlatives (e.g., the highest) do not pose a problem in contrast to other question answering systems that map NL input to purely triple-based representations. This approach was tested against the benchmark dataset of the 1st Workshop on Question Answering over Linked Data (QALD)[4], which defines 50 questions to DBpedia and their answers. This system generated the correct answer for 19 questions after manually correcting erroneous POS tags in seven questions, as mentioned in [Un12].

[1]http://www.w3.org/TR/2003/CR-owl-guide-20030818/

[2]https://wordnet.princeton.edu/

[3]The Mooney geography dataset is available from http://www.ifi.uzh.ch/ddis/research/talking-to-the-semantic-web/owl-test-data/

[4]http://www.sc.cit-ec.uni-bielefeld.de/qald-1

FREyA [DAC10], which combines syntactical parsing with the knowledge encoded in ontologies to interpret a natural language question. If the system fails to derive an answer automatically, it involves the user through clarification dialogs. Selections made by the user are saved and used to train the system to improve its performance over time. Involving the user for clarification has one significant advantage, it minimizes customization of the NLI system. On the other hand, the naive end-user is often not knowledgeable about the modeling and vocabulary of the underlying data and thus is not able to help. Evaluation of FREyA against Mooney geography dataset showed very high precision and recall, as reported by [DAC10].

3.2 Controlled Natural Language

Controlled natural languages (CNLs) are subsets of natural languages, where grammar and vocabulary are restricted to eliminate or reduce complexity and ambiguity. There are mainly two major types of CNLs: the first category comprises those that improve readability for human readers (e.g. non-native speakers), and the second category contains those that enable reliable automatic semantic analysis of the language.

We are interested in the CNLs which can work with Semantic Web technologies, these languages are formal and focuses on unambiguity instead of focusing solely on improving human readability. Recently, several CNLs has been proposed which can work with Semantic Web and some can be translated to OWL also. In this Section we will discuss about different CNLs which can work with Semantic Web.

3.2.1 Attempto Controlled English (ACE)

Attempto Controlled English (ACE) is a logic language with an English syntax, which is one of the most mature controlled natural languages and under active development for more than 20 years since 1995. Fuchs and Schwitter first introduced ACE [FS96], more than 60 scientific papers have been published by the Attempto group[5] since then. Google Scholar lists more than 1000 articles containing the term "Attempto Controlled English"[6], which makes it probably the most widespread CNL in academia(retrieved in April 2016). ACE is not domain specific, and it is a general-purpose CNL. It has an exchangeable vocabulary and thus can be adapted to particular problem areas. A large part of natural English is covered by ACE: singular and plural noun phrases, active and passive voice, relative phrases, anaphoric references, existential and universal quantifiers, negation, and

[5] http://attempto.ifi.uzh.ch/site/pubs/

[6] http://scholar.google.com/scholar?q=%22Attempto+Controlled+English%22

much more. There is a reference implementation of ACE, named Attempto Parsing Engine (APE). APE translates ACE texts into logic. The source code of APE is open source.

ACE Parser (APE)

APE implements the concrete grammar of ACE. It is also used to translate ACE to OWL/SWRL. At first, the translator tries to transform an ACE text into its corresponding discourse representation structure (DRS), such DRS can be mapped to first-order logic in a direct and simple way. In next step, the translator attempts to convert the DRS into OWL by using the algorithm mentioned in [KA07]. In the case of failure, an attempt is made to convert the implication into Semantic Web Rule Language (SWRL). An error message is generated if this case fails as well. So, the final output is either a pure OWL ontology, or an ontology that mixes OWL and SWRL.

OWL-Verbalizer

OWL verbalizer translates OWL to ACE. It takes an OWL ontology in OWL 1.1 XML syntax as input and converts it into OWL 1.1 Functional-Style Syntax in Prolog notation. Then the resulting ontology is verbalized in ACE as described in Section 5.6.6. SWI-Prolog is used to implement the OWL verbalizer.

Predictive Editor

A predictive editor is provided as part of the AceWiki to help the users to write valid ACE sentences. The predictive editor shows step-by-step the words that are syntactically possible at a given position in the sentence. Without it, one has to learn the grammar of ACE, which is time-consuming, and there is chance to make some mistakes also.

3.2.2 Rabbit

Rabbit [Do07] (for a shorter overview see [HDG07]) is another CNL and used as an OWL-compatible knowledge representation language. The primary goal of Rabbit is to help writing OWL ontologies, i.e. the focus is on the Rabbit $\rightarrow$ OWL direction. The domain expert has to be familiar with Rabbit and treated as the main author of Rabbit texts, on the other hand, the knowledge expert has to be familiar with both Rabbit and OWL, and plays the main role to convert Rabbit texts into OWL. In the evaluation of Rabbit with domain experts, it was found that each generated ontology was different, and the knowledge engineers had to correct those ontologies [De09].

3.2.3 Rabbit to OWL Ontology Authoring (ROO)

Rabbit to OWL Ontology Authoring (ROO) [De08] is an ontology editing tool developed by the University of Leeds. It is an open source plug-in for Protégé and supports the domain expert to create and edit ontologies using Rabbit.

3.2.4 CLOnE

CLOnE(Controlled Language for Ontology Editing) [Fu07] is another CNL, which aims to ontology authoring. Its conversion to OWL is built on top of the GATE[7] framework. Only a tiny fragment of OWL is targeted in CLOnE, where domain, range, subclass, and individual assertion axioms can be expressed. Eleven sentence patterns are used to compose the grammar of CLOnE, which roughly correspond to eleven OWL axiom patterns.

Summary

In this Chapter, we have studied different NLIs to work with ontologies. QA systems translate an NL input into the formal query languages such as SPARQL. On the other hand, CNLs restrict the user by using predefined grammar rules and hence resolve the ambiguity. In the next Chapter, we will discuss about Semantic Wikis, which can simplify ontology engineering.

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

Chapter 4

Semantic wikis

The online encyclopedia Wikipedia[1] has shown that the philosophy of wikis has been a huge success. Using wikis, users can contribute to Web representations with basic Web editing skills. Since the start of Wikipedia in 2001, more than 40 million articles have been created in different languages[2]. Properties of the Wikis, e.g., ease of use, collaboration, linking, etc. has made it so popular. A flexible and dynamic way to share knowledge is provided by the Wiki systems.

Techniques of the Semantic Web (i.e. enriching the data on the web with well-defined meaning) and the philosophy of wikis (i.e. quick and easy editing of textual content in a collaborative way over the web, collaboration, linking) are combined in Semantic Wikis. The Semantic Wikis can significantly simplify the ontology engineering as discussed by Schaffert et al. in [Sc08]. Section 4.1 discusses about ontology management using Semantic Wikis, and different types of Semantic Wikis are discussed in Section 4.2 and 4.3.

4.1 Ontology Management Using Semantic Wikis

As discussed by Tobias Kuhn in [Ku10a], Semantic Wikis show signs of being a promising approach to get the domain experts more involved in creating and maintaining ontologies. Moreover, this could increase the quality and number of available ontologies. Ontology engineering can be significantly simplified by using Semantic Wikis, as discussed by Schaffert et al. in [Sc08]. Semantic wikis can also hide the underlying complexities of Semantic Web from domain experts. Domain experts can create Wiki pages with textual descriptions, and then the knowledge can successively be formalized and refined by close interaction between domain experts and computer scientists and by using other

[1] https://www.wikipedia.org/

[2] http://meta.wikimedia.org/wiki/List_of_Wikipedias retrieved in June, 2016

tools like Protégé. In contrast to the ontology editors like Protégé, Semantic Wikis allow the domain experts to update and maintain an ontology without having knowledge of the underlying formalism.

According to Tobias Kuhn [Ku10a], Semantic Wikis can be broadly categorized into two types: text-centered and logic-centered. Section 4.2 and 4.3 will discuss Text-centered Semantic Wikis and Logic-centered Semantic Wikis respectively.

4.2 Text-Centered Semantic Wikis

Text-centered approaches use semantic annotations to enrich classical wiki environments.

Semantic MediaWiki [Vö06] is a well-known example of text-centered semantic wikis. It emphasizes scalability and backward compatibility. It is one of the most mature existing semantic wiki engines. The MediaWiki engine (which is used e.g. for Wikipedia) is used to build it. Similar to tagging systems, users can add new annotations as needed. No predefined schema or ontology is needed for annotations. However, Semantic MediaWiki does not support inferencing and similar advanced functionalities.

IkeWiki presented by Schaffert in [Sc06], is another matured and popular text-centered semantic wiki engine. It is a Java based web application and supports collaborative knowledge management. In contrast to Semantic MediaWiki, it provides advanced semantic functionalities like reasoning. Pages from Wikipedia can be imported here and can be annotated afterward. The developers of IkeWiki has extended IkeWiki into KiWI [Sc09].

The SweetWiki presented by Buffa et al. in [Bu08] is another example of a text-centered semantic wiki and is implemented in Java. It focuses on the combination of social tagging with formal ontologies. Numerous editing tools are proposed by this project, such as a lite ontology editor, a “what you see is what you get” (WYSIWYG) editor, and an auto-completion system.

4.3 Logic-Centered Semantic Wikis

In the logic-centered approaches, semantic wikis are used as a form of online ontology editors. In this Section, we will discuss about logic-centered wikis.

OntoWiki [ADR06] is a logic-centered semantic wiki, and the classical textual content is not in the foreground of this system. It proposes efficient forms and integrates RDF triples directly into the text to collaboratively create and maintain ontologies. Furthermore, it

uses interesting functionalities such as faceted navigation and the integration of Web services (iCal, GoogleMap). It supports semantic search and navigation, as well as the possibility to version metadata.

myOntology [SH07] is another logic-centric semantic wiki which aims at exploiting the collective intelligence of a community for ontology engineering and uses horizontal ontology management approach. This system combines simplicity of wikis with intuitive visualization techniques.

Tobias Kuhn presented AceWiki in [Ku08a] which is a semantic wiki and uses ACE as CNL. Since ACE is a subset of English, contents of AceWiki looks completely natural. Instead of using technical terms (e.g., "ontological element", "subclass", or "property"), AceWiki interface uses terms like "word", "hierarchy", or "transitive verb", which should be more familiar to the end users with no background in ontology engineering. Five types of words are supported in AceWiki: proper names are interpreted as individuals; nouns are interpreted as classes; transitive verbs, of-constructs, and transitive adjectives are interpreted as binary relations. A predictive editor is integrated with the AceWiki to assist the users to create ACE sentences. The predictive editor shows all the possible words at a given position in the sentence. OWL reasoner Pellet[3] is used in AceWiki and every new sentence is checked for consistency. If the new sentence is not consistent with the ontology, then it is displayed in red font and excluded from reasoning.

[3]http://pellet.owldl.com/

Part III

Thesis Contribution

Chapter 5

Tool Survey

In Chapter 3, we discussed in detail how NLIs can be used to query and add data to knowledge bases. We also discussed about semantic wikis in Chapter 4 which can be used to hide the underlying complexity of formal representation of ontologies and can also make ontology engineering significantly simplified. This Chapter will focus on selecting the appropriate tools and technologies for our study. QA tools accept NL query as input and generate SPARQL query to fetch the result from the underlying KB. So, our first approach is to study these QA tools and try to find out a way to generate SPARQL query for updating and adding data to the KB, more specifically, analyze how these tools create the Select query and adapt that approach to generate Update query. Another possible approach is to work with CNL. CNL has a grammar which makes it suitable to work in combination with Semantic Web ontologies. To chose the best fitting tool for our purpose, a comparison between existing systems will be presented in Section 5.1. Finally, Section 5.2 will discuss about the reasons behind our chosen tools and technologies.

5.1 Comparison Between Identified Tools and Technologies

We focused on the tools that are open source. The summary of the comparison is shown in Table 5.1. Based on the requirements presented in Section 1.4, we have compared the tools.

Table 5.1: Comparison between existing NLIs to KBs based on different features related to our study

	Approach	User guidance	Domain independence	OWL → NL conversion	NL → OWL conversion	Adding data	Updating data	Search	Open source	Automatic ambiguity resolution	Web-based	S/w Architecture	Tool type	Extension
AquaLog	QA	-	+/-	-	-	-	-	+	+	+/-	+	Client/Server	Website	API
NLP-Reduce	QA	-	-	-	-	-	-	+	+	+/-	-	Standalone	Desktop Application	-
AutoSPARQL	QA	-	-	-	-	-	-	+	+	+/-	+	Client/Server	Website	-
FREyA	QA	+/-	+	-	-	-	-	+	+	+/-	+	Client/Server	Website	-
ROO	CNL		+	-	-	+	+	-	+	+	-	Standalone	Protégé plugin	Protégé plugin
ACE	CNL	+	+	+	+	+	+	+	+	+	+	Client/Server	Website, Web-service	Web-service

Caption: + supported, +/- partly supported, - not supported.

Results of the comparison will be discussed in the following Sections; Section 5.1.1 will discuss about the QA systems, and Section 5.1.2 will discuss about the CNL tools.

5.1.1 Discussion on QA Systems

Adding and updating data of the KB are important requirements for our study, but QA systems do not possess the ability to do so as seen from Table 5.1. One possible solution could be to analyze how QA systems generate the Select query and use that approach to convert NL to SPARQL Update query. However, the results presented in the corresponding publications of the QA tools are not promising. We collected the results in Section 3.1, which show that the SPARQL Select queries generated in QA systems are far below than 100% accuracy. One of the main reasons is that the user can write free text in NL, and it is not possible to accurately process the input in all cases. So, we will argue that converting NL to "SPARQL Update query" will not be accurate in all cases and will add unwanted data in the KB.

Table 5.1 also shows that, only FREyA is entirely domain independent, and it has partial user guidance. However, FREyA does not guide the user to formulate a correct sentence at the first place. Above mentioned reasons suggest that QA systems are not the strong candidate for our study.

5.1.2 Discussion on CNLs

As shown in Table 5.1, CNLs can automatically resolve the ambiguity. CNLs have formal syntax and semantics, and they can be mapped unambiguously to OWL. Moreover, CNL tools facilitate adding and updating data, which was not possible using QA systems.

However, there is a learning curve added by the grammar of the CNL. The predictive editor provided by ACE can solve this problem to a degree. Above mentioned features suggest that CNL tools are better candidates than QA systems for our study.

5.2 Reasons Behind Selecting ACE and AceWiki

After analyzing QA systems and CNL tools, we figured out that ACE is the best choice for our study. We have also decided to use AceWiki as a Semantic Wiki.

- ACE: Reasons behind choosing ACE are listed below:
  - Round-trip: We can create OWL ontology using ACE, and we can also verbalize OWL ontology into ACE sentences.
  - User guidance: The user can be guided to create a new ACE sentence by using the predictive editor. It decreases the learning curve of ACE grammar.
  - Web-services: ACE provides web-services for ACE parser (APE) and OWL-verbalizer. These web-services can be used in other external systems.
- AceWiki: Reasons behind choosing AceWiki are listed below:
  - AceWiki uses ACE as the CNL, and we have already decided to use ACE in our prototype. There are other Wikis which use CNL. WikiOnt-CNL [Sm09] supports Rabbit and ACE for verbalizing OWL axioms. However, it does not allow to create or edit the CNL sentences directly. Moreover, it does not support reasoning. Moreno and Bringert [MB08] uses a multi-lingual CNL framework in a wiki environment. Again, reasoning is not enabled here. On the other hand, AceWiki uses ACE as CNL and integrates reasoning over the content of the Wiki. It also uses a predictive editor to help the user in writing ACE sentences.
  - Search: One can search data in AceWiki by writing questions in ACE sentences and can also search concepts by keywords.

Chapter 6

Conceptual Design

This Chapter contains detail discussion about the conceptual design of the prototype. A mockup driven development approach [ZC03] is followed to understand how the tool should act. Developing a web application can be complicated if there is a lack of an appropriate application model, architecture, and framework. A good implementation model facilitates the application developers to speed up the development and to communicate with clients effectively [ZC03]. This chapter is organized as follows: Section 6.1 will focus on the main requirements, Section 6.2 will present the UML use case diagram, the mockups will be presented in Section 6.3, and finally, a proposed workflow will be shown in Section 6.4.

6.1 Main Requirements of the Prototype

To begin with, let us summarize the main requirements of this prototype. These requirements are derived from the research questions formulated in Section 1.4, but described in more detail:

1. **Providing web-based NLI:** As discussed in Section 5.2, CNL has been selected instead of entirely natural language since former one is more robust and less error prone than the later one.
   Reasons behind choosing a web-based interface have some advantages: users do not need to install any software and need not to bother with security issues of the tool since security engineers will test the tool in the server and everyone will be able to access the tool through their browsers.
2. **Converting existing OWL ontologies into NL representation:** The prototype should be able to convert existing OWL ontologies into NL for reuse.

3. **Adding data to OWL ontology through web-based NLI:** The NLI should allow the user to create a new ontology or add data to existing one.
4. **Updating OWL ontology using web-based NLI:** The NLI should provide support to update existing OWL ontology.
5. **Search support in OWL ontology through NLI:** User should be able to search in the OWL ontology through NLI.
6. **Providing guided user interface:** Prototype should guide the user to add new knowledge, update knowledge and query into the knowledge base.
7. **Creating domain specific lexicons automatically:** There should be a way to create the domain specific lexicons automatically from existing ontologies.
8. **Exporting the content to OWL format:** System should be able to export data in OWL format, so that changes made by domain experts inside the prototype can be easily reused in some ontology editing software, e.g., Protégé.

6.2 UML Use Case Diagram

The UML use case diagram of the prototype is shown in Figure 6.1. It shows the expected functional behavior of the system.

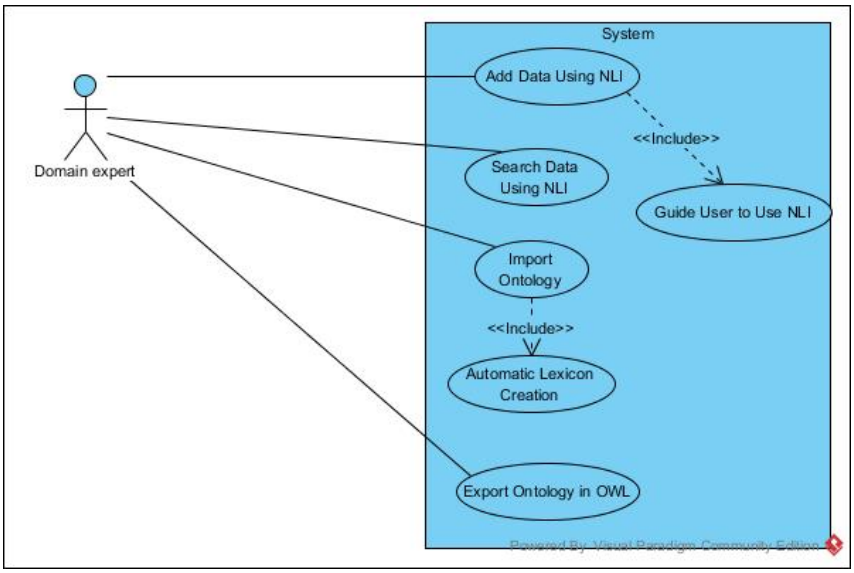


Figure 6.1: UML Use case diagram showing the expected functional behavior of the prototype.

6.3 Mockups

Several mockups were created to realize the system regarding workflows and functionalities. However, the AceWiki's user interfaces are not reflected in the mockups and we kept the mockups as simple as possible. The mockups also helped to analyze the system in more detail. These mockups were developed at the beginning of the design phase, which continuously evolved during the implementation of the prototype.

Initially, the idea was to provide a textbox in which the user could write a natural language sentence to add data into the knowledge base, as shown in Figure 6.2.

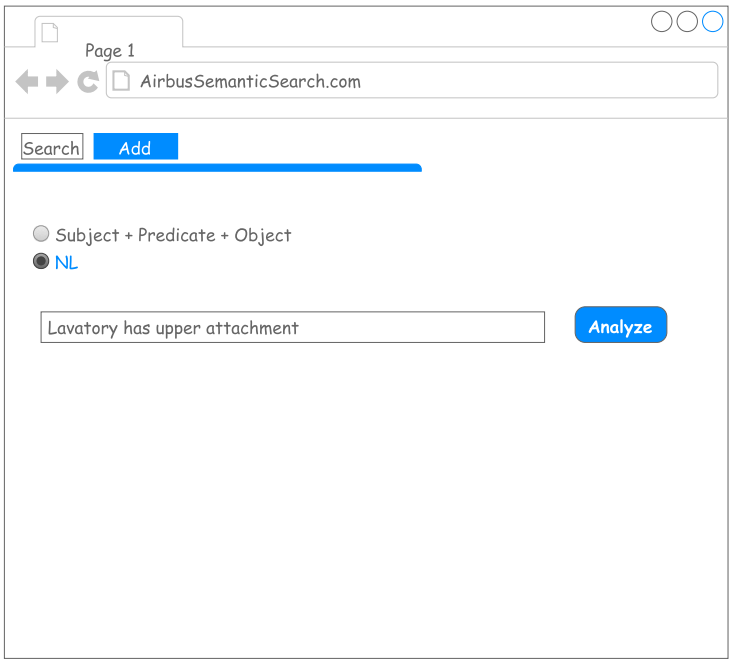


Figure 6.2: Mockup showing that user can write NL to add data into the knowledge base. NLP will be used to parse this input

Then the sentence will be analyzed to identify each part of the triple, i.e. the subject, predicate and object. Next, each part will be analyzed to match against the underlying lexicon. After the matching, each part will have any of the following three states: the word is completely matched with an existing concept (marked as green), or the system found some ambiguity while matching (marked as yellow), or the word is not yet defined in the system (marked as red). This scenario is depicted in Figure 6.3. Subject (*Lavatory*, in this example) is marked as green, which means that the system has perfectly matched this word with the existing lexicon.

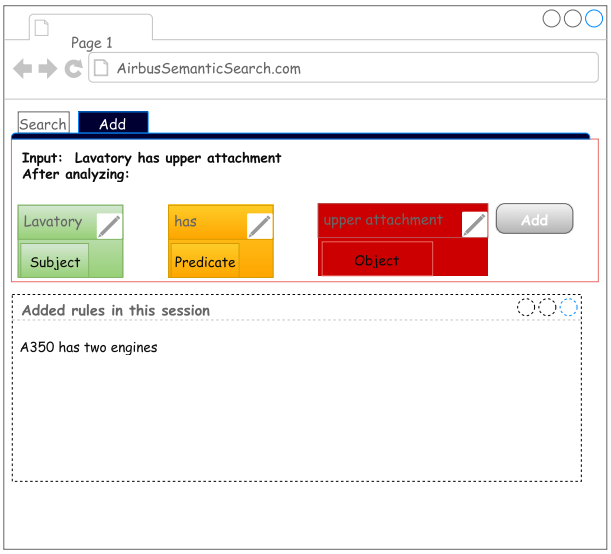


Figure 6.3: Mockup showing the identification of subject, predicate, and object from NL input

Yellow marking on the predicate (*has*, in this example) means that the system found ambiguity while matching this word. Hence, suggestions will be provided from the existing lexicons to resolve the ambiguity. Additionally, the option to create a new concept will be given as well. Figure 6.4 and 6.5 show a suggestion list and the option to add new concept respectively.

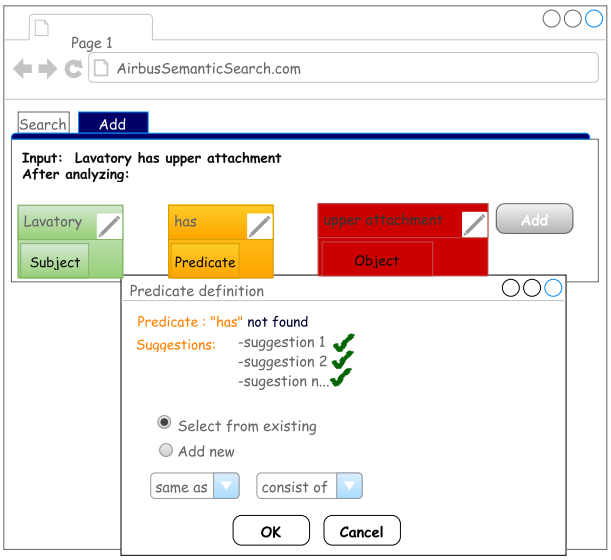


Figure 6.4: Mockup showing the ambiguity resolution by engaging the user. As an option to resolve the ambiguity, a list of suggestion will be provided from existing lexicons to choose from

Figure 6.5 and 6.6 depict the situation when the user creates a new definition. Depending

on the type of the word, different options will be provided to set the properties of this new definition, e.g., domain and range properties will be available for the predicate, on the other hand, subclassOf and sameAs properties will be available for an object. However, the above-mentioned property list is only a sample list; more properties can be added depending on the type of the word in the implementation.

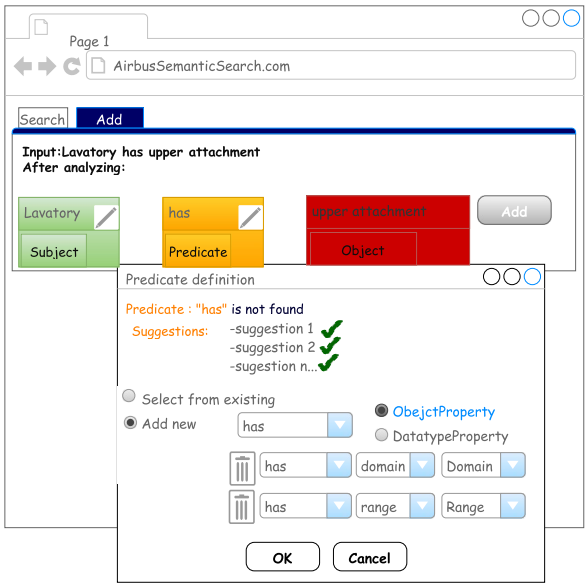


Figure 6.5: Define new concept for a predicate. User can set various properties of the predicate

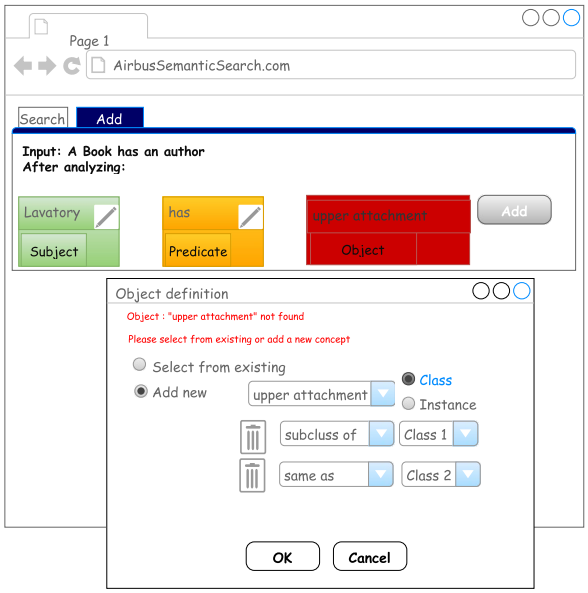


Figure 6.6: Define new concept for an object. User can set various properties of the object

Finally, when each word from the triple is defined in the system, the sentence will be

available to be added to the knowledge base as showed in Figure 6.7.

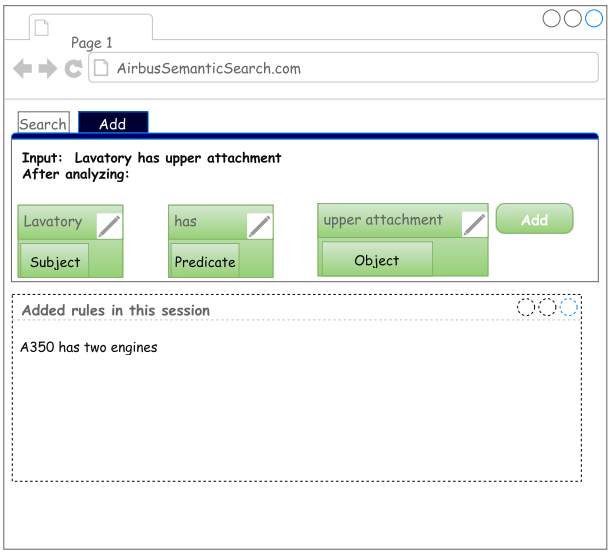


Figure 6.7: All the concepts are defined or matched with existing lexicon. Therefore, everything has become green and user can add this sentence in the KB

Because of the error-proneness nature of NLP, users having knowledge of triple format may want to input the triple by themselves. This situation is realized by the mockup shown in Figure 6.8. Drop-down list and auto-completion are provided to choose from existing concepts.

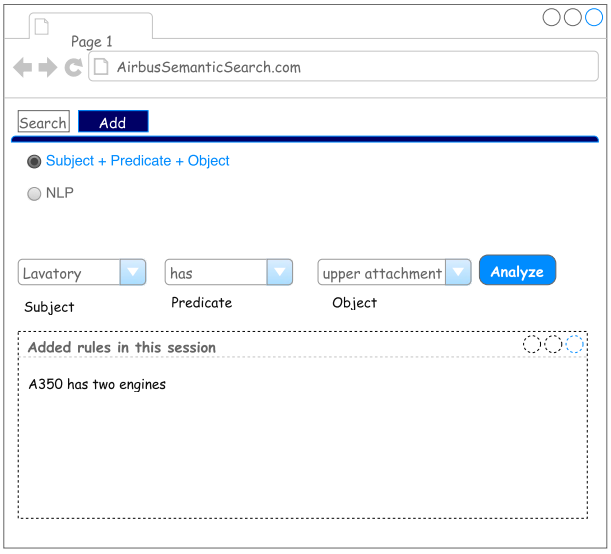


Figure 6.8: Mockup showing that the user can directly input the triple format

6.4 Proposed Workflow

In this Section, an informal top level architecture of the conceptual system will be presented, then a workflow will be introduced to add new knowledge to the knowledge base.

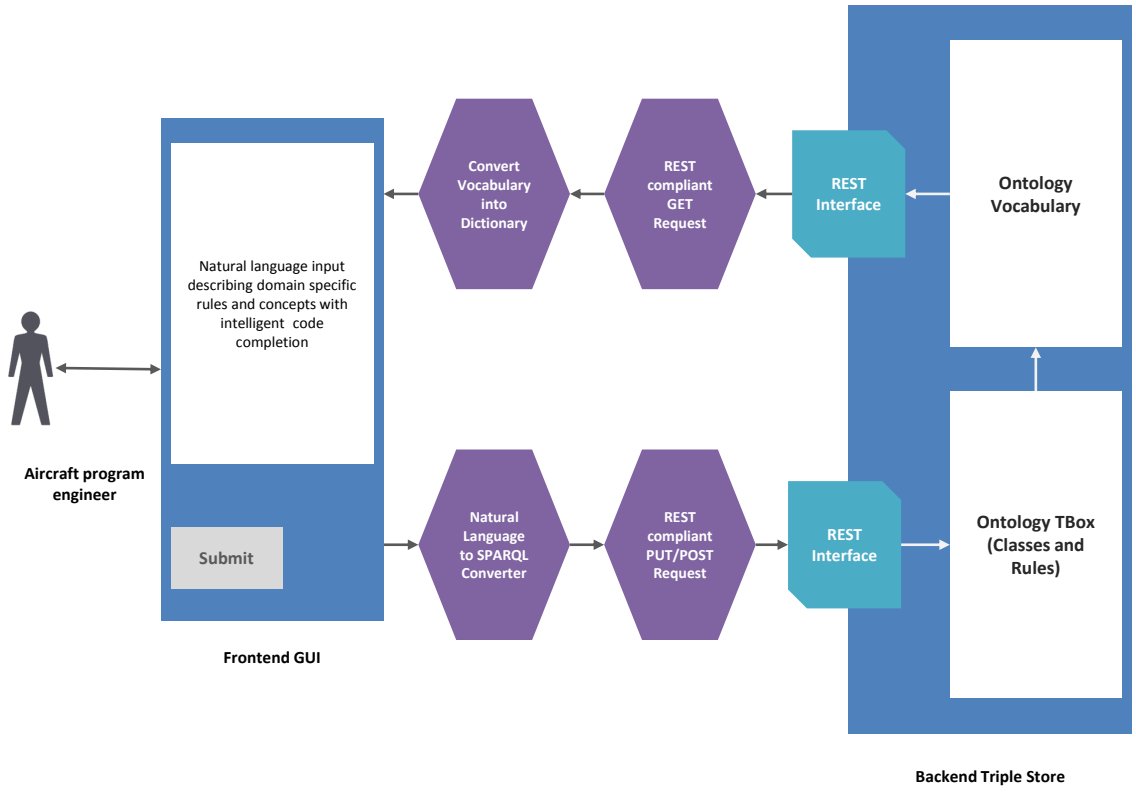


Figure 6.9: An informal top level architecture of the proposed system (This image originates from Airbus research group)

The informal architecture presented in Figure 6.9 shows several important components, e.g., NL to SPARQL converter, REST interfaces and GUI for user input.

Proposed workflow is shown in Figure 6.10. At first, the user will write new knowledge in the natural language. Then the system will check whether the entered concepts are already defined in the system or not. If any of the concepts is found undefined in the system, then the user will be involved to define it properly. The new knowledge will be added into the knowledge base as soon as all the concepts are properly defined.

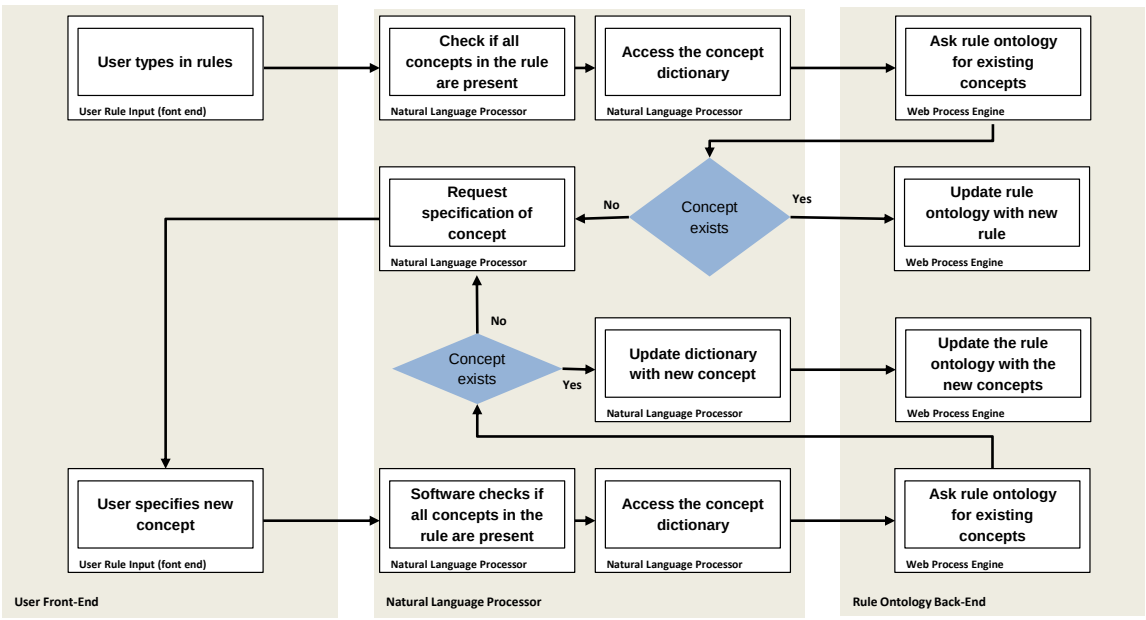


Figure 6.10: Proposed workflow showing how new concept and knowledge can be added to the knowledge base (This image originates from Airbus research group)

Chapter 7

System Design and Architecture

This Chapter presents the design and architecture of the final system. In Section 7.1, at first, the current architecture of the AceWiki is described to have a better understanding of our working environment. After that, evaluation of the AceWiki regarding the requirements of this study will be discussed. The evaluation shows that there are numerous obstacles with AceWiki, which need to be taken care of. In addition, the limitations of AceWiki lead us to add new features on top of AceWiki which will be presented in Section 7.2. Finally, we will go through the design and the architecture of our prototypical system step by step in Section 7.3.

7.1 Evaluation of AceWiki

Reasons behind choosing AceWiki for this thesis as well as the features and purpose of Acewiki have already been discussed in detail in Section 5.2. In this Chapter, the architecture of AceWiki will be discussed briefly. Then, the AceWiki will be evaluated against our dataset to find out to what extent it supports our requirements. Finally, the limitations of the AceWiki and its associated components will be listed.

Figure 7.1 shows the top-level architecture of AceWiki. Different third-party libraries and systems are being used by it. Echo framework is used as a web framework to implement this semantic wiki. AceWiki uses APE to validate whether the input sentence is a valid ACE sentence or not. APE is also used to find out which ACE sentences can be converted to OWL. The sentences which are not convertible to OWL, are not available to participate in reasoning. Those sentences are marked by red triangles in AceWiki.

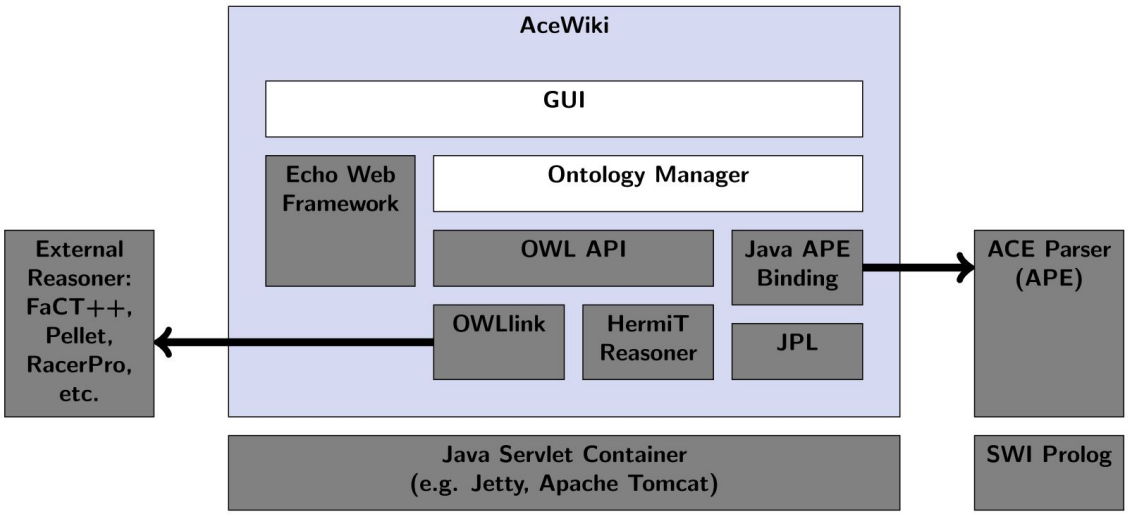


Figure 7.1: Architecture of AceWiki [Ku10b]

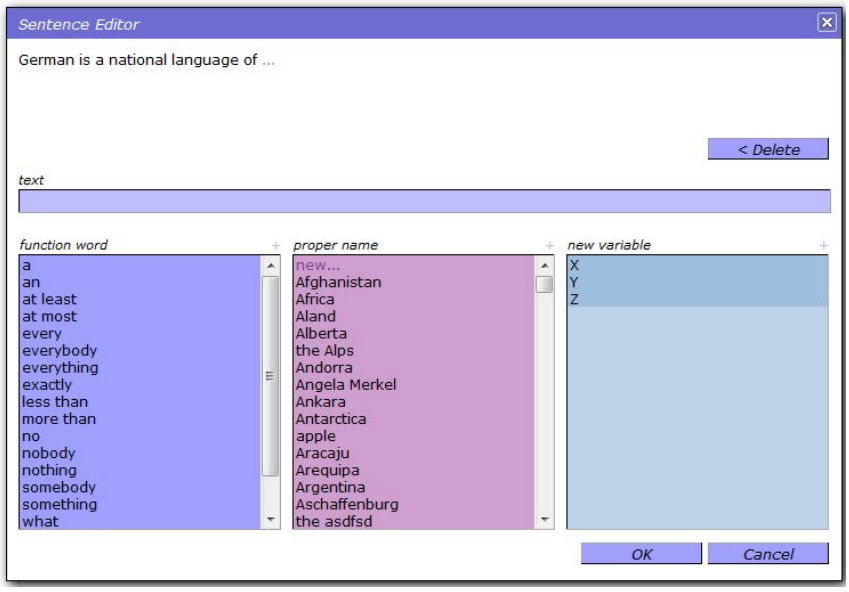
7.1.1 Support for Requirements

This Section will start with the discussion about the features of AceWiki which are aligned with our requirements. Section 7.1.2 will discuss the shortcomings of AceWiki, which needs to be tackled to fulfill our needs.

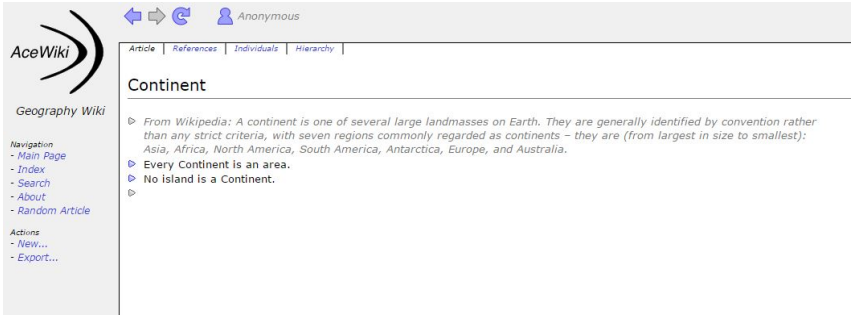
The Acewiki possesses some valuable features regarding our requirements. It provides the feature of creating ontology using an NLI. It also possesses the functionality of updating the existing knowledge through the NLI. Moreover, to guide the user to create a new ACE sentence, it provides a predictive editor which is shown in Figure 7.2a. The editor shows all the possible words to continue a sentence from a current position.

The layout of each article is shown in Figure 7.2b. Each article can contain several ACE sentences.

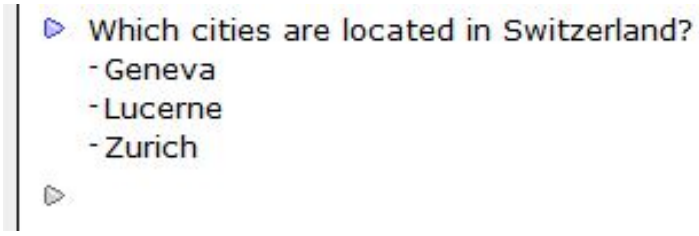
AceWiki provides two types of search possibilities. One can write questions in ACE sentences, and AceWiki can answer to that query as shown in Figure 7.2c. However, AceWiki can answer only about instances of classes. Moreover, there is another search possibility. One can search the classes by keywords from the search box.



(a) A screenshot of the predictive editor of AceWiki. This editor shows all possibilities to continue the sentence from current position. This screenshot is originated from [Ku08b]



(b) A screenshot of the web interface of AceWiki showing the wiki article for the class "Continent". This screenshot is originated from [Ku08b]



(c) A screenshot of AceWiki showing the functionality of answering questions based on the KB. This question-answering system facilitates searching. This screenshot is originated from [Ku08b]

Figure 7.2: Screenshots of AceWiki showing related features for our use cases. These screenshots are originated from [Ku08b]

Table 7.1 discusses the requirements of our study and to what extend AceWiki fulfills them.

Table 7.1: Evaluation of AceWiki regarding our requirements

Requirement	Supports	Comment
Providing web-based NLI	Yes	
Adding data to OWL ontology through a web-based NLI	Partial	Owl-verbalizer can not convert all OWL segments to ACE sentences. Furthermore, AceWiki cannot handle all valid ACE sentences, because it uses a subset of ACE.
Updating OWL ontology using a web-based NLI	Partial	Since all ACE sentences can not be entered into AceWiki and owl-verbalizer can not work with all OWL constructs.
Importing existing OWL ontology	No	
Search support in OWL ontology through the NLI	Yes	
Guided user interface	Yes	Using predictive editor.
Creating domain specific lexicon automatically	No	
Exporting in OWL format	Yes	
Exporting in Turtle format	No	

7.1.2 Limitations of AceWiki

The Acewiki is tested against real world dataset containing cabin ontology provided by Airbus. The result shows that several features need to be implemented on top of AceWiki to be compatible with our dataset.

Limitations of the AceWiki are discussed here:

1. No import functionality: AceWiki has no option to import existing ontologies, which is a barrier to utilizing existing ontologies.
2. Can not work with floating point numbers: Floating point numbers occurred several times in our real world dataset to define the position of various components. But, AceWiki can not handle floating point numbers.

3. Can not handle more than two classes in a DisjointClasses block: If there are more than two classes in a DisjointClasses block of OWL ontology, then OWL-verbalizer could not handle it.

4. Labels and comments from OWL ontology are lost: AceWiki has no way to save labels and comments from OWL.

5. Wrong URI: If there is an import statement in OWL ontology, then URI for the imported classes are not the same as the base URI of initial ontology, but AceWiki has no way to define different URI for those imported classes.

6. All ACE sentences are not supported: OWL-verbalizer has been used to translate owl ontology into ACE sentences. Since AceWiki implemented a subset of ACE [Ku10a], so all the ACE sentences which were translated by Owl-verbalizer were not compatible with AceWiki. Moreover, OWL-verbalizer itself can not handle some OWL properties. Limitation of OWL-verbalizer will be discussed in Section 7.1.3

   Now we are going to list some of the ACE sentences which are not supported by AceWiki. Owl-verbalizer tags every word while verbalizing to ACE. Before listing the unsupported sentences, we list the meaning of the tags first:

   - f = function word
   - qs = quoted string
   - comment = comment
   - cn_sg = singular common noun
   - tv_sg = singular transitive verb
   - tv_pl = plural transitive verb
   - tv_vbg = past participle verb
   - pn_sg = singular proper noun

   Now we will list and discuss some ACE sentences which AceWiki can not handle. At first, a sentence is shown in Table 7.2, which is supported by AceWiki.

Table 7.2: Example 1, Supported ACE sentence in AceWiki. Owl-verbalizer tokenized these sentences from OWL

Token type	Token
f	Every
cn_sg	Lavatory-A
f	is
f	a
f	LaterralLavatory.

Corresponding OWL:

```
<Declaration>
  <Class IRI="/LateralLavatory"/>
</Declaration>
<Declaration>
  <Class IRI="/Lavatory-A"/>
</Declaration>
<SubClassOf>
  <Class IRI="/Lavatory-A"/>
  <Class IRI="/LateralLavatory"/>
</SubClassOf>
```

AceWiki can not handle floating point number as shown in Table 7.3.

Table 7.3: Example 2, Unsupported ACE sentences in AceWiki. From the red portion, it is not possible to write the sentence in AceWiki since AceWiki does not support floating point number.

Token type	Token
f	Every
cn_sg	Lavatory-A
f	is
f	a
f	thing
f	that
tv_pl	has depth
f	53.0
f	and
f	that
tv_pl	has width
f	41.0
f	.

Corresponding OWL:

```
<SubClassOf>
 <Class IRI="/Lavatory-A"/>
 <ObjectIntersectionOf>
  <DataHasValue>
   <DataProperty abbreviatedIRI="equipments2monuments:has depth"/>
   <Literal datatypeIRI="&xsd;float">53.0</Literal>
  </DataHasValue>
  <DataHasValue>
   <DataProperty abbreviatedIRI="equipments2monuments:has width"/>
   <Literal datatypeIRI="&xsd;float">41.0</Literal>
  </DataHasValue>
 </ObjectIntersectionOf>
</SubClassOf>
```

Conditional sentences are not supported by AceWiki even though those are supported by ACE. Table 7.4 shows one example of conditional ACE sentence.

Table 7.4: Example 3, Unsupported ACE sentences in AceWiki. Conditional sentence is not supported in AceWiki

Token type	Token
f	If
f	X
tv_sg	has attached
f	Y
f	then
f	Y
tv_sg	is attached to
f	X
f	.

Corresponding OWL:

```
<InverseObjectProperties>
 <ObjectProperty abbreviatedIRI="equipments2monuments:has attached"/>
 <ObjectProperty IRI="/is attached to"/>
</InverseObjectProperties>
```

Table 7.5: Example 4, Another unsupported ACE sentences in AceWiki which contains floating point number

Token type	Token
f	Every
cn_sg	ChilledGalley
tv_pl	has depth
f	39.0
f	.

Corresponding OWL:

```
<SubClassOf>
 <Class IRI="/ChilledGalley"/>
 <DataHasValue>
  <DataProperty abbreviatedIRI="equipments2monuments:has depth"/>
  <Literal datatypeIRI="&xsd;float">39.0</Literal>
 </DataHasValue>
</SubClassOf>
```

7.1.3 Limitations of Owl-Verbalizer

Owl-verbalizer, which is a part of the ACE project, is not compatible with all OWL axioms. For this reason, some of the OWL axioms could not be converted to ACE sentence.

Here we list the owl properties which OWL-verbalizer can not handle:

- SubDataPropertyOf
- FunctionalDataProperty
- DataPropertyRange
- DLSafeRule
- DatatypeDefinition
- ObjectIntersectionOf
- DataAllValuesFrom
- DataOneOf
- DataExactCardinality

- EquivalentClasses
- Annotation

7.2 Extension for Our Use Cases

After doing the analysis in Section 7.1, we pointed out which functionalities we need to add on top of AceWiki to fulfill our requirements. This Section will discuss the extensions we planned to implement, and the system design will be discussed in Section 7.3.

7.2.1 Features to be Implemented on AceWiki

After finding out the limitations of AceWiki, we will discuss what features we have to implement on top of AceWiki.

1. **Import functionality:** To reuse existing ontologies, we have decided to add import functionality in AceWiki.
2. **Auto lexicon creation from OWL ontology:** So that domain experts do not need to input the lexicons manually.
3. **Change grammar to accept floating point number:** We have to update the grammar of AceWiki to provide support for floating point numbers.
4. **Rewrite DisjointClasses blocks:** If there are more than two classes in a DisjointClasses block of OWL, then OWL-verbalizer can not handle it. So, if there are more than two classes in a DisjointClasses block, then we have to create n(n-1)/2 blocks, where n is the number of classes in each block.
5. **Store rdfs:Labels and export them:** AceWiki has no feature to store the data of rdfs:Labels. So we have to make sure that rdfs:Labels are not missing inside AceWiki and rdfs:Labels are also correctly exported when we export AceWiki in OWL format.
6. **Store rdfs:comments and export them:** Just like rdfs:Labels, AceWiki can not handle rdfs:comments also. So, we have to handle this case also.
7. **Store URI:** In AceWiki, we can only save a lexicon but not the whole URI. We have to add functionality to store the URI also.
8. **Turtle to OWL conversion:** Inside Airbus, turtle format is used to represent ontologies. However, the OWL-verbalizer can only work with the OWL/XML

format. So, files have to be converted from turtle to OWL/XML format before sending them to the OWL-verbalizer.

9. **Export in Turtle format:** AceWiki can export ontology in OWL/XML format, but the Restful services deployed in Airbus work with turtle format. So, we need to add support to export data in turtle format also.

7.3 System Design

This Section explains the overall system design and architecture of the final prototype with various UML diagrams. Different components of the system are also described.

Many considerations were taken into account while building the architecture. At first, we tried to convert the OWL ontology into ACE sentences using the provided owl-verbalizer, but it has its limitations which caused data loss. A second attempt was made to develop the whole system with our XML parser, but it would be very complicated and also very time consuming. Consequently, a more appropriate solution resulted in a combination of the owl-verbalizer and our XML-parser. Our XML parser re-factors the OWL ontology to be more compatible with the OWL-verbalizer and on the other hand, it provides the opportunity to tackle the OWL axioms which are not supported by the OWL-verbalizer. This architecture ensures more data integrity since we are handling some OWL axioms by ourselves which were not possible by the OWL-verbalizer. Moreover, the re-factored ontology becomes more compatible with OWL-verbalizer.

Data Flow Diagram

The purpose of Data Flow Diagram is to represent the exchange of information. However, it is not used to answer in what way and in what order the information is being used in a system. The Data Flow Diagram shown in Figure 7.3 contains four processes, one external entity, and one data store. We have put the processes in the middle and data store on the right side and external entities on the left side. Based on Diagram 7.3, we see that the *Import OWL* process receives the uploaded owl from the *Upload OWL* process. This process also communicates with the *Refactor OWL* process to refactor the uploaded owl file, because the *OWL verbalizer* can not handle all OWL axioms, which has already been discussed in Section 7.2.1. *Import OWL* process sends the *Refactored OWL* to *OWL verbalizer* and receives corresponding *ACE sentences* in response. This process also stores the *lexicon* and *valid ACE sentences* in the data store.

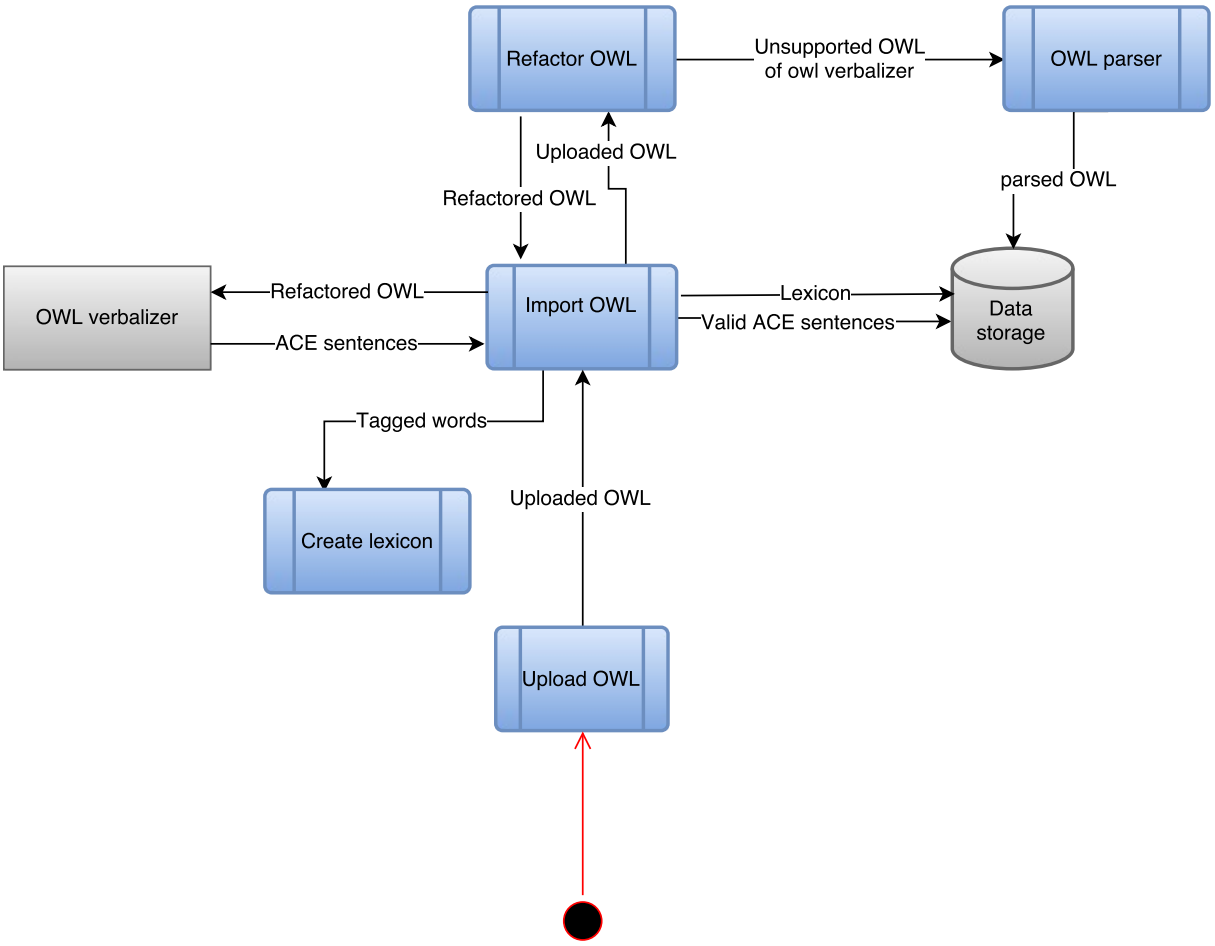


Figure 7.3: Level 1 data flow diagram for import functionality and lexicon creation. It represents the exchange of information among processes, data storage, and external entities. Blue parts show the newly added modules.

Package Diagram

A package diagram is presented in Figure 7.4, to show the dependency and structure of the subsystems in the implemented prototype. This package diagram is generated using object aid[1]. Since the AceWiki project contains many modules and some modules are not relevant to our use case, we decided to show the most important subsystems in our package diagram. This top level view will be helpful in future to quickly understand the project.

[1]http://www.objectaid.com/

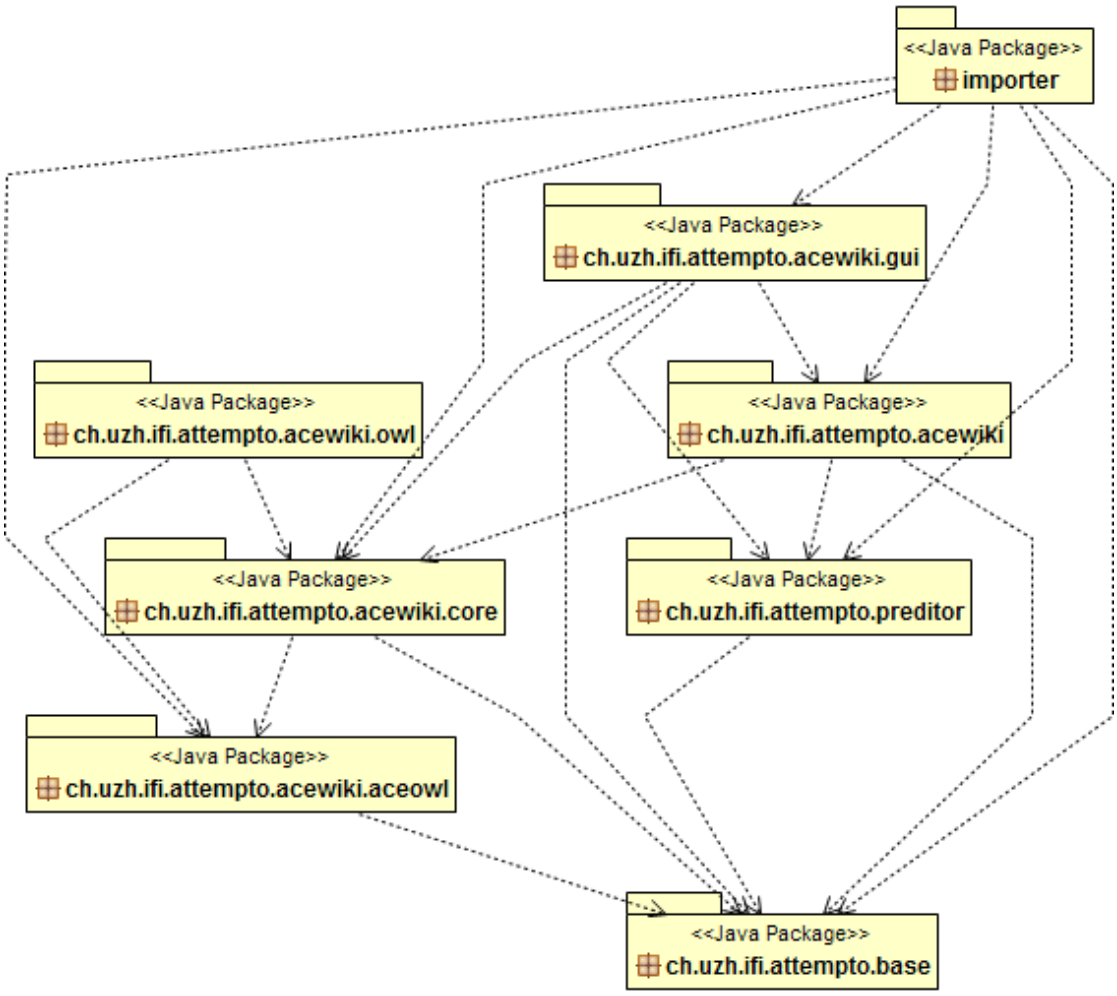


Figure 7.4: UML Package diagram of important packages of the application. Dashed arrow shows the dependency between the packages

Deployment Diagram

Execution architecture of our system, i.e. the assignment (deployment) of software artifacts to deployment targets is shown in the deployment diagram presented in Figure 7.5. Hardware devices or software execution environments are sometimes very important because changing the environment may make the system unusable or unstable. Our system is deployed on Windows 7 operating system and the execution environment is Java EE

container which is running Jetty version 9.0.0.

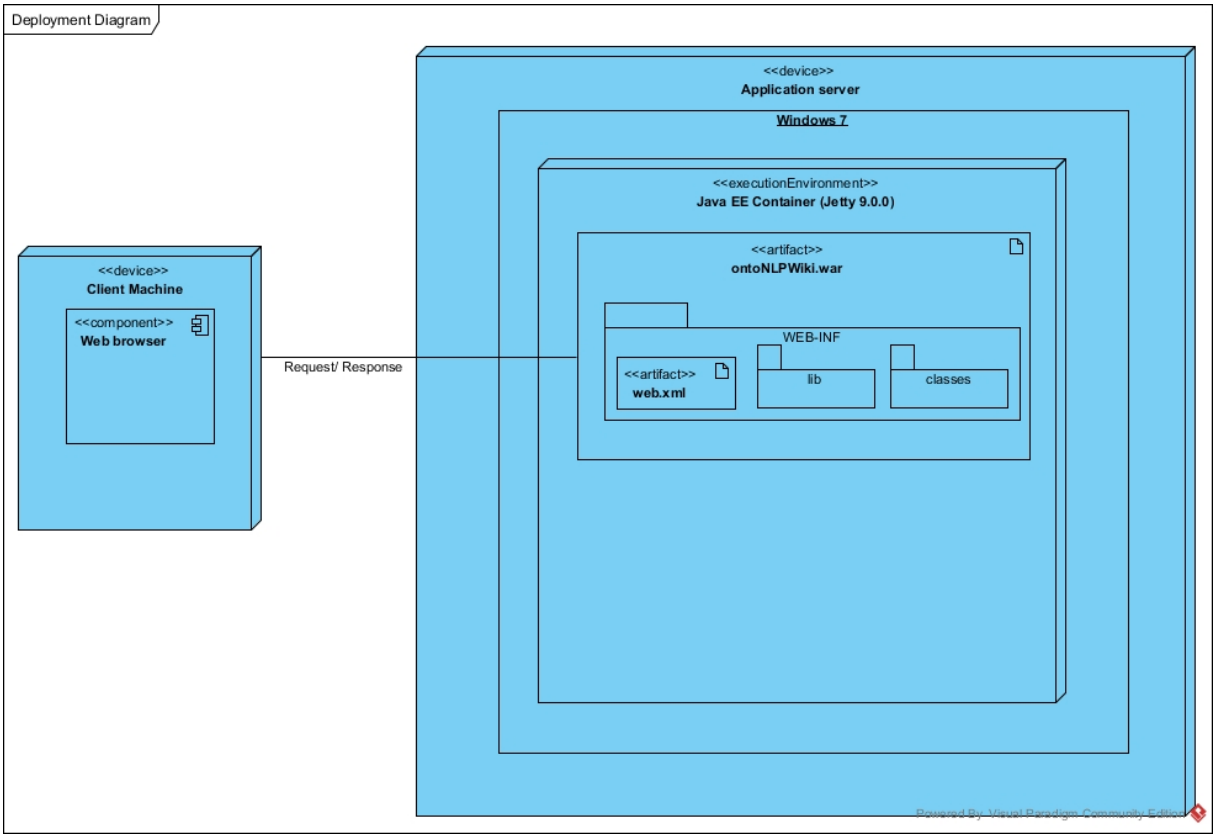


Figure 7.5: Deployment diagram of final system, which is using Windows 7 OS and Java EE container running Jetty version 9.0.0

Chapter 8

Prototype Implementation

The implemented prototype is a web-based application that utilizes the Java Servlet technology. The NLI is based on the AceWiki, which is a semantic wiki and uses ACE as a CNL. At first, the key technologies which are used to develop this prototype are listed in Section 8.1. Then, Section 8.2 will present a detailed and technical description of the prototype and its main functionalities.

8.1 Key Technologies

Here we present the list of the key technologies used to develop the whole system as well as a brief description for each of them:

1. **OWL API**: OWL API is a Java API for creating, manipulating and serializing OWL Ontologies and includes parser and writer for RDF/XML, OWL/XML, OWL Functional Syntax, Turtle and some other formats. [1].
2. **Echo Framework:** "Echo is an open-source framework for developing rich web applications. Echo applications can be created entirely in server-side Java code using a component-oriented and event-driven API." [2]
3. **AceWiki:** AceWiki [Ku08a] is a semantic wiki which is powerful and at the same time easy to use. Since it uses ACE as a CNL, the articles in the wiki look like natural English.
4. **Owl-verbalizer web service:** OWL verbalizer [3] is a tool that converts an OWL

[1] https://github.com/owlcs/owlapi
[2] http://echo.nextapp.com/site/
[3] https://code.google.com/archive/p/owlverbalizer/

ontology into ACE text. It was developed as part of ACE project[4]. We used the webservice of owl verbalizer.

5. **APE:** AceWiki requires to communicate with APE, so we used an HTTP interface to APE[5].
6. **Jetty runner:** AceWiki is a web application that has to be run on Java servlet. Jetty[6] is used as a servlet container.
7. **Jersey:** "Jersey is a framework for developing RESTful Web Services in Java that provides support for JAX-RS APIs."[7]
8. **XML:** XML is needed to design UI in Echo framework.
9. **JDOM XML parser:** JDOM[8] is used as a XML parser in our implementation.
10. **Git:** Git is used as a version controlling system.

8.2 Implemented Features on AceWiki

As explained in Chapter 7, we have taken AceWiki as our base and added features on top of it to meet our requirements. In this Section, we will discuss the features we developed as par our requirement. MVC architectural pattern is used for our implementation.

8.2.1 Enhancement of User Interface

In Figure 7.2, we have shown the interface of AceWiki. To give the UI a modern look and feel we have enhanced the interface. We focused on giving the look and feel which is used in other Airbus softwares, e.g., color scheme, layout design, etc. Different possible ways were investigated to upgrade the UI.

Vaadin[9] provides user interface components for web applications developed using Java. So, at first an attempt was taken to incorporate Vaadin into our prototype. During the first tests, it became apparent that it is not possible to integrate Vaadin into the Echo framework within the time frame of this thesis, since the Echo framework has no direct interface that can be used for the integration. The use of HTML and Cascading Style Sheets (CSS) was considered as an alternative but again, it was not possible to integrate

[4]http://attempto.ifi.uzh.ch/site/description/
[5]https://github.com/Attempto/APE
[6]http://www.eclipse.org/jetty/
[7]https://jersey.java.net/
[8]http://www.jdom.org/
[9]https://vaadin.com/home

CSS with Echo framework. Moreover, because of time constraint we could not replace the Echo framework since the whole AceWiki was developed using this framework. Then the last option is chosen, which is XML stylesheet. However, designing with XML is not as flexible as CSS, but we succeeded to give the UI a better look and feel which is compatible with Airbus softwares.

We present the new UI of the application in Figure 8.1

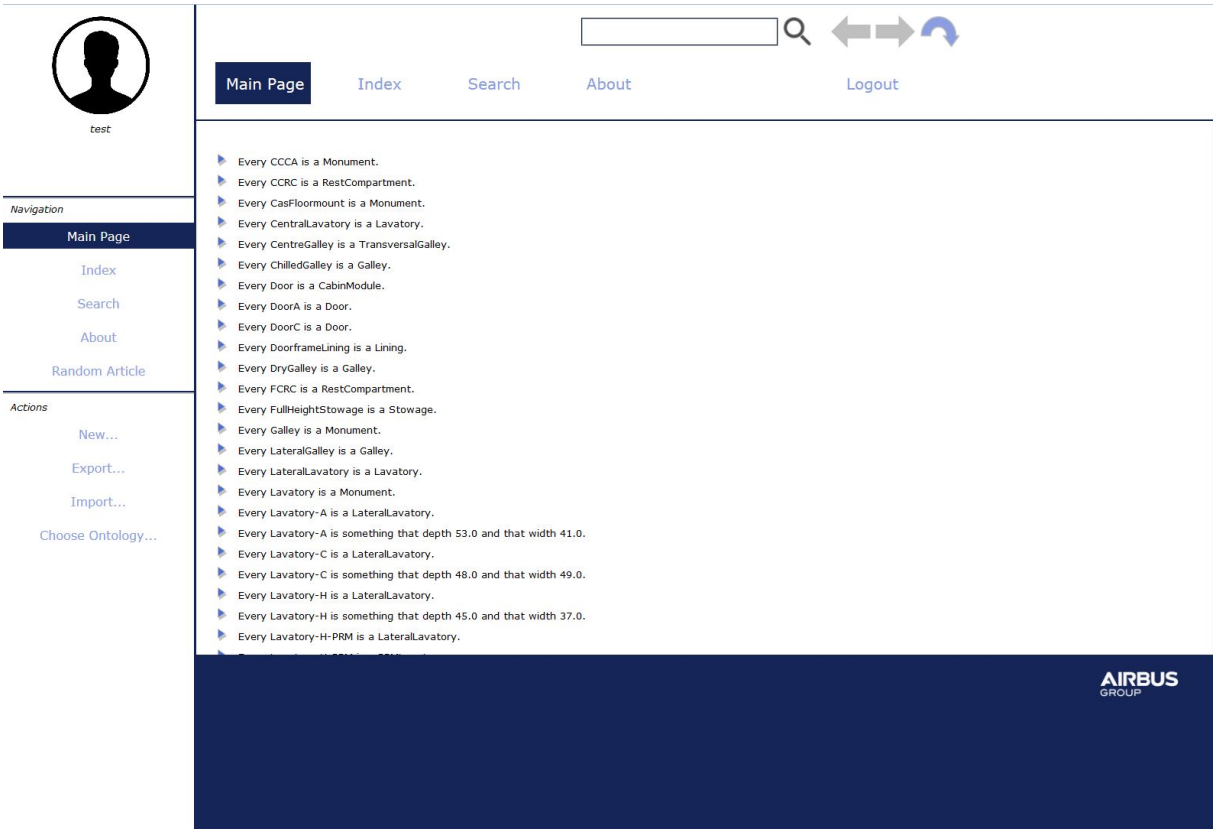
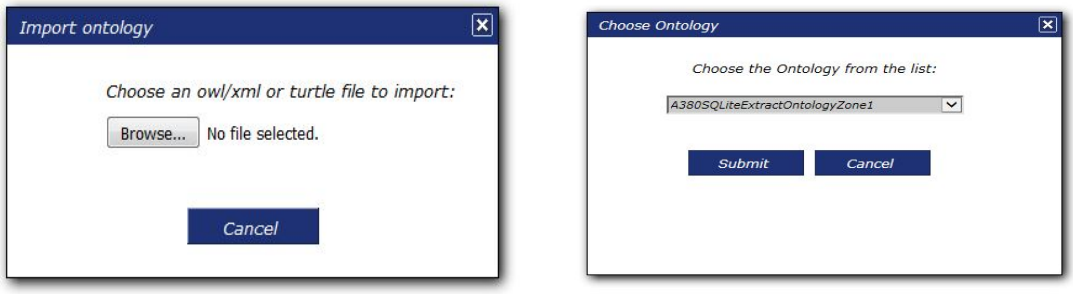


Figure 8.1: Enhanced user interface conforming the look and feel of Airbus softwares.

8.2.2 Import Functionality and Dynamic Lexicon Creation

We have provided two different options to import an ontology. The user can either upload an ontology from his disk as shown in Figure 8.2a, or can select from existing ontologies which are available via a web service, as shown in Figure 8.2b. There is a RESTful web service which provides the existing ontologies from the server of Airbus. A Rest client is developed using Jersy as part of our prototype to invoke that web service.



(a) User can upload an ontology to import it in the prototype

(b) User can choose an ontology from existing ontologies through a web service

Figure 8.2: Screenshots of different options to import an ontology

Refactoring OWL

If there are more than two classes in a DisjointClasses block of the OWL, then owl-verbalizer can not handle it. So, if there are more than two classes in a DisjointClasses block, then we have created n(n-1)/2 blocks, where n is the number of classes in each block. This is done by XML parsing.

Dynamic Lexicon Creation

At first, the lexicons need to be created to import an ontology. Automatic lexicon creation is done before importing the ontology. At first, the OWL-verbalizer web service is called which creates ACE sentences from OWL ontology and also tags each part of the sentence. Then using those tagged words, we create lexicons with surface forms and insert them into AceWiki. Finally, the wiki is filled with ACE sentences, and the import process is finished. Figure 8.3 shows the automatically created lexicons.

Conversion Between Ontology Formats

Web service from Airbus provides turtle files to represent their ontologies. On the other hand, OWL-verbalizer works with only OWL/XML. Moreover, the Acewiki also exports ontology as OWL/XML but web service from Airbus expects data in turtle format. For these reasons, we implemented a converter to convert OWL/XML to turtle and from turtle to OWL/XML.

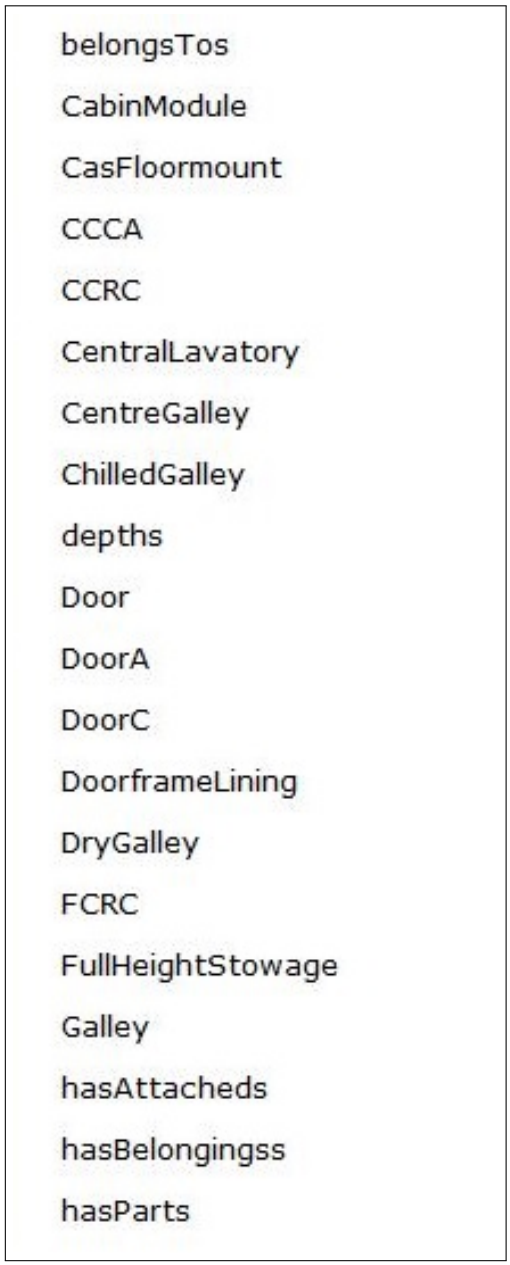


Figure 8.3: A screenshot of the list of lexicons which are created automatically while importing an ontology

8.2.3 Grammar Extension

AceWiki introduced *ACE Codeco grammar* that defines a subset of ACE in a formal and declarative way. Since this grammar defines only a subset of ACE, it was not possible to integrate some valid ACE sentences with AceWiki. For this reason, improvement of the

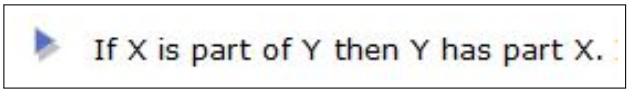
grammar of AceWiki was required.

For example, it was not possible to enter below sentences in AceWiki. The first example (8.1) shows a conditional sentence and the second one (8.2), shows floating point number.

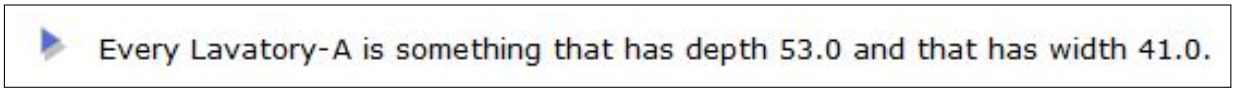
If X is part of Y then Y has part X. (8.1)

Every Lavatory-A is something that has depth 53.0 and that has width 41.0. (8.2)

After changing the grammar and updating the corresponding code base, we were able to import above sentences automatically as shown in Figure 8.4a and 8.4b, respectively.



(a) Our prototype supports conditional sentences



(b) Our prototype supports floating point numbers

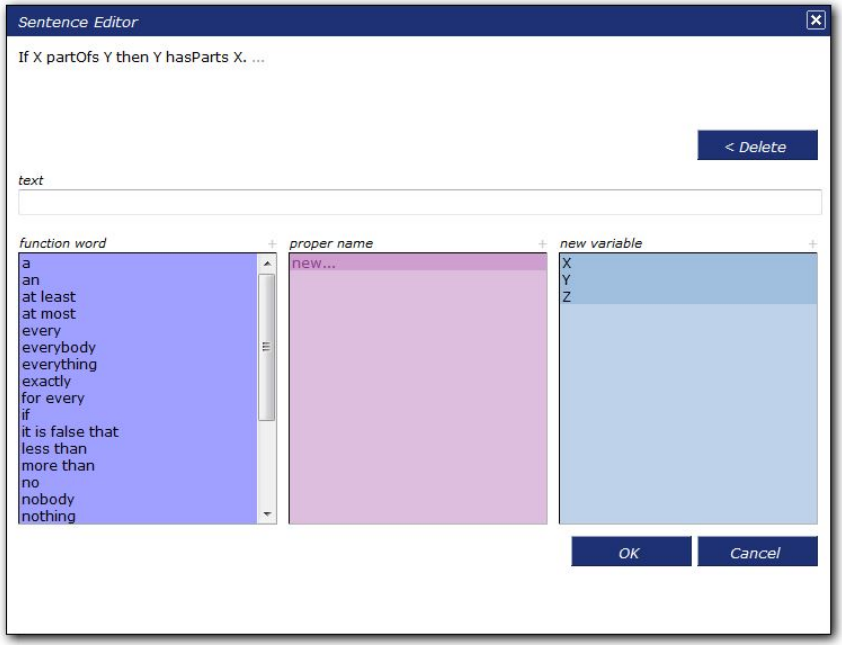
Figure 8.4: We have added support for different sentences which are not supported in AceWiki

8.2.4 Improvement of Predictive Editor

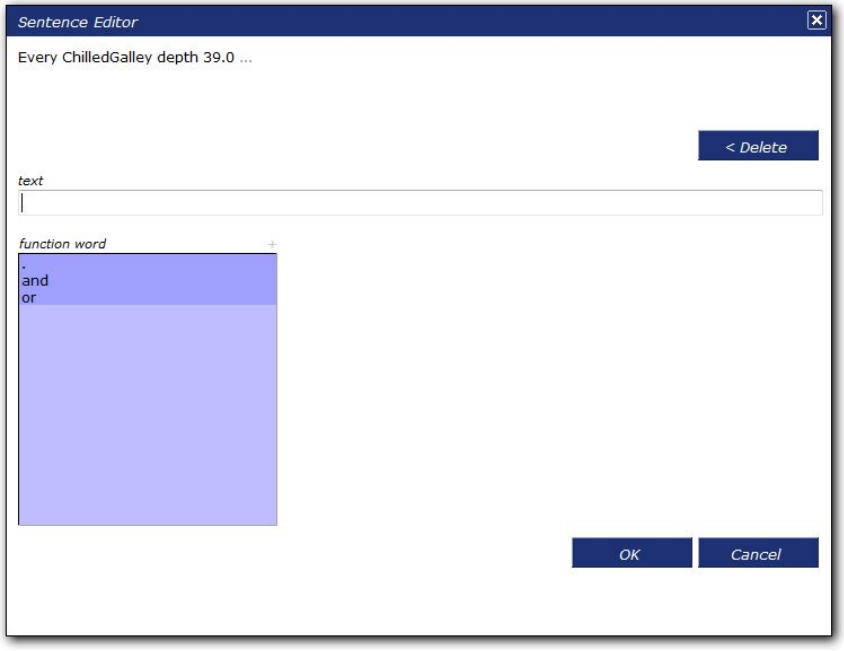
The predictive editor assists the domain expert to write ACE sentences, so we have also updated the predictive editor to support the sentences for which we have added grammar in AceWiki. Figure 8.5 shows that one can write the conditional sentence and floating point numbers in the predictive editor.

8.2.5 Preventing Data Loss

AceWiki was not able to handle *"rdfs:comment"* and *"rdfs:label"* and some data annotations, e.g., *<AnnotationProperty abbreviatedIRI="dct:description"/>*, *<AnnotationProperty abbreviatedIRI="dct:title"/>*. To prevent data loss, we added functionality to store those data in our ontology and also preserved those data while exporting. However, it was not implemented for all annotations because of time constraint, but it will be the same for all missing properties. We parsed the XML file and stored the data of our targeted properties inside AceWiki.



(a) Screenshot showing support for if-then sentence in predictive editor to write ACE sentence



(b) Screenshot showing floating point number support in predictive editor to write ACE sentence

Figure 8.5: Screenshots showing the improved predictive editor which supports if-then and floating point numbers

Part IV

Evaluation

Chapter 9

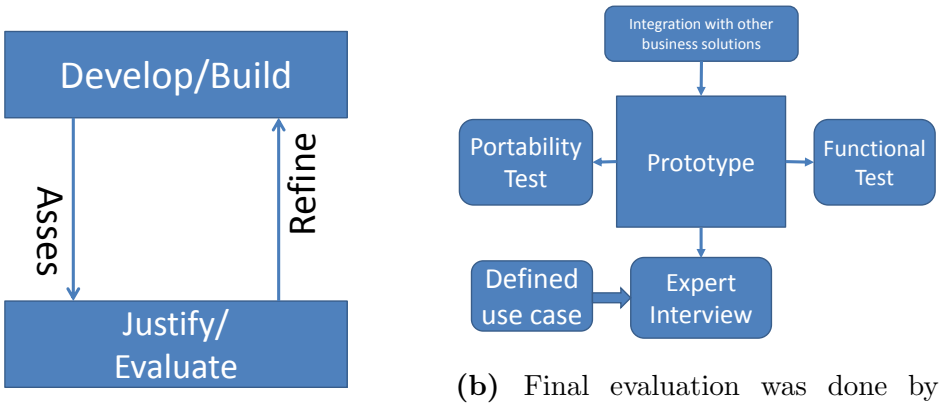
Evaluation

Following the design and prototypical implementation of a platform for managing complex ontologies using NLI (Chapters 6, 7 and 8), the goal of this Chapter is to evaluate the prototypical implementation. Section 9.1 describes the methodology of the evaluation. Sections 9.2 evaluates the prototype from the perspectives of Airbus stakeholders representing different potential application domains. Section 9.3 presents the evaluation of the functionality test and the portability test based on real-world ontologies of an aircraft study. Finally, Section 9.4 evaluates the prototype regarding integration with other business solutions.

9.1 Methodology

Hevner et al. in their information systems research framework, describe design as a "search process to discover an effective solution to a problem" - an iterative cycle to generate/test that alternates between Develop/Build and Justify/Evaluate phases [He04]. Taking this into consideration, informal interviews were performed within the research group for the first assessments of the design artifacts, as well as surveilled usage of the prototypical implementation. Valuable feedback has been yielded from these preliminary evaluations, especially concerning the prototype's user interface and usability.

To get feedback from different domain experts, we decided to perform an exploratory case study evaluation with a use case design. After that, several real ontologies of an aircraft were used to assess the prototype by doing functionality test, which Hevner et al. suggested as one of the evaluation methods. Evaluating the prototype with various ontologies also helped to check the generality of the prototype, in other words, how effectively the prototype can work with different types of ontologies. The above-described evaluation methodology is illustrated in Figure 9.1.



(a) Develop/Build and Justify/Evaluate cycles within the research group to build the final artifact

(b) Final evaluation was done by conducting five expert interviews, functional test, portability test and integrating with other business solutions

Figure 9.1: Evaluation methodology

9.2 Expert interview

This Section will discuss the performed expert interview, its design along with the case study design. Finally, the results of the interviews will be analyzed.

9.2.1 Case Study Design

The evaluation realized for the prototype is based on one exemplary use case. The use case describes the selection of a cabin ontology of an aircraft, which has to be imported inside the prototype and to be manipulated using natural language interface. At first, the wiki is populated by importing the ontology. Then required concepts are searched using the search box and by asking questions which were answered by the integrated reasoner. ACE sentences were manipulated using the predictive editor as well as new knowledge were added in the wiki. Finally, the ontology is exported in various formats for example in turtle, OWL/XML and RDF/XML and checked accordingly to find out whether the changes made in the wiki are reflected or not.

9.2.2 Interview Design

A demonstration of the prototype was held following the case study which is designed in Section 9.2.1. After that, a questionnaire was sent to the participants and we received valuable feedback and suggestions from the interviewees in response.

During the interviews, the background of the research project was discussed first and then the corresponding design artifacts were introduced:

1. Introduction to the concepts of NLI to manage complex ontology as discussed in Chapter 6, accompanied by questions addressing NLI demand on different use cases.
2. Introduction, discussion and questions regarding usability and usefulness of the prototype and also towards further improvement.

9.2.3 Participants

The choice of the interviewees is another important design decision. It is a good idea to choose participants from different domains, who deal with different kind of ontologies as well as different kind of tasks. So, they would be able to give feedback in a more generic way regarding the solution approach as well as the future extension of the prototype. Requirement specialist, ontology expert, operational intelligence expert, stakeholders in the area of Model/ Variant Management and IT platform specialists took part in this evaluation phase and gave their valuable feedback based on our questionnaire.

9.2.4 Interview Result

The qualitative expert interviews yielded comprehensive feedback for the prototype. In general, they liked different features of the prototype as well as they found those features intuitive to use. Moreover, they discussed potential use cases and gave future direction for the prototype's development.

Feedback for the Prototype

- Import functionality: All the participants found that the import functionality is very intuitive.
- Search: The interviewees found the search options helpful.
- User guidance via the predictive editor: In general, the interviewees found the predictive editor was helpful, but the also gave some suggestion to improve the human machine interface (HMI) for creating new sentences, e.g., by providing auto-completion while typing. Currently, the prototype provides suggestion words after pressing the tab key, which could be improved by providing auto-completion.

Potential Use Cases:

Interviewees were asked about potential use cases where they think that this prototype could be used. Several possible use cases were proposed:

- Managing requirements: Some users need to work with verbalized ontology which is provided by the import functionality of the prototype. As suggested by one stack holder, it is useful for requirements management.
- Model management: A user could use the sentence editor to enter new knowledge and in this case it might be useful to connect the database with existing model or document.
- To quickly create a generic ontology: The prototype could be very efficient to add different domain concepts quickly to a generic ontology and easily create the links between them. It can be done by importing the concepts that belong to this domain ontology (through a glossary or a list of domain's terms) and creating new sentences to make connections between them.

Future Direction

Interviewees were also asked about adding new functionalities, future direction or any problem they want to report. They provided valuable feedback in this regard:

- Morphological improvement of ACE: Some ACE sentences are not grammatically correct in English language, work has to be done to improve it so that end users feel more comfortable while working with the prototype.
- User management: Two out of five interviewees mentioned that they want user management so that multiple users can use the tool simultaneously.
- Auto-completion while typing: Now suggestions are shown after pressing the tab key, but one interviewee mentioned about auto-completion while typing.

9.3 Functionality and Portability Test

The prototype was tested extensively with various data from different ontologies. Key features were selected to test for this moment; those are: import functionality to populate the wiki with existing data, exporting in different formats to check data integrity, manipulating data for which we have added support on AceWiki.

Portability of an NLI system from one domain to another is a big issue since sometimes adaptation is needed by annotating the ontology or by configuring the NLI itself. It creates an overhead on the NLI and makes it difficult for the end user to work with different ontologies.

9.3.1 Dataset

We have taken five different ontologies from various data sources. Among those, four ontologies are Airbus internal and one is publicly available. The popular pizza ontology is chosen as the publicly available ontology which can be found in the website of Protégé[1].

9.3.2 Result of Functionality Test

The prototype successfully handled all the ACE sentences for which we added support. However, owl-verbalizer is not able to verbalize all owl-axioms, for this reason, some knowledge could not be added to our prototype. Moreover, AceWiki implemented a subset of ACE in their grammar, which also prevented to import some valid ACE sentences into the prototype. But, we have already improved the grammar of AceWiki to accept more ACE sentences which give a hint that it is possible to extend the grammar of AceWiki to accept other ACE sentences also.

9.3.3 Result of Portability Test

Experimenting with different ontologies shows that our prototype can handle any OWL ontology without modification, and the supported file types are dependent on those allowed by OWL API[2]. Our prototype does not require any customization to work with different OWL ontologies. Therefore, the prototype is portable.

9.4 Integration With Other Business Solutions

To evaluate the prototype regarding integration with other business solutions, we selected one software running in Airbus, which publishes ontologies in turtle format through a restful web-service. We invoked that web-service to import ontologies in our prototype. Hence, we were able to integrate our system with another software.

[1] http://protege.stanford.edu/ontologies/pizza/pizza.owl
[2] https://github.com/owlcs/owlapi

Since our prototype can also export ontologies, it would also be possible to make a round trip by exporting the ontologies to another web-service. But, because of the unavailability of such a web-service at Airbus, we could not test this scenario.

Integration with another business solution through a web-service is shown in Figure 9.2.

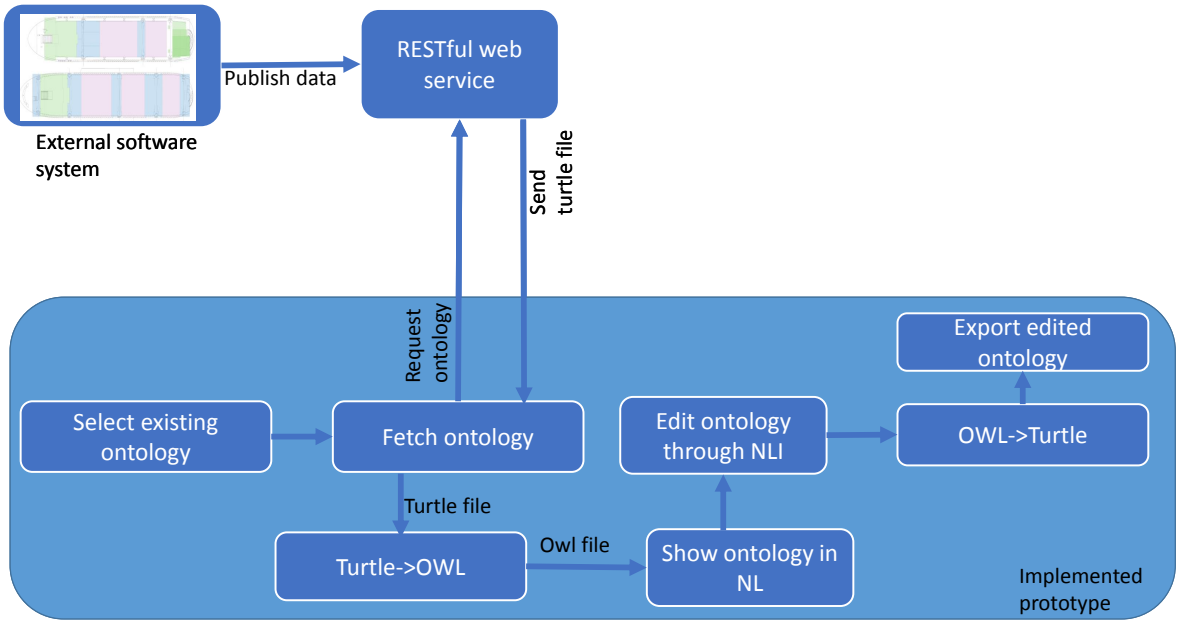


Figure 9.2: The prototype is integrated with another business solution through a restful web-service. It imports ontology from the web-service.

9.5 Summary and Discussion

While evaluating the prototype, we discovered more options to improve the tool. The qualitative interviews with Airbus stakeholders of different potential application domains yielded comprehensive feedback. In general, they liked different features of the prototype. They also gave different suggestions to improve the prototype, e.g., to improve the human machine interface (HMI) for creating new sentences, regarding the morphological improvement of ACE, to add user management functionality. Small bugs were identified and fixed during the functional test. Our prototype is portable to all OWL ontologies since it does not require any customization. Integration with another business solution through a web-service was also successful. The evaluation also helped us to organize the future work for the next phase of development.

Part V

Future Work and Conclusions

Chapter 10

Future Work

The work described in this thesis can be improved in many aspects. Some ideas are outlined here.

1. User management and activity logging: User management can be added to allow multiple users to access the tool simultaneously. Databases like CouchDB can be integrated to allow multiple users to modify the same ontology concurrently. Keeping the activity logs will help collaborative ontology development and will also contribute to find out inconsistencies.

2. Improving search mechanism: AceWiki supports simple wh-questions with exactly one wh-word, and the individuals belonging to the given class description are returned as a result. One improvement can be made by supporting more expressive query languages like SPARQL.

3. Potential use cases: Interviews with domain experts yielded some potential use cases, e.g., managing requirements, model management, etc. The prototype can be tailored to work with those use cases in future.

4. Auto-completion: The predictive editor gives suggestion after pressing the tab key. But the human machine interface (HMI) can be improved by providing auto-completion while typing a word.

5. Morphological improvement: Sometimes, the ontology engineers give names to classes or properties in such a way, that the verbalized ACE sentences seem to be wrong English sentences. For example, an object property is given the name hasDepth. This camel case format is well known to the programmers, but confusing to the domain experts. Future research can focus on this issue.

6. Improving OWL-verbalizer: Because of the limitations of the OWL-verbalizer, we were unable to import several OWL axioms into our prototype. However, we have

demonstrated a mechanism to prevent data loss. Future work can be done to improve the OWL-verbalizer.

Chapter 11

Conclusions

The primary objective of this thesis was to provide a web-based NLI to guide domain experts for managing complex product ontologies. We extended a prototype which combines a CNL (ACE) and a semantic wiki (AceWiki). This prototype is web-based, domain independent and guides the domain experts to create, update and search OWL ontologies without learning ontology engineering. Moreover, our prototype can reuse existing ontologies by importing them into the prototype and by creating domain specific lexicons automatically.

At first, a state of the art analysis was conducted to find out the right solution approach. Well known QA systems and CNL tools were analyzed in detail. It was identified that none of the existing QA systems can adapt the essential requirements of our study. On the other hand, CNLs provide a useful solution to accommodate major requirements, e.g., adding and updating data, automatic ambiguity resolution, etc. After comparing the tools, we selected ACE as the CNL as a basis for our prototype. Semantic Wikis are useful to hide the underlying complexity of the ontologies from the domain experts. Moreover, as discussed by Schaffert et al. in [Sc08], Semantic Wikis can significantly simplify the ontology engineering. After studying different Semantic Wikis, AceWiki is chosen for the prototypical implementation.

We adjusted the AceWiki and implemented a number of features to fulfill our requirements. Domain experts are guided by the predictive editor to create ACE sentences without learning the grammar of ACE. However, the expert interviews showed that the predictive editor needs to be improved for better usability.

The evaluation of the prototype with Airbus stakeholders yielded valuable feedback. In general, they liked different features of the prototype, found those features intuitive to use and also gave suggestions for future work, e.g., to add user management. Functional tests with five real-world ontologies showed that the prototype can successfully handle

all the ACE sentences for which we added support. Portability test with dataset from different domains demonstrated that our prototype is highly portable to the different OWL ontologies since it does not need any customization. Moreover, successful integration of the prototype with another business solution shows that the prototype can be integrated with other systems through web services.

Appendices

Appendix A

List of Abbreviations

The following Table describes the full form of various abbreviations used throughout the thesis.

	Abbreviation	**Meaning**
1.	NL	Natural Language
2.	CNL	Controlled Natural Language
3.	NLI	Natural Language Interface
4.	KB	Knowledge Base
5.	KR	Knowledge Representation
6.	QA	Question Answering
7.	NLP	Natural Language Processing
8.	WYSIWYG	What You See Is What You Get

Table A.1: Abbreviations list.

Appendix B

User Guide

Here we will show step by step how to run the prototype on Windows operating system and how to use it.

1. Install SWI Prolog[1]. Add SWI-Prolog's bin-directory to the PATH system variable. Something like:

```
SWI_HOME_DIR = <SWI-Prolog home directory>
PATH=%SWI_HOME_DIR%\bin\;%PATH%
```

2. Run APE web-service: Open a command prompt and go to the path of ape.exe. Run below command:

```
ape.exe -- -httpserver -port 8001
```

   Here *port* refers to the port which is used in the web.xml file of the prototype. Port 8001 is used for APE in our prototype which is deployed in Airbus. More information can be found in the website of APE[2].

3. Run OWL-Verbalizer web-service: Open a command prompt and go to the path of owl-to-ace.exe. Run below command:

```
owl-to-ace.exe -httpserver -port 5123 -workers 2
```

   Port 5123 is used for OWL-Verbalizer in the web.xml file of the prototype. "Above command will start a webserver (SWI HTTP server) on port 5123 with 2 worker threads. The optimal number of workers depends on the number of CPUs"[3].

[1] http://www.swi-prolog.org/
[2] https://github.com/Attempto/APE
[3] https://github.com/Kaljurand/owl-verbalizer

4. Download and install Jetty server. Our prototype works best with version 7.6.19 of Jetty.
5. Put the war file of our prototype inside <jetty_root_folder>/webapps directory.
6. Run Jetty server.
7. Now access the prototype: http://localhost:8080/acewiki/acewiki/
8. Click the Import button of the prototype and import an OWL ontology. Now the prototype is ready to use.

Appendix C

Questionnaire

The questionnaire which is used for the expert interviews is shown below.

Expert's Name:
Field of expertise:

1. The import process is intuitive

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly Agree
☐	☐	☐	☐	☐

(Optional) Include any comments you may have regarding the import functionality in the field below:

---

2. The search options are helpful

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
☐	☐	☐	☐	☐

(Optional) Include any comments you may have regarding the search functionality in the field below:

---

3. The sentence editor to guide the user to create new sentence in Controlled Natural Language (CNL) is useful

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
☐	☐	☐	☐	☐

(Optional) Include any comments you may have regarding the predictive editor in the field below:

---

4. Creating new concepts is intuitive.

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
☐	☐	☐	☐	☐

(Optional) Include any comments you may have regarding the creating new concepts functionality in the field below:

---

5. Do you have any use case in mind where you can use this prototype? If yes, then what is the use case and what modifications the prototype may need to support the use case?

---

6. Do you have any comment on how the prototype could be improved?

---

7. Do you have any specific problem to report?

---

8. The system is easy to use

Strongly Disagree	Disagree	Neither Agree nor Disagree	Agree	Strongly Agree
☐	☐	☐	☐	☐

9. What did you like best about the system?

---

10. We provide import option from OWL/XML, Turtle and RDF/XML files. And the prototype also exports ontology in OWL/XML or Turtle format. Do you need any other input/ export format?

Bibliography

[ADR06] Auer, S.; Dietzold, S.; Riechert, T.: *OntoWiki–a tool for social, semantic collaboration.* In *The Semantic Web-ISWC 2006.* pages 736–749. Springer. 2006.

[Al04] von Alan, R. H.; March, S. T.; Park, J.; Ram, S.: *Design science in information systems research. MIS quarterly.* 28(1):75–105. 2004.

[BL99] Berners-Lee, T.: *Weaving the Web: The Past, Present and Future of the World Wide Web by Its Inventor (with M. Fischetti). London: Orion Business Books.* 1999.

[Bu08] Buffa, M.; Gandon, F.; Ereteo, G.; Sander, P.; Faron, C.: *SweetWiki: A semantic wiki. Web Semantics: Science, Services and Agents on the World Wide Web.* 6(1):84–97. 2008.

[CP82] Church, K.; Patil, R.: *Coping with syntactic ambiguity or how to put the block in the box on the table. Computational Linguistics.* 8(3-4):139–149. 1982.

[Da] Damljanovic, D.: *Natural Language Interfaces to Conceptual Models.* PhD thesis.

[DAC10] Damljanovic, D.; Agatonovic, M.; Cunningham, H.: *Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction.* In *The semantic web: Research and applications.* pages 106–120. Springer. 2010.

[De08] Denaux, R.; Holt, I.; Corda, I.; Dimitrova, V.; Dolbear, C.; Cohn, A. G.: *Roo: A tool to assist domain experts with ontology construction.* In *Proceedings of the 5th European Semantic Web Conference.* 2008.

[De09] Denaux, R.; Dimitrova, V.; Cohn, A. G.; Dolbear, C.; Hart, G.: *Rabbit to OWL: ontology authoring with a CNL-based tool.* In *Controlled Natural Language.* pages 246–264. Springer. 2009.

[Do07] Dolbear, C.; Hart, G.; Kovacs, K.; Goodwin, J.; Zhou, S.: *The Rabbit language:*

description, syntax and conversion to OWL. Ordnance Survey Research Labs Technical Report. 2007.

[Dz06] Dzbor, M.; Motta, E.; Buil, C.; Gomez, J.; Görlitz, O.; Lewen, H.: *Developing ontologies in OWL: An observational study*. 2006.

[FS96] Fuchs, N. E.; Schwitter, R.: *Attempto controlled english (ace). arXiv preprint cmp-lg/9603003*. 1996.

[Fu07] Funk, A.; Davis, B.; Tablan, V.; Bontcheva, K.; Cunningham, H.: *Controlled language IE components version 2. SEKT project deliverable D. 2.2*. 2. 2007.

[GMM03] Guha, R.; McCool, R.; Miller, E.: *Semantic search*. In *Proceedings of the 12th international conference on World Wide Web*. pages 700–709. ACM. 2003.

[Gr87] Grosz, B. J.; Appelt, D. E.; Martin, P. A.; Pereira, F. C.: *TEAM: an experiment in the design of transportable natural-language interfaces. Artificial Intelligence*. 32(2):173–243. 1987.

[HDG07] Hart, G.; Dolbear, C.; Goodwin, J.: *Lege Feliciter: Using Structured English to represent a Topographic Hydrology Ontology*. In *OWLED*. 2007.

[He04] Hevner, A. R.; March, S. T.; Park, J.; Ram, S.: *Design Science in Information Systems Research. MIS Q.* 28(1):75–105. 2004.

[KA07] KALJURAND, K.: *ATTEMPTO CONTROLLED ENGLISH AS A SEMANTIC WEB LANGUAGE*. PhD thesis. Faculty of Mathematics and Computer Science, University of Tartu, Estonia. 2007.

[KB07] Kaufmann, E.; Bernstein, A.: *How useful are natural language interfaces to the semantic web for casual end-users?* Springer. 2007.

[KBF07] Kaufmann, E.; Bernstein, A.; Fischer, L.: *NLP-Reduce: A "naive" but Domain-independent Natural Language Interface for Querying Ontologies*. ESWC Zurich. 2007.

[Ki03] Kiryakov, A.; Popov, B.; Ognyanoff, D.; Manov, D.; Kirilov, A.; Goranov, M.: *Semantic annotation, indexing, and retrieval*. In *The Semantic Web-ISWC 2003*. pages 484–499. Springer. 2003.

[Ku08a] Kuhn, T.: *Acewiki: A natural and expressive semantic wiki. arXiv preprint arXiv:0807.4618*. 2008.

[Ku08b] Kuhn, T.: *Combining Semantic Wikis and Controlled Natural Language. arXiv preprint arXiv:0810.3076*. 2008.

[Ku10a] Kuhn, T.: *Controlled English for Knowledge Representation*. PhD thesis.

Faculty of Economics, Business Administration and Information Technology of the University of Zurich. 2010.

[Ku10b] Kuhn, T.: *An Introduction to AceWiki*. Tutorial at the Second Workshop on Controlled Natural Language, Marettimo Island (Italy. September 14 2010.

[LB11] Lehmann, J.; Bühmann, L.: *Autosparql: Let users query your knowledge base*. In *The Semantic Web: Research and Applications*. pages 63–79. Springer. 2011.

[LM04] Lopez, V.; Motta, E.: *Ontology-driven question answering in aqualog*. In *Natural Language Processing and Information Systems*. pages 89–102. Springer. 2004.

[Lo07] Lopez, V.; Uren, V.; Motta, E.; Pasin, M.: *AquaLog: An ontology-driven question answering system for organizational semantic intranets*. *Web Semantics: Science, Services and Agents on the World Wide Web*. 5(2):72–105. 2007.

[MB08] Moreno, M. S. M.; Bringert, B.: *Interactive Multilingual Web Applications with Grammatical Framework*. In *Advances in Natural Language Processing*. pages 336–347. Springer. 2008.

[No01] Noy, N. F.; Sintek, M.; Decker, S.; Crubézy, M.; Fergerson, R. W.; Musen, M. A.: *Creating semantic web contents with protege-2000*. *IEEE intelligent systems*. (2):60–71. 2001.

[Sc06] Schaffert, S.: *IkeWiki: A semantic wiki for collaborative knowledge management*. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2006. WETICE'06. 15th IEEE International Workshops on*. pages 388–396. IEEE. 2006.

[Sc08] Schaffert, S.; Bry, F.; Baumeister, J.; Kiesel, M.: *Semantic wikis*. *software, IEEE*. 25(4):8–11. 2008.

[Sc09] Schaffert, S.; Eder, J.; Grünwald, S.; Kurz, T.; Radulescu, M.: *KiWi–a platform for semantic social software*. In *The Semantic Web: Research and Applications*. pages 888–892. Springer. 2009.

[SH07] Siorpaes, K.; Hepp, M.: *myOntology: The marriage of ontology engineering and collective intelligence. Bridging the Gep between Semantic Web and Web*. 2:127–138. 2007.

[Sm09] Smart, P. R.; Bao, J.; Braines, D.; Shadbolt, N. R.: *Development of a controlled natural language interface for semantic mediawiki*. In *Controlled Natural Language*. pages 206–225. Springer. 2009.

[Un12] Unger, C.; Bühmann, L.; Lehmann, J.; Ngonga Ngomo, A.-C.; Gerber, D.; Cimiano, P.: *Template-based question answering over RDF data*. In *Proceedings of the 21st international conference on World Wide Web*. pages 639–648. ACM. 2012.

[Vö06] Völkel, M.; Krötzsch, M.; Vrandecic, D.; Haller, H.; Studer, R.: *Semantic wikipedia*. In *Proceedings of the 15th international conference on World Wide Web*. pages 585–594. ACM. 2006.

[WMS04] Welty, C.; McGuinness, D. L.; Smith, M. K.: *Owl web ontology language guide. W3C recommendation, W3C (February 2004) http://www. w3. org/TR/2004/REC-owl-guide-20040210*. 2004.

[ZC03] Zhang, J.; Chung, J.-Y.: *Mockup-driven fast-prototyping methodology for web application development. Software Practice & Experience Journal, 33 (13), 2003, pp.* page 1251. 2003.