TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

PROTOTYPE SYSTEM FOR AUTOMATIC ONTOLOGY CONSTRUCTION

Ludovic Jean-Louis

MASTER THESIS 2007
INFORMATION TECHNOLOGY

TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

EXAMENSARBETE 2007 Information Technology

PROTOTYPE SYSTEM FOR AUTOMATIC ONTOLOGY CONSTRUCTION

Ludovic Jean-Louis

Detta examensarbete är utfört vid Ingenjörshögskolan i Jönköping inom ämnesområdet datateknik. Arbetet är ett led i teknologie magisterutbildningen med inriktning informationsteknik. Författarna svarar själva för framförda åsikter, slutsatser och resultat.

Handledare: Eva Blomqvist
Examinator: Vladimir Tarassov

Omfattning: 20 poäng (D-nivå)
Datum:
Arkiveringsnummer:

Abstract

Though a constantly increasing number of ontologies are now available on the Internet, the ontology construction process remains generally a manual task, so consequently an effort demanding task. As no unified ontology construction method is available in the literature, researchers started investigating different frameworks for automatically generating ontologies and, therefore shorten the time required for their construction. This master's thesis presents a prototype system for automatic construction of ontology, based on ontology design patterns and unstructured texts, such as natural language texts. The use of ontology design patterns allow constructing well structured ontologies and reducing the demand of knowledge experts. A difference between our prototype system and the systems presented in the literature is, the possibility to increase accuracy of the generated ontology by selecting the more relevant terms and associations from the unstructured text and match them against the ontology design patterns. Also, a matching score is introduced to define the level of similarity between the terms extracted and the ontology design patterns. By setting a threshold value on the matching score, the relevant ontology design patterns are selected and used for the ontology construction process. The ontology construction framework used by the prototype system has been developed by the research group in Information Engineering of the School of Engineering, Jönköping University.

Sammanfattning

Även fast ett ökande antal ontologier är tillgängliga på Internet, är ontologikonstruktion fortfarande till största delen en manuell process, som därigenom kräver en stor arbetsinsats. Eftersom ingen enhetlig metodologi för att konstruera ontologier finns i litteraturen, forskare började undersöka olika ansatser för att automatiskt generera ontologier och därigenom förkorta konstruktionstiden. Detta examensarbete presenterar ett prototypsystem för automatisk konstruktion av ontologier, baserat på designmönster för ontologier och ostrukturerad text (text i naturligt språk). Att använda designmönster ger en välstrukturerad ontologi och minskar behovet av expertkunskap. En skillnad mellan vårt system och system i litteraturen är möjligheten att få en mer korrekt ontologi genom att välja de mest relevanta termerna och relationerna från texterna och matcha dem mot designmönstren. Ett värde för överensstämmelsen har införts för att kunna beskriva hur stor likhet som finns mellan termerna och designmönstren. Genom att sätta ett tröskelvärde väljs de relevanta designmönstren ut och används för att konstruera ontologin. Den generella processen för ontologikonstruktion som används av prototypsystemet har utvecklats av forskningsgruppen i Informationsteknik vid Ingenjörshögskolan i Jönköping.

Acknowledgements

First of all, I would like to thank **Eva Blomqvist**, the supervisor of this final thesis work, for the guidance provided during the different revisions of this document, and her immensely helpful comments, criticisms and suggestions.

Also, I want to express thanks to all the teachers from the Information Engineering group, the foreign and Swedish students, all of them have somehow contributed to that wonderful experience I had during my stay in the city of Jönköping.

Finally, I want to express gratitude to my family, my mother and my father, my brother, my aunts and uncles. I would not have made it without their encouragements and their support. Also to my cousins, who was always present in good and bad times when I needed someone to give me a piece of mind.

Key words

Automatic Ontology Construction, Ontology Design Pattern, Protégé, Ontology

Contents

List of Figures

List of Abbreviations

ALC: Attributive Language with Complements

AOC: Automatic Ontology Construction

DAM+OIL: A combination of Darpa Agent Markup Language and Ontology Inference Language

HTML: HyperText Markup Language

NEPOMUK: Networked Environment for Personalized, Ontology-based Management of Unified Knowledge

ODP: Ontology Design Pattern

OKBC: Open Knowledge Basic Connectivity

OSI: Open System Interconnection

OWL: Ontology Web Language

POM: Probabilistic Ontology Model

RDF: Resource Description Framework

RDFS: Resource Description Framework Schema

RTF: Relative Term Frequency

SDP: Software Design Pattern

TFIDF: Term Frequency Inverted Document Frequency

T-Rex: Trainable Relation Extraction

UML: Unified Modelling Language

XML: eXtensible Markup Language

1 Introduction

During the past years, a wide range of ontologies have been constructed by a large amount of researchers and developers world-wide. Those ontologies have different purposes and belong to different areas of activities such as management of enterprise knowledge and competences, bioinformatics, e-commerce, etc. The Information Engineering group of Jönköping University focuses on two research aspects that are information logistics and knowledge supply. Ontologies are used in those research fields to collect and organize knowledge, adapt semantics to be comprehensible by machines.

As no united ontology construction method has been implemented so far, and in order to construct ontologies in a formal and reusable manner, an automatic ontology construction (AOC) method based on ontology design patterns (ODPs) has been realized by the research group on Information Engineering [1].

This thesis is a part of the Master of Science program of Information Technology at the School of Engineering in Jönköping.

1.1 Background

Originally ontology was used in philosophy to investigate conception of the reality using entities and relationships to describe so called categories of being in metaphysics. Nowadays ontologies are used in computer science to describe the knowledge of a specific domain.

Ontologies have been used in a wide range of projects in different areas; in bioinformatics, the project Gene Ontology[1] provides a description of the molecular function. Also in software development area, project NEPOMUK[2] - Networked Environment for Personalized, Ontology-based Management of Unified Knowledge – aims at improving sharing of knowledge by adapting a personal desktop into a collaborative environment.

Regardless the accomplishment of many projects based on ontologies, there is no standard method for building ontologies. In [2] several methods are described and evaluated, one conclusion of this report is that the methodologies for building ontologies lack of maturity and therefore are not united. Some of the methods described in [2] require a lot of manual effort since they are based on expert knowledge. As a result the different steps involved in the construction process are hard to automate and are realized as manual task or using semi-automatic tools.

[1] http://www.geneontology.org/
[2] http://nepomuk.semanticdesktop.org/xwiki/bin/Main1/

Some researchers now propose approaches to automate the construction of ontologies, in [1] the ontology construction process is based on design patterns that have the advantage of constructing well structured ontologies. In [3] the ontology construction process is based on knowledge extraction tools. This approach faced some difficulties with duplicate information from different sources, since the same knowledge can be expressed with different words or expressions. In [4] ontologies are built based on the reuse of existing ontologies that are available on the Internet, the advantage of this approach is that less domain-expert knowledge are required since existing ontologies analysed by experts are reused to enrich other ontologies.

Important issues for ontology builders are, reducing the manual tasks required to construct ontologies, reducing involvement of domain-expert knowledge and constructing well structured ontologies. As a result, this thesis will focus on how to design and implement a prototype system for automated ontology building by using ODPs and unstructured text.

1.2 Purpose/Objectives

The purpose of this thesis is to study how an enterprise ontology can be constructed automatically using ontology design patterns and a text corpus. Although ontologies can be built manually or semi-automatically, and regarding the amount of effort required for ontology construction [5], the automatic construction process should facilitate construction of ontologies and elevate the reliability of the constructed ontologies by using different threshold values for ODP selection. In order to evaluate the efficiency of the ontology construction process described in [1] the prototype system should be able to interface with an existing ontology editor and should provide both ontology and ODP management.

The prototype system will also help in validating the general framework for automatic ontology building presented in [1], and the comparative study between two ontology construction methods presented in [6]. A succeeding goal is to evaluate the reliability of the ontology generated by the prototype system.

1.3 Limitations

During the presentation of the subject by Eva Blomqvist, a member of the Information Engineering research group, some limitations concerning the prototype system implementation were defined:

- Both ontology and ontology design patterns shall be constructed through the Protégé OWL-framework.
- The prototype system shall be created as a plug-in for the Protégé environment.
- The prototype system shall be implemented in Java to ease compatibility with other reusable tools, and it should be easy to adapt new components to its functionalities.
- The prototype system shall work according to the automatic ontology construction framework presented in [1].

1.4 Thesis outline

This document describes the different steps executed during the final thesis work. It is divided into five parts. Part 1 is the previous introduction that presents the prototype system environment, the goals that are expected to be achieved by the prototype system and finally the limitations of the final thesis work. Part 2 gives some definitions for the main concepts used in the ontology construction process, some examples of automatic ontology construction methods and their purpose, a presentation of the ontology editing environment Protégé, and finally a presentation of some terms extraction and string matching tools. Part 3 describes the method followed for implementation of the prototype system. Part 4 gives explanations concerning the realisation of the prototype system functionalities. An example of an automatically constructed ontology is presented and analysed in part 5. Finally part 6, draws conclusions about the results achieved during the thesis work.

2 Theoretical Background

This part presents and explains the main concepts and mechanisms that compose the theoretical base of the thesis work. The following notions will be presented in the subsequent sub-sections;

- Ontology (section 2.1)
- Automatic ontology construction methods (section 2.2)
- Protégé environment (section 2.3)
- Information extraction (section 2.4)
- String matching (section 2.5)
- Ontology design patterns (section 2.6)
- Software requirement and design description (section 2.7)

2.1 Ontology

A great amount of definitions for the term ontology can be found in the computer science literature, therefore this section give an overview of those definitions and also presents some ontology representation languages.

2.1.1 Ontology definition

Different definitions more or less precise can be found for ontology. According to [7] an ontology is "*an explicit specification of a conceptualization*", where a conceptualization is a simplified representation of an area of the real world. For instance an ontology on the cinema would include information about the number of rooms in the cinema, the number of seats in each room, the size of the screen in each room, etc. This definition does not indicate the importance of the relations between the objects used in the conceptualization.

Another more precise definition can be found in [1] and [8], it defined ontology as "*A hierarchically structured set of concepts describing a specific domain of knowledge that can be used to create a knowledge base. Ontology contains concepts, a subsumption hierarchy, arbitrary relations between concepts, and axioms. It may also contain other constraints and functions*".

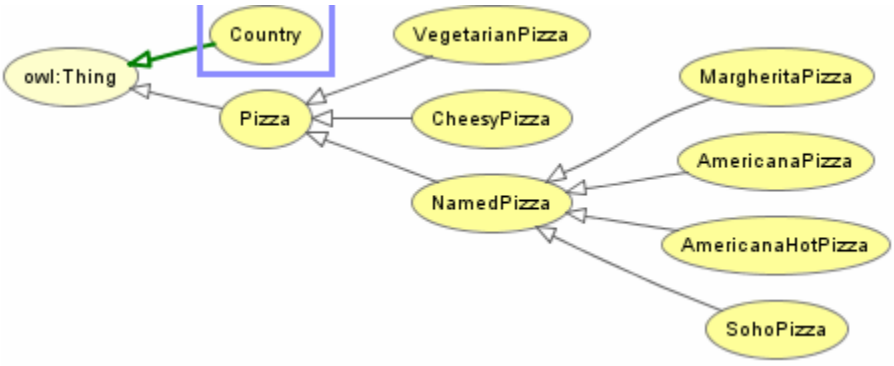

Figure 2-1 Example of Pizza ontology

In Figure 2-1 an example of an ontology is shown, in this ontology hierarchy links are made between the pizza names (MargueritaPizza, AmericanaPizza, etc.) and the "NamedPizza" concept. Also the pizzas are divided into two categories "CheesyPizza" and "VegetarianPizza".

During the past years, ontologies have been largely used in the Knowledge Management area, in order to build applications that are based on common knowledge for a specific domain (e.g. the Gene Ontology[3]), or knowledge-based service that are able to use the Internet as described in [3]. This is due to the fact that an ontology aims at reusing and sharing knowledge across systems and the users of those systems [5]. For instance in [3] an application has been used to automatically query a knowledge base at hand, and on other hand generate biography of artists.

2.1.2 Ontology representation languages

According to the previous definitions, ontology helps in describing a domain of knowledge. Consequently, this domain of knowledge needs to be represented in a machine understandable language in order to perform basic operations such as query or storage. As a result, different classes of languages allow representing ontology; **Frame-based languages**, **Description Logics-based languages**, **XML-related languages**, etc [9].

Frame-based languages: In [9] a frame is defined as "*a data structure that provides a representation of an object or a general concept*". Frames can be considered as classes in object-oriented languages but without methods .So called slots are used to represent frame attributes and associations between frames [9]. Examples of Frame-based languages are; Ontolingua (also used for the name of the system compatible with the language), OKBC (Open Knowledge Base Connectivity) [9], etc.

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

Description Logics-based languages: Permit thanks to a formal logic-based semantics, to represent the knowledge of a specific domain in a well structured way [9]. Description logics main idea is to use basic sets of concepts and binary relations to create complex concept and relation expressions [9]. Examples of Description Logics-based languages are; ALC, DAM+OIL, OWL (Ontology Web Language)[9], etc.

XML-related languages: In addition to validate XML (eXtensible Markup Language) documents XML-related languages can be used to represent and perform operations on information (metadata) contained on the Web documents [9]. The languages used for ontology representation are RDF (Resource Description Framework) and RDFS (RDF Schema). Both RDF and RDFS are defined in XML syntax. The main idea of RDF is to use resources (e-g a web page) and properties (a specific attribute of a resource) to create statements in form of "*subject-predicate-object expressions*" [9]. Here a subject is considered as a resource. RDFS enriches RDF by providing mechanisms to structure RDF resources such as defining restrictions on resources, defining classes and subclasses, [9] etc.

2.2 Automatic ontology construction method

Since several ontology construction methods are available in the literature, this section aims at, giving an overview of existing AOC approaches, and finding differences and similarities among the different approaches. Finally the ontology construction framework of the prototype system is presented.

2.2.1 Existing automatic ontology construction approaches

As previously said, the purpose of the thesis is to implement a prototype based on the AOC process described in [1]. Nevertheless other AOC processes have been used in other areas and for different purposes as in [3][4][10][11].

In [3] the methodology used for automatically building ontologies consists of applying information extraction tools on online web pages and then combining this information with an ontology and the WordNet lexicon to populate a knowledge base. This knowledge base is finally queried to automatically construct biographies about artists. One difficulty faced by this experiment was the duplicate information in the documents that created redundant explanations. This difficulty is also mentioned in [12] as a problem in the AOC approach used by the semantic agent InfoSleuth. A proposal solution in [12] to solve the problem of different sentences that refers to the same concept is, "*differentiate them via the co-occurrence frequency"*, that is to say; take into account how often the same sentence appears in text. Though the process we intend to implement is different there are similarities in the extraction of knowledge step since we need to extract terms and relations or associations in a text corpus.

In [11] ontologies are automatically built from statistical treatment of biological literature. The aim of the method used in [11] is to extract terms from their frequency of appearance in the documents and the group of gene products. Key-terms are also extracted for the associated genes. Although the method described in [11] has produced satisfying results such as identifying easily the genes that share common information in the literature after the automatic classification of the genes, this method is not suitable for our purpose since the ontologies are built by grouping concepts that have similar information and functions in the literature to enrich the GO ontology.

In [10] a different method for AOC is proposed based on tree main sources, a technical text corpus, a plant dictionary and finally a multilingual thesaurus. A different term extraction approach [11] and [3] is used in this case, the approach for terms extraction uses a Shallow Parser. The methodology used in [10] for ontology construction can be summarized in three steps; i) extraction of terms based on text corpus, ii) dictionary based ontology extraction to extract relational information with other plants since the ontology domain is plants iii) thesaurus translation to ontology terms. One main advantage of this methodology is its relatively high reliability, 87% accuracy for the system, the 13% of error is due to terms extraction errors.

As suggested in [13] ontologies are automatically constructed by reusing existing knowledge. The method used aims at improving the reliability problem of automatically generated ontologies. In [13] the summarized process for building ontologies is; i) constructing a frame ontology for a specific domain from WordNet lexicon, ii) combine knowledge from domain expert with the frame ontology previously built. One disadvantage of the suggested process in [13] is that the knowledge from the domain expert is not collected automatically.

Although the AOC process suggested in [14] is also based on reuse the approach used is different than the one in [13]. In [14] the goal is to generate ontologies from existing ontologies by using an ontology search engine to find different ontologies of the same specific domain and then combine fragments of those ontologies to construct a more complete ontology. This approach is efficient since well structured ontologies that have already been checked by domain expert are reused. Another advantage is that more and more ontologies are available via ontology search engines. On the other hand the approach suggested in [14] is inefficient, when it comes to building ontologies for a new domain when few ontologies are available through search engines, or when the available ontologies are not reliable because of a lack of domain experts.

2.2.2 Automatic ontology construction method for the prototype

The prototype system is intent to follow at a first stage, the general framework for automatic ontology construction developed by the Information Engineering research group of the School of Engineering, Jönköping University, as presented in [1]. A next stage for the future update of the prototype could be to add a different methodology for AOC.

The main idea of the ontology construction approach presented in [1] is, to extract terms and concepts from a text corpus (a text corpus is a set of text files), match those extracted terms against the terms and concepts contained in a set of ODPs and afterwards select the patterns that best match the extracted terms and associations to construct the ontology. The steps followed to construct an ontology automatically are:

1. Construct ontology design patterns.
2. Extract terms from a text corpus.
3. Match extracted terms against concepts in patterns.
4. Extract associations from a text corpus.
5. Match the extracted associations against associations in patterns.
6. Calculate a matching score that reflect the matching process of the extracted terms and associations against the ODPs.
7. Select the successfully matched ODPs, that is to say the ones that have the most concepts and associations that match the extracted terms and associations.
8. Construct ontology with selected patterns, and extracted terms and associations.

A common step that can be found among the ontology construction framework presented in [1] and the others presented in [10][11] is that, all of them are using terms extraction. On the other hand only the approach in [1] uses ODPs and a threshold for patterns selection.

A comparative study presented in [6] has shown that, it is not yet possible to measure the difference between manual construction approaches and this method based on ODP since; the main concepts are included as part of the ontology in priority since, they are used by the enterprise. However, when using the approach in [1], the main concepts are not included in priority in the ontology since; the method includes only concepts that are in the ODPs.

2.3 Protégé environment

Protégé is a freely available environment for ontology construction, it was developed using the Java language at Stanford University[4]. In addition to be open-source, various plugins are available for extending ontology construction, constraint axioms and integration functions[5]. Protégé is available in two different frameworks, Protégé-Frames and Protégé-OWL, for our purpose, we will use the Protégé-OWL since it was a requirement that the prototype system supports this framework for both ontology and ODP construction[6].

[4] http://protege.stanford.edu/
[5] http://protege.stanford.edu/download/plugins.html
[6] http://protege.stanford.edu/overview/protege-owl.html

2.3.1 Ontology building with Protégé

As previously said, Protégé is an environment for ontology building, it has been used in several projects, as [3] and [15] during the process of automatic ontology building. In [3], Protégé is not used to build the ontology but to link a knowledge base and an ontology server. In [15], Protégé is used to build an e-learning knowledge base ontology, this knowledge base is then combined with web services in order to provide dynamic course construction.

Protégé is an extensible environment, through the use of many freely available plugins, an interested reader is advised to see the Protégé web site for more information. For instance, Query Export Tab – permits to query Protégé knowledge bases – Oracle RDF Data Model – deals with OWL ontologies and the Oracle RDF (Resource Description Framework) format. Other plugins have also been developed for other purposes, in [16] a plugin has been used to permit Protégé to support storage of RDF queries through the external application Sesame. In [17] a plugin has been developed to enable Protégé environment to create ontologies in the ontology web language (OWL). Six types of Protégé plugins can be identified; application, backends (Knowledge Base Factory), import/export plugin, project plugin, slot widget and finally tab widgets.

As described in [18] there are many advantages for using the Protégé environment such as, it is highly customisable for user interface and output file format, it has an extensible architecture that permits to integrate external applications, etc. Due to those advantages, and the requirements of the thesis work, our prototype will be implemented as a tab widget plugin for the Protégé environment. The functionalities, for creating an ontology, of the Protégé environment will be reused as a basis for constructing ODPs.

2.3.2 Ontology representation in Protégé

The Protégé-OWL framework uses different terms for the entities that compose an ontology than, the terms used in the ontology definition presented in section 2.1. This section aims at presenting the vocabulary used in this Protégé-OWL framework. A complete presentation of the OWL ontology components can be found in [19].

- Individuals: they are equivalents of concept instances. Examples of individuals for the concept colour are; red, green, yellow, etc.

- Properties: they are identical to concept associations, they have cardinalities, and they can be transitive or symmetric. In Protégé-OWL, the components of the properties are called property domain and property range. An example of property for the concepts “person” and “car” is Drive, the property domain is “person” and the property range is “car”.

- Classes: they are equivalent to ontology concepts. In Protégé-OWL all the classes are considered as subclasses of the class OWL:Thing.

- Class hierarchy: equivalent to taxonomy.

- Disjoint classes: permits to specify that individuals of several classes should not overlap, so that they cannot be instances of more than one class.

For the purpose of the prototype system, the previously introduced definitions will be used for representing the concepts, associations, instances, in both ontology design patterns and the generated ontology.

2.4 Information extraction

Terms extraction is a field of the information extraction domain. Information extraction aims at extracting the most valuable information from either, structured documents as HTML pages or, unstructured documents as natural language document. As shown in section 2.2.1 it is one of the main prior steps of several AOC approaches. Information extraction is required in two steps of the ontology construction framework presented in [1]; firstly for extracting terms from a text corpus, and secondly for extracting associations from a text corpus. In this part we will focus on different term extraction methods and different tools that have implemented those approaches.

2.4.1 Information extraction methods

Dictionary-based extraction methods

A definition of dictionary-based extraction is presented in [20]; the method "*uses existing terminological resources in order to locate terms occurrences in a text*". In other words, a set of concepts are stored in the dictionary and afterwards this dictionary is reused together with information learning methods to extract terms. An example of dictionary based extraction is presented in [20].

Shallow text processing

Shallow parsers allow extracting and representing linguistic structures from texts in compact data structures. They are founded on natural language components and generic linguistic knowledge. Finally, they permit to efficiently identify relations among a set of concepts [21]. If we consider an average size set of concepts, a large set of relations can be generated considering a combination of concepts without considering natural language rules [21], therefore, shallow parsers allows adding restrictions for the relations so that non-sense can be avoided.

Co-occurrence theory

The key idea of co-occurrence theory is to identify relations between a set of terms and another by analysing how often the terms occur together in several similar linguistic structures [21]. For example, if we consider a text describing the organization of the courses given at a university, we will probably find out that different teachers, "Math teacher", "French teacher", "Swedish teacher", are responsible for different departments in the university. As a result we would have several sentences describing the teacher's responsibilities, such as "the math teacher is responsible for the math department". Therefore co-occurrence theory would permit to retrieve relations between the different teachers and the department they are responsible for.

2.4.2 Tools using information extraction methods

Several tools implementing information extraction methods can be found in the literature, some of the tools are fully focusing on information extraction, and in others this task is included in the ontology construction process:

- Text2Onto: it is an ontology learning framework based on Probabilistic Ontology Model (POM). Shallow parsers are used for extracting linguistic features such as relation between words [22].

- ProtScan: it is a system to identify proteins in biomedical text corpora. It is based on a protein dictionary combined with a specialized algorithm [23].

- T-Rex: the Trainable Relation Extraction (T-Rex) is a tool for relations extraction from a text corpus based on different algorithms [24].

2.5 String matching

String matching methods or algorithms are largely used in string searching algorithms because they permit to identify the position of a string, or a set of strings, within a text or a large set of strings. String matching algorithms help to avoid comparing two strings by testing each position one at a time, or so called "naïve string search", by providing efficient and fast string comparison. For the prototype system string matching will be required for matching the extracted terms from the text corpus against the concepts in the ODPs.

In order to provide flexibility in the ontology construction process, the prototype system will provide a choice of several string matching algorithms so that the terms and concepts matched are different according to the algorithm chosen. Also, a threshold limit for string comparison will be introduced, to allow the researcher defining a starting point from which a term and a concept can be considered as successfully matched. As a result, only the best matches will be considered for the ontology construction process. In this section some string matching algorithms and some tools implementing those algorithms are presented.

2.5.1 String matching algorithms

String matching algorithms permit to quantify the similarity between two strings, the level of similarity or string distance, is computed using different mathematical formulas [25][26]. The formulas can be classified according to three distance categories [27];

- Edit-distance; the similarity between two strings a and b is, the smallest number of changes to change the string a into b, so that they are exactly the same string. Examples of Edit-distance formulas are Levenstein distance, Jaro distance, and Jaro-Winkler distance [25][26][27].

- Token-based distance; the similarity is expressed, considering the strings as a group of strings (tokens) and measuring the frequency of appearance of the sub-strings in the corpus [27]. Example of token-based distance formulas are Jaccard similarity, Cosinus similarity, and Jensen-Shannon [25][26][27].

- Hybrid distance; the similarity is expressed by combining two distance functions, a base function and a secondary function. In [27] the hybrid distance "Soft TFIDF" is introduced, it uses Jaro-Winkler function as secondary distance. In [26] a hybrid distance is presented using Jaro-Winkler function and Levenstein distance.

The table presented in Appendix 1 gives some examples of string matching by using different string metrics. In [27] a string matching experiment has been conducted using two datasets, the first one containing 841 strings equivalent to 5,765 tokens and the other one 1916 strings equivalent to 47,512 tokens. The results have shown that SoftTFIDF was the best distance measure for both string matching and clustering experiments for those datasets. To provide flexibility of the prototype system, the researcher will have the possibility to choose among formulas of the three categories for the matching of the extracted terms against the concepts in the ODPs.

2.5.2 Tools using string matching algorithms

Several tools implementing string matching algorithms can be found in the literature, most of them provide Edit-distance, Token-based distance and Hybrid distance formulas:

- SecondString: it is an open-source Java library implementing "Soft TFIDF" hybrid distance, TFIDF distance, Jaro-Winkler distance and other distances previously cited [27].

- SimMetrics: it is an open-source library available in both Java and .NET, it constitutes more than twenty similarity distance algorithms including Jaro-Winkler, Levenstein distance, and Monge Elkan distance. A complete list of distances is available in [25].

2.6 Ontology design pattern

Ontology design patterns are a derivate from the design patterns used in software engineering. Software design patterns (SDPs) have been used to provide general solutions to common problems that appear in different situations. SDPs are usually linked to a description of the pattern applicability range, the expected results from the pattern use, example pattern use cases, etc [28].

As a derivate of SDPs, ontology design patterns should be an application of SDPs specialised in ontology building. However, ODPs can be constructed either using semantic rules[7], or adapting other existing domain patterns to ODPs as presented in [1]. Patterns can also be extracted from other patterns by specializing or generalizing other patterns, examples are presented with conceptual ODPs in [29].

Ontology design patterns are described in [30] as, "*modelling abstract solutions to known problems in ontology engineering*", it is suggested that they are documented according to characteristics similar to the one used to describe SDPs; pattern name, problems solved thanks to the pattern, a domain of applicability for the pattern, etc [30]. Examples of different ODP types are presented; extensional patterns, good practice patterns, modelling patterns, those patterns have been implemented using the OWL format [30].

Ontology design patterns are defined in [31] as, entities that permit identifying design structure of ontologies, by dividing the representation of a set of terms from their definitions [30][31]. As a result, the representation and the implementation do not depend on each other. Design patterns also allow setting dependencies among the terms so that changes among the terms are alerted [31].

In [1], two methods are suggested to extract ontology design patterns;

- Map parts of database data model patterns to ontology design patterns, since the goal of the design patterns in this case is to model the structure of the enterprise knowledge.

- Convert a goal structure into an ontology design patterns, this conversion enables to include the processes used in a company as part of the ontology design patterns.

After extracting concepts using the previous methods [1], ontology design patterns are enriched with synonyms to provide a higher level of generalization. Finally a set of constraints or axioms on the associations in the pattern is added as pattern characteristics.

For the purpose of this thesis work, and as suggested in [1], we consider the ontology design patterns as ontologies that have concepts, associations between the concepts, a set of axioms that apply on the associations, and a set of synonyms for the concepts in the ODPs.

[7] http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

2.7 Requirement specification and design description

As parts of the generic software development processes (Waterfall model, Evolutionary Development model, Component-based model, etc.), software requirements specification and software design description helps in specifying the system functionalities. Also they help in decomposing the software functionalities into independent components that can be implemented. In this section we give an overview of how to represent requirement specifications and design description of software.

2.7.1 Software requirement specification

According to the software engineering literature [32], a software requirement specification is one of the prior steps of the software development process. It aims at describing user requirements, functional and non-functional system requirements. Finally the requirement specifications permit to ease the development process and, facilitate knowledge transfer to new users [33]. Software requirements can be represented in different ways:

- Natural language: software requirements are described using sentences from the normal language, tables and diagrams.
- Structured natural language: software requirements are defined through the use of defined templates such as Use Case descriptions in Unified Modelling Language (UML).
- Design description language: software requirements are defined using a pseudo-programming language such as the *Q* language [34].
- Graphical notations: software requirements are defined using graphical entities linked through relationships such as Use Case diagrams in UML.
- Mathematical specifications: software requirements are defined using algebraic presumption such as sets theory.

The requirements of the prototype system have been described using natural language specification according to the IEEE Standard 830-1998 [33]. The complete description of the prototype requirement specification can be found in Appendix 2.

2.7.2 Software design description

Design description or architectural design of software aims at dividing a system into a set of structured sub-systems, which permit to fulfil the requirements identified during the requirement specification process [35]. The design process is composed of three phases [32]; i) decompose the system into main sub-systems and identify links between the sub-systems, ii) define a control model, iii) decompose the sub-systems into modules. Different generic software design can be found in the literature [32]:

- Repository model: a central repository store the shared data and the sub-system are constructed around this central repository. This model is convenient for sharing of large amounts of data across the system.

- Client-Server model: a client requires services provided by a specific server over a network.

- Object models: decompose the system into object classes, and specify the classes' attributes and methods.

- OSI reference model: the Open Systems Interconnection model is a layered model for communication between systems over a network.

- Layered system model: each layer can be implemented separately from the others to run on a separate server. This model is generally used for web-based systems.

Since the prototype system is not intended to work over a network, to communicate with other systems, or to work as a web service, the generic software design chosen for the prototype system is an object model. The complete design description of the prototype system can be found in Appendix 3. The design description follows the IEEE standard 1016-1998 [35].

3 Methodology

As several AOC frameworks were present in the literature, the first step was to identify which tool was using a framework similar or close to the framework required for our prototype. Unfortunately, no such tool was identified, but some steps of the construction framework, such as string matching, already implemented in tools were identified. Also, some suggestions for the existing tools Text2Onto [22] and SecondString [27], which could be reused, have been made during the presentation of the subject by Eva Blomqvist.

Since reusable tools were identified for string matching process, information extraction process, and the ontology area was a new experience, an adaptation period was necessary for exploring the tools functionalities, and ontology construction using Protégé environment. After this adaptation period, and regarding the detailed description of the ontology description framework, a clear idea of the functionalities required for the prototype started to appear. Also, the AOC framework required for the prototype system was already established so did not require any changes. For that reason the requirements for the framework were clearly defined.

Consequently a waterfall software development process has been chosen. In order to establish a good basis for the development process, a great time has been spent on the description of the requirements so that the future users of the prototype are satisfied with the functionalities that should be implemented. From the detailed description of the requirements established during the previous stage, each requirement has been analysed and divided into six modules (or design entities); "*Extraction*", "*Matching*", "*Ontology Design Pattern Handling*", "*Graphical User Interface*", "*Score Computation*", "*Ontology Construction*". Figure 4-1 shows the complete architecture of the prototype system. In order to facilitate the design description of the complete system, several components have implemented in parallel to their design description. With respect to the waterfall development process, all the module functionalities have been tested individually after being coded. Finally the functionalities have been joined together and linked to the graphical user interface.

4 Realisation

The prototype system presented in this thesis aims at implementing the automatic ontology construction framework presented in [1], in addition to allow management of ODPs, and management of the generated ontology. In this section, the prototype requirement specification, the design choices and the implementation method followed are presented.

4.1 Requirement specification for the prototype system

During the requirement specification process, some **specific requirements** that should be fulfilled by the prototype system have been identified, together with their **detailed description**. The following general requirements have been defined from the ontology construction framework presented in [1]:

- Construction of ontology design pattern.
- Extract terms in a text corpus.
- Match extracted terms to the concepts in ontology design patterns.
- Extract associations in a text corpus.
- Match extracted associations to the relations in ontology design patterns.
- Compute a score based on the amount of terms and associations matched.
- Set a threshold for ontology design patterns selection.
- Select the ontology design patterns having a score above the threshold for ontology construction process.
- Build ontology with the selected patterns and the matching terms and relations.

From those general requirements, other **specific requirements** have been identified:

- Add generated synonyms and user's own synonyms to concepts in ontology design patterns.
- Select a string matching algorithm for extracted terms and patterns matching.
- Set a threshold for string matching algorithm.
- Generate a list of concepts and associations in ontology design patterns.
- Convert the extracted associations of terms to associations of concepts.
- Match converted associations against associations in ontology design patterns.

- Compute the number of matched concepts and the number of matched associations for each ontology design patterns.
- Select a formula for matching score computation.
- Set predefined values for the parameters of the matching score formula.
- Save the list of terms and concepts successfully matched.
- Save the list of terms associations successfully matched.
- Update an ontology design pattern.

According to the IEEE standard 830-1998, each specific requirement has been; i) uniquely identified, ii) ranked according to a degree a stability (stating the number of changes that could be necessary for the requirement description), iii) classified according to a degree of necessity (essential, conditional or optional), iv) described according to a stimulus/response sequence, v) linked to a list of associated requirements.

Example of a **detailed description** for the specific requirement "Update an ontology design pattern";

Name: SRE23: Update an ontology design pattern.

Purpose of feature: The AOC prototype shall permit to edit an ontology design pattern and make changes on this one. The changes can be to add/remove/update the concepts, synonyms, or associations into the pattern.

Stability: Stable.

Degree of necessity: Essential.

Stimulus/Response sequence:

User	**AOC Prototype System**
1. Request to open an ontology design pattern.	
	2. Display a file explorer.
3. Select an ontology design pattern through the file explorer.	
4. Validate the selection.	
	5. Display the ontology design pattern concepts, associations.
6. Edit the elements (association, concept, and synonym) of the ontology design pattern.	
7. Request to save the updated ontology design pattern.	
	8. Request for saving confirmation.

9. Confirm saving.	
	10. Update the file containing the ontology design pattern.

Associated functional requirements: No associated functional requirements.

The description of all the specific requirements can be found in the requirement specification document for the prototype system presented in Appendix 2.

4.2 Design Options and Decisions

In order to ease the understanding of the design choices, an analysis of the different steps performed by the ontology construction framework used by the prototype system is necessary. In this section, the architecture of the prototype system is presented together with a description of the components purpose.

The following diagram shows the interaction between the prototype system components.

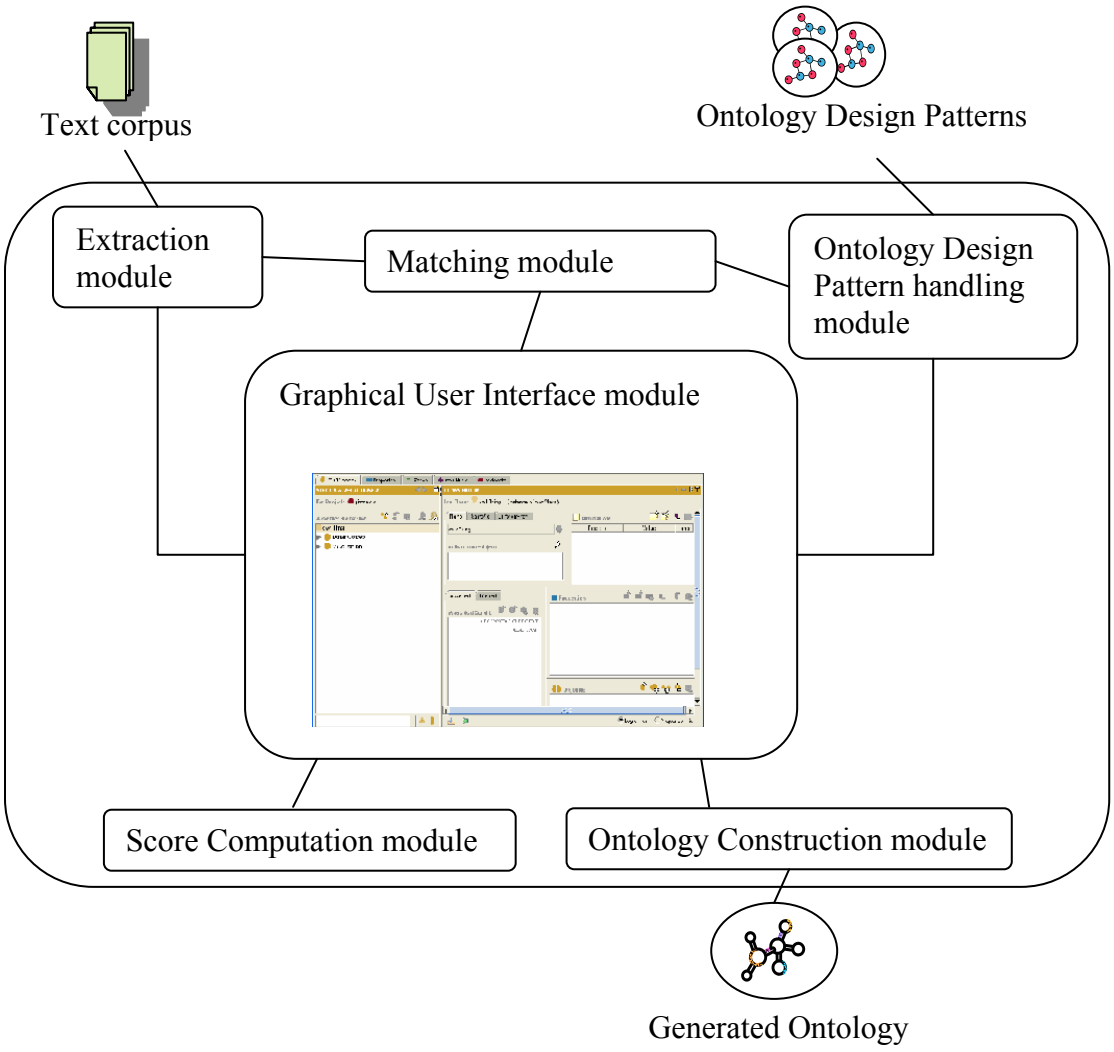

Figure 4-1 Architecture of the prototype system

4.2.1 Extraction module

The extraction module is responsible for handling terms and associations extraction from a text corpus.

Terms extraction from a text corpus

This component aims at, reusing extraction algorithms from existing information extraction tools, such as Text2Onto, in order to extract the terms from a text corpus. The terms are then saved in a text file together with, the corpus name used for extraction.

Association extraction from a text corpus

This component aims at, reusing association extraction algorithms from existing information extraction tools, extracting associations of terms from a text corpus. The associations are saved in a text file together with, the corpus name used for extraction.

4.2.2 Matching module

The matching module is responsible for handling; i) matching of extracted terms against concepts in ODPs, ii) conversion of extracted associations to associations of concepts, iii) matching of converted associations against associations in ODPs.

Matching of extracted terms against concepts in ontology design patterns

This component has two purposes; firstly to list the concepts in a set of ODPs and save them in a text file together with the pattern name. Secondly, to match the concepts against the terms extracted from the text corpus. As string metrics are used for the matching process, the string matching score of a specific term and a specific concept should be saved. Consequently, the list of terms and concept matched against each other are saved in a text file together with the matching score. In order to save only the best matched terms and concepts, a string matching threshold is used for selecting only the terms and concepts having a matching score above a defined limit. The name of the pattern, used during the matching process, is also saved with the matching score.

Conversion of extracted associations to associations of concepts

In order to improve the association matching process, it is suggested in [1] to convert the extracted associations to associations of concepts, by using the list of terms and concepts matched. The next step is, match the converted associations against the associations in the patterns. Therefore this component aims at converting, in the extracted associations, the domain and range labels, by the concept labels that best match the terms. Finally the converted associations are saved in a text file together with the text corpus name, and the ODPs name used for conversion.

For instance if we consider two extracted associations $A1$ and $A2$;
$A1 = (analysis, components)$
$A2 = (discover, components)$

And if we consider the conversion table T ;

$$T = \begin{bmatrix} analysis & analysis\_components & 0.9 \\ components & discover\_components & 0.95 \\ discover & discover\_problem & 0.7 \end{bmatrix}$$

The conversion of $A1$ and $A2$ using T should be;

$$A1' = (analysis\_components, discover\_components)$$

$$A2' = (discover\_problem, dicover\_components)$$

Matching of converted associations against associations in ontology design patterns

This component has several purposes; firstly list the associations in a set of ODPs and save them in a text file together with the pattern name. Secondly, match the converted associations against the associations in ODPs. In case a converted association matches an association in the ODPs, it is saved in a text file together with the text corpus name, and the design pattern name.

4.2.3 Score computation module

In addition to permit calculation of the matching score for each ODP, the score computation module permits the selection of the matched ontology design patterns.

Calculate a matching score for each ontology design pattern

This component aims at quantifying the amount of concepts and associations in ODPs that is matched against extracted terms and associations. As a result the component is responsible for several tasks; i) retrieve the percentage of concepts in the ODPs that is matched successfully, ii) retrieve the percentage of associations in the ontology design patterns that is matched successfully, iii) compute a score based on the percentages, iv) save in a text file the matching score of the ODPs together with the pattern names.

In addition to the **linear combination** solution for computing the matching score, the author suggests another score computation formula; "**Automated weight values**".

Linear combination score formula:

$$Score = a * \%Terms\_matched + b * \%Association\_matched$$

Where "a" and "b" are two real weight values that can be set by the user through a graphical interface.

Automated weight values formula:

$$\alpha = \left[(Amount\_Of\_Concept\_In\_Pattern)/(Amount\_Of\_Association\_In\_Pattern)\right]$$

$$\beta = \left[(Amount\_Of\_Association\_In\_Pattern)/(Amount\_Of\_Concept\_In\_Pattern)\right]$$

$$Score = 1/2 * \alpha * \%Terms\_matched + 1/2 * \beta * \%Associations\_matched$$

Selection of matched ontology design patterns

This component aims at selecting the ontology design patterns that have a matching score above a definite threshold. The selected ODPs, and their associated extracted terms and associations, should be reused as input for the ontology construction process.

4.2.4 Ontology construction module

This module aims at constructing the ontology from the accepted ontology design patterns, the terms and associations extracted, and a set of heuristics for ontology construction. The resulting ontology is saved in an OWL file. In order to create an OWL file from the Java programming language, the Protégé OWL API has been used.

4.2.5 Ontology design pattern handling module

This module aims at constructing ontology design patterns. As presented in section 2.6 they are considered as common ontologies therefore the Protégé-OWL framework will be reused for their construction. The module is also used for reading the content of the OWL ODPs; the Jena 2 Ontology API [36] has been chosen for this task since it provides ontology management facilities from the Java language. Jena 2 API is used for listing the concepts and associations in ODPs, since they are considered as an ontology.

4.2.6 Graphical user interface

The user interface permits the user to interact with the previous components and setup different parameter values, for the ontology construction process;

- a string matching algorithm
- a value for the string matching threshold
- a value for the pattern selection threshold
- a formula for calculating the matching score
- algorithm for concepts and associations extraction from the text corpus

The user interface is presented as a Protégé tab-widget to the user, with a main menu regrouping all the prototype system functionalities.

4.3 Implementation

In this section details concerning the mechanisms used for the implementation of the different modules of the prototype system are presented.

The following modules were implemented through a collection of JAVA classes. In order to ease the development, all the output of the methods (list of extracted terms and associations, list of converted associations, etc.) were saved in text files, so that it was possible to operate in the middle of two steps, such as association conversion and association matching. The development was conducted using the Open source development platform Eclipse in combination with different open source tools; i) SecondString library for string matching process, ii) Jena API for OWL file processing, iii) Text2Onto for terms and associations extraction from a text corpus.

4.3.1 Extraction module

In order to extract concepts and associations from a text corpus, the prototype system uses the Probabilistic Ontology Model (POM), which is one of the ontology learning paradigms, the other is data-driven change discovery, used by the tool Text2Onto [22]. The POM aims at identifying learned structures (for instance a subject to object relation) in a text corpus and assigning probabilities to those structures. The POM is able to identify such structures thanks to different types of ontology learning algorithms [22];

- Concept extraction algorithms;
    - Relative Term Frequency (RTF)
    - Term Frequency Inverted Document Frequency (TFIDF)
    - Entropy and C-value/NC-value method
- Subclass-of relations algorithms
- Mereological relations (part-of relations) algorithms
- General relations algorithms (used to identify transitive, intransitive + complement, and transitive + complement relations)
- Instance-of relations algorithms (used to identify instances of concepts)
- Equivalence algorithms (used to identify equivalence of terms)

Finally, in addition to identify the learned structures in a text corpus, the POM is also responsible for their storage.

The data-driven change discovery permits, to identify changes in the corpus, and calculating the probabilities only for the new identified structures, without computing new probabilities for the complete text corpus [22].

For terms extraction from a text corpus, the prototype system provides three algorithms (RTF, TFIDF, Entropy and C-value/NC-value method). Concerning association extraction one algorithm allow extraction of general relations has been used. Further work could be done for adding other relation extraction algorithms. The user has the possibility to choose among different extraction algorithms via the graphical user interface.

4.3.2 Matching module

The prototype system uses different mechanisms for string matching depending on, comparing extracted terms and concept in patterns, or comparing extracted associations and associations in patterns.

Match extracted terms against concepts in ontology design patterns

In order to automate the ontology construction process, one should be able to identify one or several terms in a text corpus that refer to a concept in an ODP. As a result using "string naïve search" should be avoided for the matching process of extracted terms against the concepts in the ODPs. Therefore the prototype system provides different string matching metrics from the tool SecondString [27] for this process; JaroWinkler, Jaro, Jaccard, SoftTFIDF, TFIDF, JaroWinklerTFIDF, Level2JaroWinkler, MongeElkan, and Level2MongeElkan.

Those string metrics returns a number that expresses the level of similarity between extracted terms and the concepts in the ODPs. With reference to section 2.5, a string threshold is used for defining a starting limit for considering two strings as a match. In order to facilitate the matching process of extracted associations against associations in the patterns, the matching score of the extracted terms and the concepts in the patterns, and the name of the pattern used for matching are saved in a text file.

Convert extracted associations to associations of concepts

For improving the matching process of the extracted associations against the associations in the ODPs, the AOC framework suggests to replace the term labels in the extracted associations by the labels of the concepts in the ODPs. As a result the labels of the terms in the associations are replaced by the label of their best match concept in a state pattern. In case several best match concepts are found, the first occurrence is selected. Future work could be done to include several best match concepts and consequently include as much as new relations as there are best match concepts.

Match converted associations against associations in ontology design patterns

Once the extracted associations are converted to associations of concept label, the converted associations domain and range are matched against the ODP association's domain and range. In this case the use of string metrics is not required since the extracted associations are converted to associations of concept labels. One converted association and one association in an ODP are considered as a match is they have identical domain and range. For our purpose it is not required to have an identical association label.

4.3.3 Score computation module

For computation of the matching score of the extracted terms and associations against the ODPs, the prototype system provides two main formulas "Automated weight values" and "Linear combination" as presented in section 4.2.3. Therefore, this module permits to select a score formula, set values for the weight values (if necessary) and calculate the matching score according to the chosen formula. In addition to calculate the matching score, the score computation module is responsible for comparing the computed score against the ODP selection threshold. Only the ODPs having a matching score above this limit value are accepted for AOC process.

4.3.4 Ontology construction module

With reference to the limitations established in section 1.3, the prototype system should be able to generate the ontology using OWL syntax. Therefore OWL Models from the Protégé API are used to construct the generated ontology. OWL Models permit to create, query or delete components of OWL ontologies such as classes, properties or individuals. For our purpose an OWL model has been used for storing the structure of the generated ontology. Afterwards the content of this model has been written into an OWL file.

For constructing an ontology from the accepted ODPs, the prototype system verifies for each accepted ODP; if each concept of this ODP were successfully matched against one extracted term. When a matched concept is identified then the prototype checks if the concept is not already in the generated ontology. If not, the matched concept is added to the generated ontology, otherwise all the synonyms for this concept are added in the new ontology as synonyms for the concept.

Once all the concepts have been added to the generated ontology, the matched associations identified during the association matching process are added to the generated ontology.

4.3.5 Ontology design pattern handling module

Since ODPs are considered as ontologies, using the Protégé environment for their construction presents several advantages; i) reuse of already implemented ontology edition facilities (construction of OWL classes, creation of OWL properties, creation of OWL restrictions) ii) provide the same interface for both ontology and ODP construction. The following figure shows the Protégé-OWL ontology editor interface, with the different components for editing ontology concepts and restrictions on those concepts.

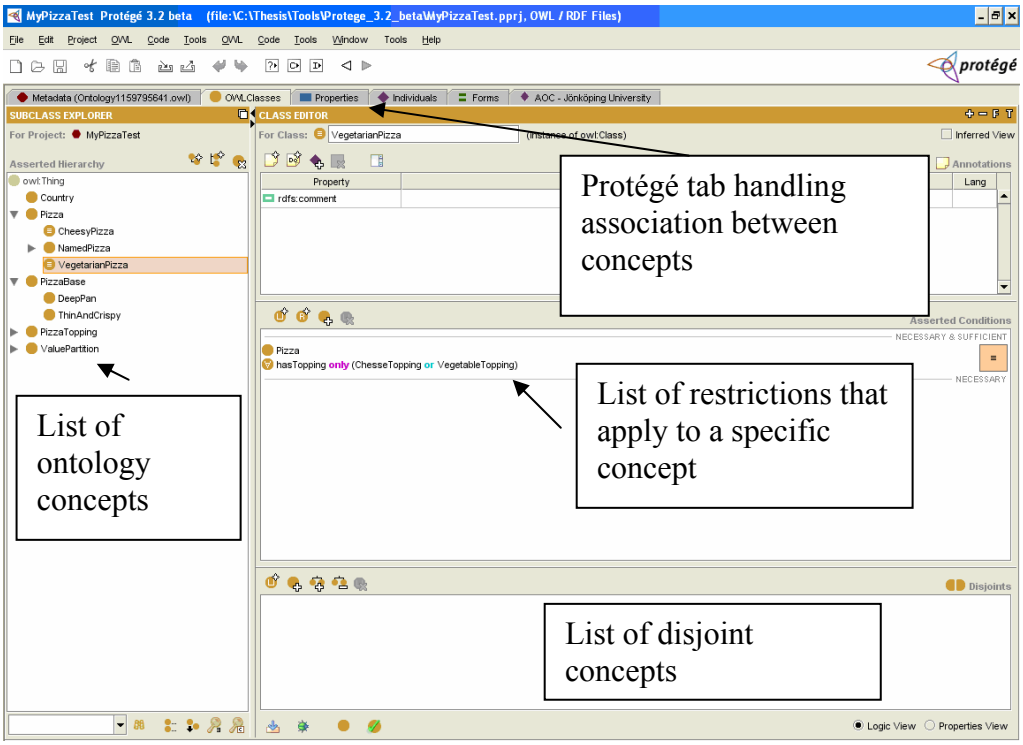

Figure 4-2 Interface of the Protégé-OWL ontology editor

In addition to permit to construct ontology design patterns, this module permits to retrieve the label of the concepts, and associations in the ODP. For this task the prototype system uses the OntModel from the Jena API. OntModel permits to wrap ontology data (concepts, relations, restrictions, etc.) from RDF or OWL ontologies. Consequently an OntModel has been used for retrieving the concept labels, association domain labels, association range labels, and the association name labels for all the ODPs that have been selected for the AOC process.

4.3.6 Graphical user interface

A graphical interface has been created to allow the user to interact with the prototype system functionalities via an easy to use and friendly interface. This user interface, shown in Figure 4-3, permits to set up the parameter values for the AOC construction process. In this section, the components of the user interface are presented.

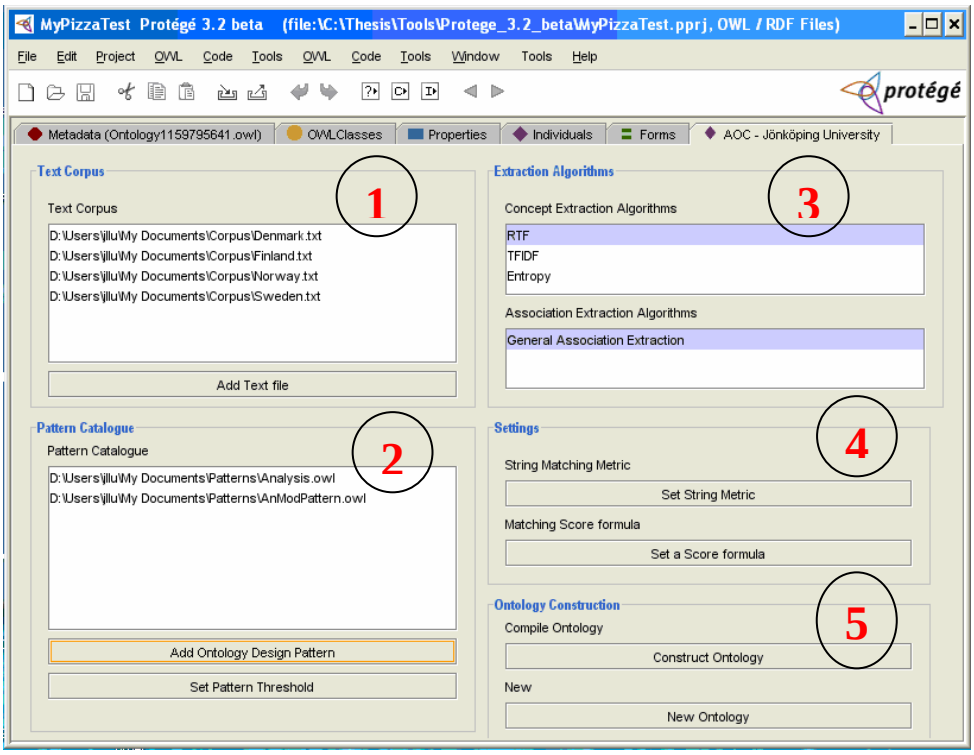

Figure 4-3 Graphical user interface of the prototype system

As shown in Figure 4-3, the user interface is composed of five panels; "*Text Corpus*", "*Pattern Catalogue*", "*Extraction Algorithms*", "*Settings*" and "*Ontology Construction*".

The "*Text Corpus*" panel (numerated 1 in Figure 4-3) allows the user to add texts from different file formats (.txt, .pdf, .html) to compose a text corpus. It is also possible for the user to remove a text file in the corpus before starting the ontology construction process.

The "*Pattern Catalogue*" panel (numerated 2 in Figure 4-3) allows the user to add or remove ontology design patterns from the pattern catalogue in a similar way as the "*Text Corpus*" panel. The user can also set a threshold value for the matching process of the ODPs against the extracted terms and associations. The Figure 4-4 shows the interface for setting the threshold value. Figure 4-5 shows the popup menu allowing removing an ODP from the pattern catalogue.

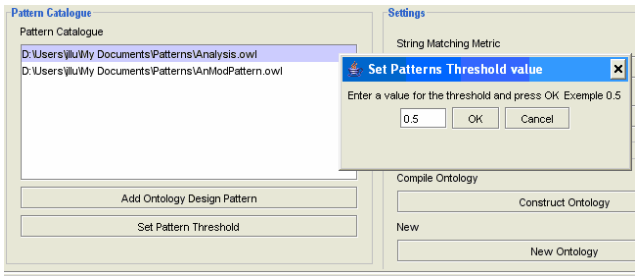

Figure 4-4 Setting of the pattern threshold value

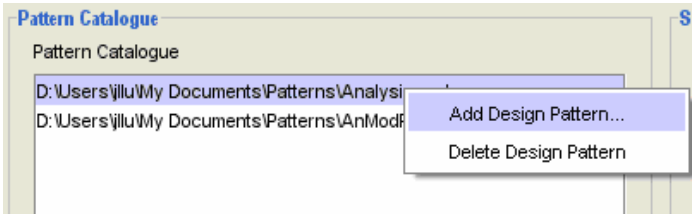

Figure 4-5 Popup menu for the management of the pattern catalogue

The "*Extraction algorithm*" panel (numerated 3 in Figure 4-3) allows choosing one or several algorithms for extracting concept and association from the text corpus.
The "*Settings*" panel (numerated 4 in Figure 4-3) allows choosing a string metric for matching the extracted terms against the concepts in the ODPs, and also choosing a formula for the computation of the matching score. It is also possible to set the string matching threshold and the values for the weight parameters of the score formula by using this panel. Figure 4-6 shows the interface for configuration of the string metric.

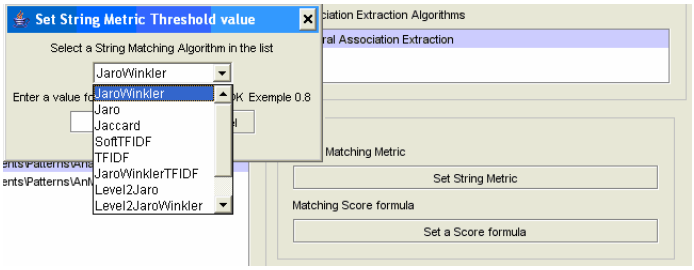

Figure 4-6 Interface for setting the string metric configuration

The "*Ontology Construction*" panel (numerated 5 in Figure 4-3) allows starting the ontology construction process and choosing a name for the generated ontology.

5 Results

This section presents the results of the work presently achieved, and how the development process has reached the objectives of the thesis work.

According to the work realized during this final thesis work, the Protégé ontology editor is convenient for construction of both ontology design patterns and generated ontology using the OWL language. The facilities offered by the Protégé API are also efficient for handling the graphical layout of the generated Protégé plugins.

Once all the prototype system requirements were defined, the methodology consisting of reusing existing tool for information extraction and string matching was very efficient, for shorten in the implementation time. So far, few effective information extraction algorithms are implemented in Java and available through the Internet, consequently few of them are proposed by the prototype system.

Even though many improvements can be brought to the prototype system - add information extraction algorithms, add score computation formulas, etc. - it benefits in terms of effort required to construct an ontology, compared to manual ontology construction, seems obvious (large amount of text treated in a relatively short time etc). The prototype system permits realizing all the steps of the AOC framework presented in [1], so it allows construction of an ontology by using a text corpus and ontology design patterns.

The terms extraction part seems to produce good results (lot of terms extracted from the corpus, and lots of terms matched against the concepts), the association extraction part needs some improvements, since few associations are extracted from the text corpus, consequently few extracted associations are matched against associations in patterns, and finally the generated ontology is composed of few associations.

In order to measure the reliability of the constructed ontology, experiments have been conducted for evaluating;

- The variation of the number of ODP concepts matched against the extracted terms with respect to the string metric chosen. (See Table 5-1)
- The recall and the precision of the string metrics used by the prototype system. (See Figure 5-2 and Figure 5-3)

Recall and precision are used in the string matching area for evaluating the relevance of the results obtained by the string metrics. According to [21], precision is "*a measure of the proportion of selected items that the system got right*" and recall is "*a measure of the proportion of the target items that the system selected*". They are defined by the following formulas [21]:

$$\mathrm{Re}call = \frac{A}{A+B}$$

$$A = correct\_matches\_retrieved\_by\_the\_system$$

$$B = total\_set\_of\_matches\_retrieved\_by\_the\_system$$

The recall formula adapted to the string metric evaluation task is:

$$\mathrm{Re}call = \frac{|Correctly\_matched\_concepts|}{|Total\_number\_of\_concepts\_that\_should\_have\_been\_mathed|}$$

$$\mathrm{Pr}ecision = \frac{A}{A+C}$$

$$A = correct\_matched\_retrieved\_by\_the\_system$$

$$C = incorrect\_matched\_retrieved\_as\_correct\_by\_the\_system$$

The precision formula adapted to the string metric evaluation task is:

$$\mathrm{Pr}ecision = \frac{|Correctly\_matched\_concepts|}{|Total\_number\_of\_mathed\_concepts|}$$

In [37] the harmonic mean (or F-measure) and the E-measure, are introduced as two ways of combining recall and precision. They are given by this general formula:

$$F-measure(j) = \frac{2*(\mathrm{Re}call(j)*\mathrm{Pr}ecision(j))}{\mathrm{Re}call(j)+\mathrm{Pr}ecision(j)}$$

$$E-measure(j) = 1-\frac{(1+b^2)*(\mathrm{Re}call(j)*\mathrm{Pr}ecision(j))}{b^2*\mathrm{Pr}ecision(j)+\mathrm{Re}call(j)}$$

For the experiment we used;

- A corpus composed of a collection of 10 texts from the software development literature available at the Wikipedia online encyclopaedia[8]. From this corpus a set of 440 terms and 17 associations were extracted by using the information extraction algorithms previously presented in section 4.3.1

- A pattern catalogue composed of 2 ontology design patterns, which had in total, 27 concepts and 26 associations.

- For considering a correct matched, a reference list of 99 matches, created from the text corpus and the ODPs concepts. This reference list includes the exact matches in addition to, the terms that are not exact matches but have the same meaning as the concepts in the ODPs (they can be considered as synonyms).

- A value of 1 for the *b* parameter used in the E-measure formula, so that the F-measure and the E-measure are complementing each other.

- A value of 0.6 for the string matching threshold, which permits to set a starting point for considering an extracted term and a concept as matched.

The succeeding tables and graphs show the results of the matching process of the extracted terms and associations against the concepts and associations in the ODPs.

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

String metric	Concepts in ODP	Matched concept[9]	Eff. Matches[10]	Recall (%)	Precision (%)
JaroWinkler	27	27	13	100.00	48.14
Jaro	27	27	12	100.00	44.44
Jaccard	27	2	2	7.40	100.00
SoftTFIDF	27	15	12	55.55	80.00
TFIDF	27	12	5	44.44	41.66
JaroWinklerTFIDF	27	15	12	55.55	80.00
Level2Jaro	27	27	16	100.00	59.25
Level2JaroWinkler	27	27	16	100.00	59.25
MongeElkan	27	27	8	100.00	29.62
Level2MongeElkan	27	27	17	100.00	62.96

Table 5-1 String metric comparison

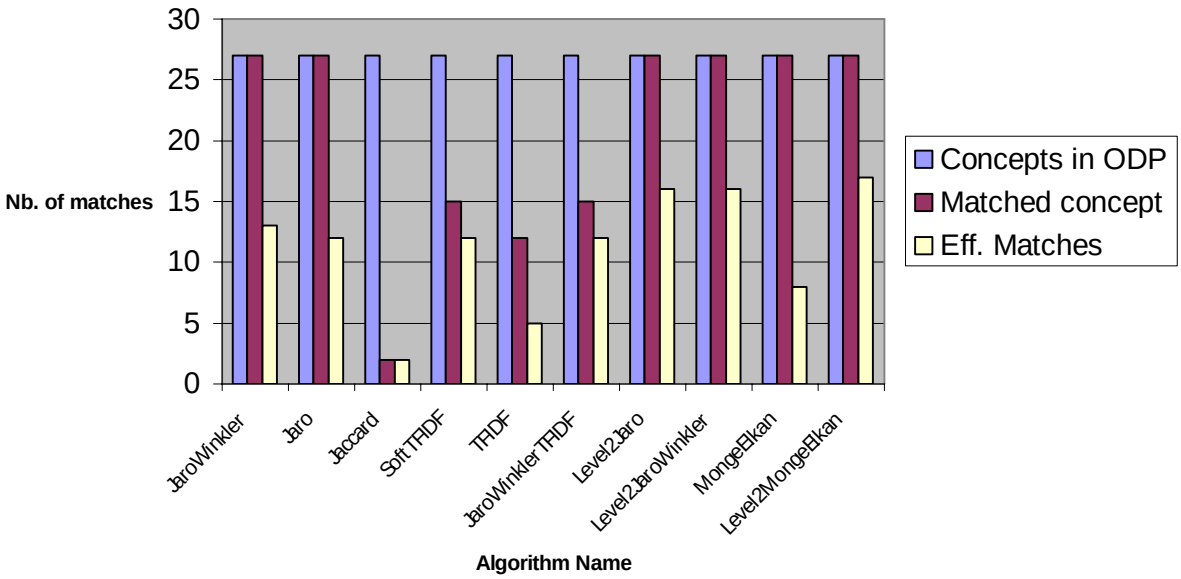

Figure 5-1 Number of concept matched with reference to the string metric

[9] The number of distinct concepts matched by the prototype system
[10] The effective number of concept matched, which are also matched in the reference list

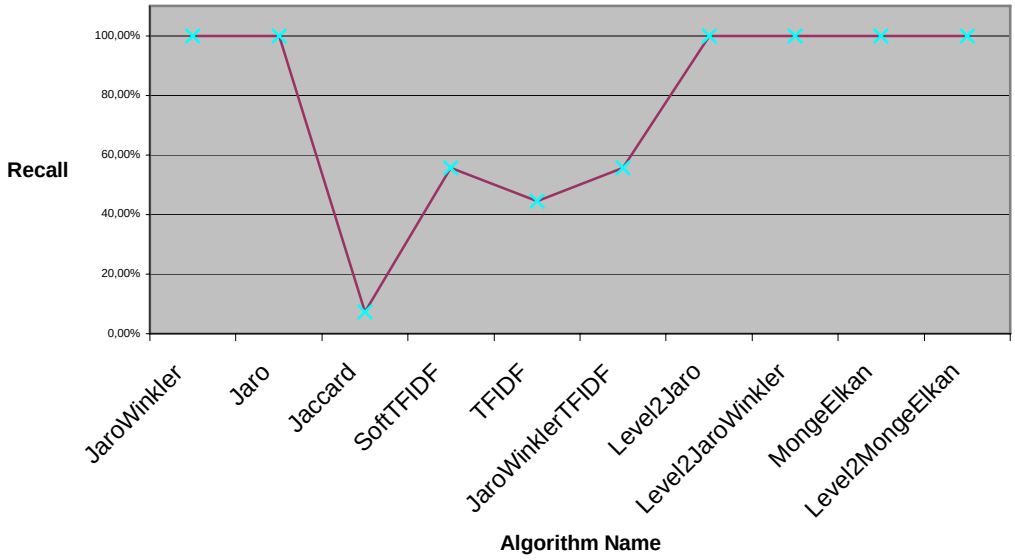

Figure 5-2 Recall evolution with reference to the string metric

Figure 5-2 and Figure 5-3 are graphs representing the values of the recall and precision for different algorithms.

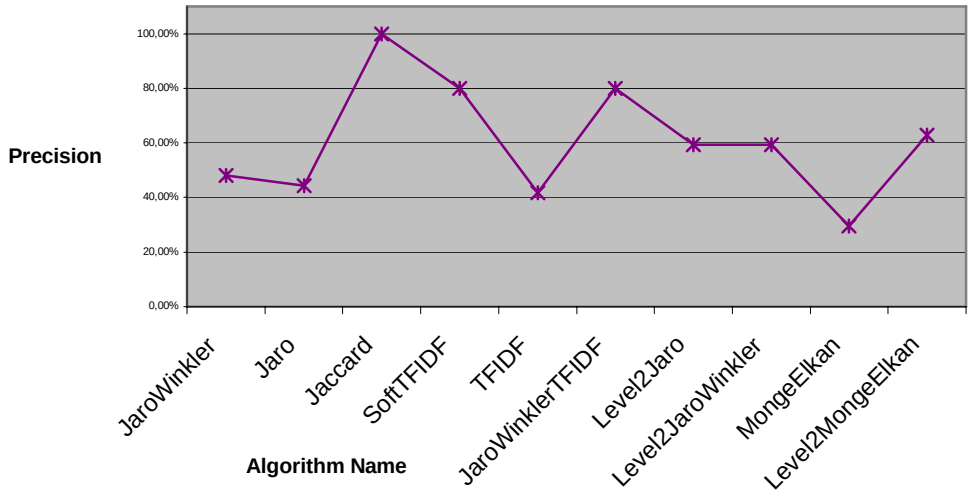

Figure 5-3 Precision evolution with reference to the string metric

According to the different recall and precision values obtained by the algorithms, we can deduce that the reliability of the concepts and associations that are in the generated ontology is closely linked to the string metric used for the terms and concepts matching process. As a result, the choice of a string metric with a high precision value will imply the construction of an ontology having few ODP concepts. On the contrary, a string metric with a high recall value will imply a generated ontology with a lot of ODP concepts. For this experiment, the Jaccard metric obtained precision value of 100% but with a recall of 7.40% which very small compare to other metrics. The more accurate metrics, the ones which have a high precision value with a high recall value, are the SoftTFIDF and JaroWinklerTFIDF metric, they both obtained a precision value of 80.00% and a recall value of 55.55%.

String metric	Recall (%)	Precision (%)	F-measure (%)	E-measure (%)
JaroWinkler	100.00	48.14	65.00	35.00
Jaro	100.00	44.44	61.54	38.46
Jaccard	7.40	100.00	13.79	86.21
SoftTFIDF	55.55	80.00	65.57	34.43
TFIDF	44.44	41.66	43.01	56.99
JaroWinklerTFIDF	55.55	80.00	65.57	34.43
Level2Jaro	100.00	59.25	74.42	25.58
Level2JaroWinkler	100.00	59.25	74.42	25.58
MongeElkan	100.00	29.62	45.71	54.29
Level2MongeElkan	100.00	62.96	77.27	22.73

Table 5-2 Evolution of F-measure and E-measure

The following figure shows the evolution of the F-measure and the E-measure.

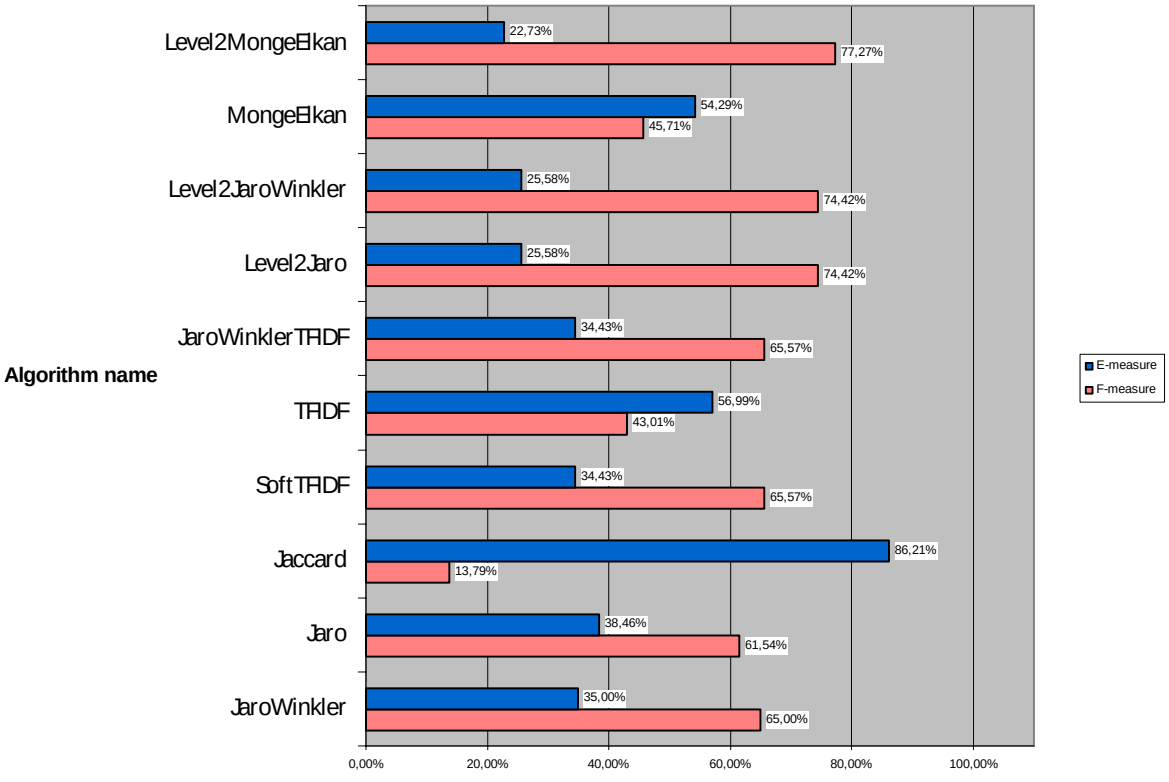

Figure 5-4 Evolution of the F-measure and the E-measure

The following pictures Figure 5-5 shows a generated ontology constructed by the using the previous text corpus and pattern catalogue.

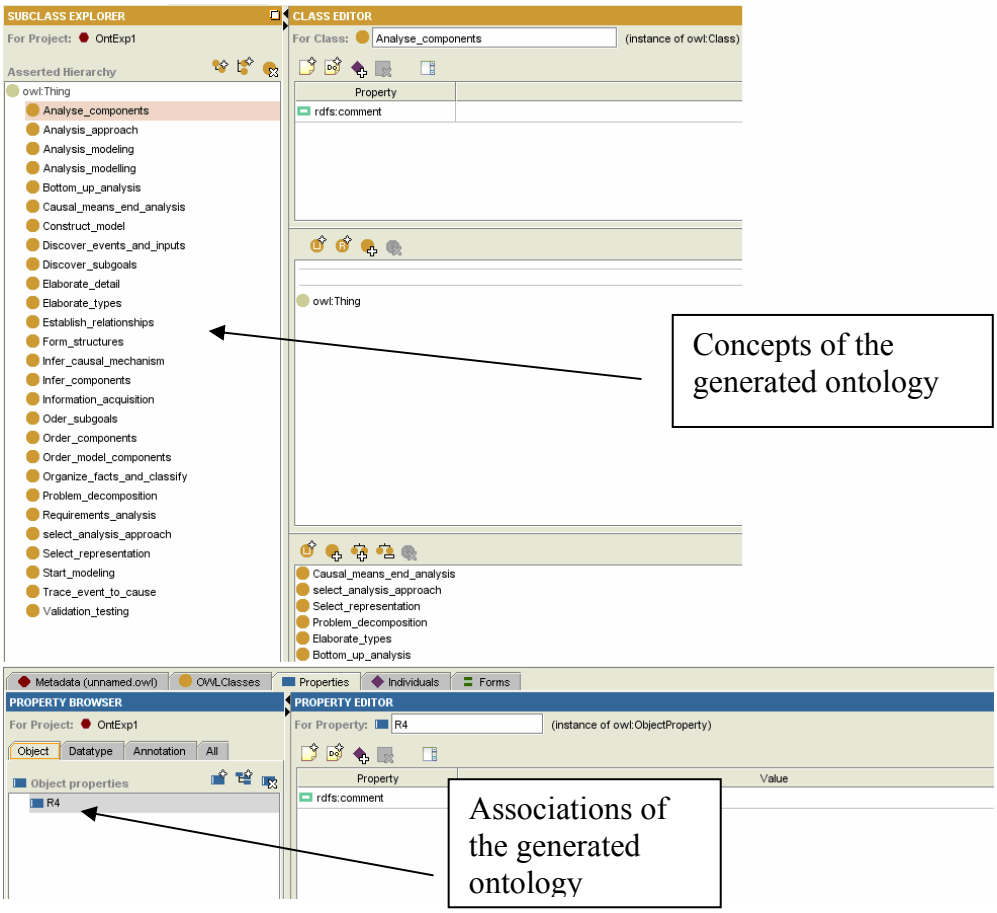

Figure 5-5 Picture of the generated ontology

6 Conclusion and discussions

An ontology can be constructed according to three categories of frameworks; manual, semi-automated or automatic. In this thesis we presented a prototype system for automatic ontology construction based on ontology design patterns and unstructured text. The main steps of the ontology construction framework provided by the prototype system are;

I. extract relevant terms and associations of terms in a text corpus

II. match the previously extracted terms against the concepts and associations contained in the ontology design patterns

III. calculate a matching score that reflects the matching process

IV. select the patterns that have a matching score above a threshold

V. construct an ontology from the matched concepts and associations in the ontology design patterns

The prototype system, allows the use of several algorithms for extracting terms and associations from the text corpus. Although good results were obtained for extracting important terms from a text corpus, the extraction of associations still needs some improvements. Due to the small amount of associations extracted the generated ontology contained very few or no associations.

A choice of several string metrics is provided to perform the matching process of the extracted terms against the ontology design patterns content. According to an experiment conducted on the string metrics provided by the prototype system, the Level2Jaro metric produced the best number of effective matches. This experiment permitted to point out that the choice of the string metric influences the reliability of the concepts contained in the generated ontology. The higher the precision value of the string metric, the more reliable is the generated ontology. For our experiment the Jaccard metric reached a 100% precision rate but the generated ontology was composed of very few concepts. The more accurate metrics for our experiment were the hybrid metrics SoftTFIDF and JaroWinklerTFIDF, both obtained a precision value of 80.00% and a recall value of 55.55%.

With reference to a comparative study between a manual ontology construction framework, which is based on cookbook-like instructions, and the ontology construction framework used by the prototype system, a conclusion concerning the advantages and disadvantages of automated construction framework have verified;

- Some important concepts in an ODP may not appear in the automatically generated ontology since they might have not been matched against any extracted terms. This will imply a generated ontology which contains concepts less important than the initial ontology design patterns.

For calculating a score that reflects the matching process of the unstructured text against the ontology design patterns, the prototype system provides a formula based on linear combination, and an "Automated weights values" formula. The difference between those formulas is that the linear combination allows the user to set the parameters for calculating the score, whereas the other one automatically computes values for the score parameters by using the amount of concepts and associations contained in the ontology design patterns. Finally, a threshold value for the matching score permits to generate an ontology, by using the best matched ontology design patterns and hence the reliability of the concepts and associations in the generated ontology.

In order to solve the problem of few associations in the generated ontology, future work will permit to adapt new algorithms for association extraction into the extraction module of the prototype system. In this thesis we only presented a framework based on ontology design patterns and unstructured texts. However, some ontology construction frameworks that support ontology search engine are presented in the literature.

As a result an evolution of the construction framework could be to use ontology search engines, such as OntoSearch or Swoogle, to provide the pattern catalogue with online ontologies, instead of constructing the ontology design patterns. For instance, an ontology search module can be adapted to the prototype system to look for the ontology patterns from some keywords, then select the relevant ontologies for the construction process (update them if necessary), and finally perform the different steps of the construction process as presented in the beginning of this chapter.

Another evolution for the prototype system could be to support synonyms for the concepts in the ontology design patterns and add them to the generated ontology. For this purpose, a suggestion could be to use while constructing the ontology design patterns, the "Annotations" section from the Class-editor of Protégé-OWL framework, in order to store the synonyms for each concepts. Then add for each synonym a new "Annotation value" by using, "rdfs:comment" as property for the annotation and the synonym label as "value" for the comment. As a result all the synonyms would be stored in the ontology design patterns file, and the following step would be to adapt the prototype system, so that it can get the synonyms labels from this file, and then include the correct synonyms for the concepts of the generated ontology.

7 References

[1]. Blomqvist, E. (2005) Fully Automatic Construction of Enterprise Ontologies Using Design Patterns: Initial Method and First Experiences. In Proceedings of OTM 2005 Conferences, Ontologies, DataBases, and Applications of Semantics (ODBASE), Agia Napa, Cyprus, Oct 31- Nov 4, 2005.

[2]. Fernndez, L.M. (1999): Overview of Methodologies for Building Ontologies. In Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5)

[3]. Alani, H., Kim, S., Millard, D. E., Weal, M. J., Lewis, P. H., Hall, W. Shadbolt, N. R. (2003) Automatic Extraction of Knowledge from Web Documents. In Proceedings of 2nd International Semantic Web Conference - Workshop on Human Language Technology for the Semantic Web abd Web Services, Sanibel Island, Florida, USA

[4]. Gal, A. Modica, G. Jamil, H. (2004) OntoBuilder : Fully Automatic Extraction and Consolidation of Ontologies from Web Sources. In Proceedings of the 20th International Conference on Data Engineering p.853, March 30-April 02, 2004

[5]. Noy, N.F. and McGuiness D.L. (2001), Ontology Development 101: A Guide to Creating Your First Ontology by Noy, N.F. and McGuinness, D.L. SMI technical report SMI-2001-0880, Stanford University

[6]. Blomqvist, E. Öhgren, A. Sandkuhl, K. (2006) Ontology Construction in an Enterprise Context: Comparing and Evaluating two Approaches. Proceedings of 8th International Conference on Enterprise Information Systems, Paphos, Cyprus, May 2006.

[7]. Gruber, T. R. (1993) .A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5(2), 199-220

[8]. Blomqvist, E. Sandkuhl, K. (2005) Patterns in Ontology Engineering: Classification of Ontology Patterns. ICEIS (3) In Proceedings of the Seventh International Conference on Enterprise Information Systems, Miami, USA, May 25-28, 2005 p413-416

[9]. Lenzerini, M. Milano, D. Poggi, A. State of the Art Report 1.Ontology Representation & Reasoning. Available at http://www.dsi.uniroma1.it/~estrinfo/1%20Ontology%20representation.pdf (Acc 2006-12-31).

[10]. Asanee Kawtrakul, Mukda Suktarachan , Aurawan Imsombut.(2004). Automatic Thai Ontology construction and Maintenance System. In Proceedings of OntoLex Workshop on LREC, Lisbon, Portugal.

[11]. Blaschke C.Valencia A (2002). Automatic Ontology Construction from the Literature. In Proceedings of Genome Informatics Ser Workshop Genome Informatics 13 p201–213

[12]. Ding, Y., Foo, S. (2002). Ontology research and development, Part 1 - A review of ontology generation. Journal of Information Science., 28(2), 123-136

[13]. Hyunjang Kong, Myunggwon Hwang, Pankoo Kim (2006). Design of the Automatic Ontology Building System about the Specific Domain Knowledge. In Proceedings of 8th International Conference on Advanced Communication Technology, ICACT 2006. Volume: 2, page(s): 4 pp.

[14]. Alani H. (2006) Ontology Construction from Online Ontologies. In Proceedings of 15th World Wide Web Conference, Edinburgh, Scotland

[15]. Hogeboom, M. Fuhua Lin. Esmahi, L. Chunsheng Yang. (2005) Constructing Knowledge Bases for E-Learning Using Protégé 2000 and Web Services. In Proceedings of the 19th International Conference on Advanced Information Networking and Applications, AINA 2005.

[16]. Askar, K., Vemuri, S.; Siril, Y., Dougherty , M. (2004) Plug-in for Protégé 2000 witch supports Sesame. Working Papers in Transport, Tourism and Information Technology, 3rd International Semantic Web Conference, Hiroshima, Japan, 2004.

[17]. Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, Mark A. Musen. (2004) The Protégé OWL Plug-in: An Open Development Environment for Semantic Web Applications. In Proceedings of the 3rd International Semantic Web Conference - ISWC 2004, Hiroshima, Japan - An architectural overview for developers and decision-makersBlomqvist, E. Öhgren, A. Sandkuhl, K. (2006) Ontology Construction in an Enterprise Context: Comparing and Evaluating two Approaches. Proceedings of 8th International Conference on Enterprise Information Systems, Paphos, Cyprus, May 2006.

[18]. Noy, N.F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R.W., Musen, M.A.(2001) Creating Semantic Web contents with Protege-2000. Intelligent Systems, IEEE Volume 16, Issue 2, Mar-Apr 2001 Page(s):60 – 71

[19]. M. Horridge, H. Knublauch, A. Rector (2004). A Practical Guide to Building OWL Ontologies Using the Protégé-OWL Plugin and CO-ODE Tools. University of Manchester. Available at: http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf (Acc 2006-12-29)

[20]. Michael Krauthammer, Goran Nenadić (2004). Term Identification in the Biomedical Literature. Journal of Biomedical Informatics archive Volume 37 , Issue 6 December 2004)

[21]. Alexandre Maedche, (2002) Ontology learning for the semantic web. Kluwer Academic p 97-98, p174-175

[22]. Cimiano, P. Völker, J. (2005) Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. In Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB), volume 3513 of Lecture Notes in Computer Science, pp. 227-238

[23]. Egorov, S. Yuryev, A. Daraselia, N. (2004) A Simple and Practical Dictionary-based Approach for Identification of Proteins in Medline. Journal of the American Medical Informatics Association Volume 11 Number 3 May / Jun 2004

[24]. José Iria. (2005) T-Rex: A Flexible Relation Extraction Framework. In Proceedings of the 8th Annual Colloquium for the UK Special Interest Group for Computational Linguistics (CLUK'05), Manchester.

[25]. Sam Chapman (2006) Similarity Metrics http://www.dcs.shef.ac.uk/%7Esam/stringmetrics.html (Acc. 2006-10-29)

[26]. William E. Yancey (2005) Evaluating String Comparator Performance for Record Linkage. Statistical Research Division. U.S. Census Bureau. Washington, DC 20233. Report Issued: June 13, 2005

[27]. William W. Cohen, Pradeep Ravikumar & Stephen Fienberg (2003): A Comparison of String Distance Metrics for Name-Matching Tasks. In

Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web, Acapulco, Mexico 73-78

[28]. Gamma, E. Helm, R. Johnson, R. Vlissides, J. (1993) Design Patterns: Abstraction and Reuse of Object-Oriented Design. In Proceedings of the 7th European Conference on Object-Oriented Programming Lecture Notes in Computer Science Vol. 707 pp 406 - 431

[29]. Aldo Gangemi (2005) Ontology Design Patterns for Semantic Web Content. The Semantic Web (ISWC 2005) 4th international semantic web conference Galway, Ireland, November 6-10, 2005

[30]. Mikel Egana Aranguren (2005) Phd Thesis Report: Ontology Design Patterns for the Formalization of Biological Ontologies. Available at http://gong.man.ac.uk/docs/MPhilThesis.pdf (visited on 2006-10-29)

[31]. Reich, J.R. (1999). Ontological Design Patterns for the Integration of Molecular Biological Information. In Proceedings of the German Conference on Bioinformatics GCB'99 (pp.156-166), 4-6.October, Hannover, Germany.

[32]. Sommerville I. (2004) Software Engineering (7th edition). Addison Wesley; 7th Edition edition (3 Jun 2004)

[33]. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications

[34]. Scenario Description Language Q http://www.ai.soc.i.kyoto-u.ac.jp/Q/index_e.htm (Acc 2006-11-17)

[35]. IEEE Std 1016-1998 IEEE Recommended Practice for Software Design Descriptions

[36]. Jena 2 Ontology API, http://jena.sourceforge.net/ontology/ (Acc 2006-10-17)

[37]. Baeza-Yates, R. Ribeiro-Neto, B. (1999) Modern Information Retrieval. New York: ACM Press Series/Addison Wesley, 1999. Chapter 4.

8 Appendix

Appendix 1 Examples of string matching by using different string metrics

Appendix 2 Software Requirement Specifications Document for the Automatic Ontology Construction Prototype System

Appendix 3 Software Design Description Document for the Automatic Ontology Construction Prototype System

Appendix 1: Examples of string matching by using different string metrics

The matching score presented in the following table have been computed by using the string metrics proposed by the tool SecondString[11]. The matching score is included in the interval [0, 1].

String metric	String A	String B	Matching Score
Jaro	Testing	Validation_testing	0.0
Jaro	Inform	Information	0.84
Jaro	Elaborate	Elaborate	1.0
JaroWinkler	Testing	Validation_testing	0.0
JaroWinkler	Inform	Information	0.90
JaroWinkler	Elaborate	Elaborate	1.0
Jaccard	Testing	Validation_testing	0.5
Jaccard	Inform	Information	0.0
Jaccard	Elaborate	Elaborate	1.0
SoftTFIDF	Testing	Validation_testing	0.70
SoftTFIDF	Inform	Information	0.90
SoftTFIDF	Elaborate	Elaborate	1.0
Level2JaroWinkler	Testing	Validation_testing	1.0
Level2JaroWinkler	Inform	Information	0.90
Level2JaroWinkler	Elaborate	Elaborate	1.0
Level2Jaro	Testing	Validation_testing	1.0
Level2Jaro	Inform	Information	0.84
Level2Jaro	Elaborate	Elaborate	1.0
TFIDF	Testing	Validation_testing	0.70
TFIDF	Inform	Information	0.0
TFIDF	Elaborate	Elaborate	1.0

[11] http://secondstring.sourceforge.net

Software Requirement Specifications Document for the Automatic Ontology Construction Prototype System

Student: Ludovic Jean-Louis
Teacher: Eva Blomqvist

Document Evolution		
Indices	Date	Comments
A	06/10/29	Initial version
B	07/01/22	Update on the requirements SRE03 to SRE28
C	07/01/25	Update on the requirements numbering and the section 2.2.1.

Table of contents

List of figures

1. Introduction

1.1 Purpose

The present document aims at defining all the requirements in relation to the development of the prototype system for automatic ontology construction.
The prototype system will assist researcher in the task of building ontology. The prototype will enable the researchers to automatically construct ontologies from a text corpus and one or more ontology design pattern. The prototype will enable optimizing of the time required to build ontologies. The prototype is based on the general framework for automatic ontology construction developed by the Information Engineering research group at Jönköping University. The general framework for automatic ontology construction is presented in [1].

1.2 Scope

The prototype system will help in validating the previously mentioned general framework for automatic ontology building. A succeeding goal is to avoid the use of several tools for different parts of the ontology construction process. This goal can be met, by integrating those tools in a plug-in for an existing ontology building environment. Another consecutive goal is to reduce the time and effort required to build ontologies through the use of an automated method.

Even some software have already been implemented for automatic ontology construction [2] [3], they are based on web sources documents and do not provide the use of ontology design patterns to construct ontologies. Those ontologies design patterns are especially useful since similarities can be made with enterprise modelling and the aim of the prototype is to construct enterprise ontologies.

1.3 Definitions, acronyms, and abbreviations

- **AOC**: Automatic ontology construction
- **ODP**: Ontology design pattern
- **SRE##**: Essential Software Requirement number ##
- **SRC##**: Conditional Software Requirement number ##
- **SRO##**: Optional Software Requirement number ##
- **Text corpus**: "*A large and structured set of texts*" (www.wikipedia.org)
- **Term**: We consider a term as a group of words that possibly refers to an explicit concept in a text corpus.
- **Concept**: We consider a concept as "*an abstract idea or a mental symbol, typically associated with a corresponding representation in language*" (www.wikipedia.org)
- **Association/relation**: We consider an association or relation as a link between two concepts or between two terms.

1.4 References

[1]. Blomqvist, E. (2005) Fully Automatic Construction of Enterprise Ontologies Using Design Patterns: Initial Method and First Experiences. Lecture Notes in Computer Science p1314-1329.
[2]. Alani, H., Kim, S., Millard, D. E., Weal, M. J., Lewis, P. H., Hall, W. Shadbolt, N. R. (2003) Automatic Extraction of Knowledge from Web Documents. In Proceedings of

2nd International Semantic Web Conference - Workshop on Human Language Technology for the Semantic Web abd Web Services, Sanibel Island, Florida, USA.
[3]. Gal, A. Modica, G. Jamil, H. (2004) OntoBuilder : Fully Automatic Extraction and Consolidation of Ontologies from Web Sources. In Data Engineering, 2004 Proceedings 20th International Conference.
[4]. William W. Cohen, Pradeep Ravikumar & Stephen Fienberg (2003): A Comparison of String Distance Metrics for Name-Matching Tasks. In IIWeb 2003: 73-78.
[5]. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications.
[6]. M. Horridge, H. Knublauch, A. Rector (2004). A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools. University of Manchester.

1.5 Overview

This document describes the requirements for the AOC prototype system. It is divided into 3 parts. Part 2 gives a general description of, the AOC prototype, the main functions that are expected from the prototype, the user profile required to use the prototype system and finally, the implementation constraints of the AOC prototype. In part 3 all the system requirements are detailed and presented according to the organization by feature.

2. Overall description

2.1 Product perspective

The prototype system cannot be considered as a totally autonomous system since, it should interact with other tools for some tasks, which are included the AOC process, such as; extract terms and associations, match terms and association list against the concepts and relations in the ontology design patterns (ODPs), edit and create ontology patterns, edit and enrich ontology with synonyms, concepts and associations.
The following diagrams show the links between the prototype system and the related tools used for the external tasks.

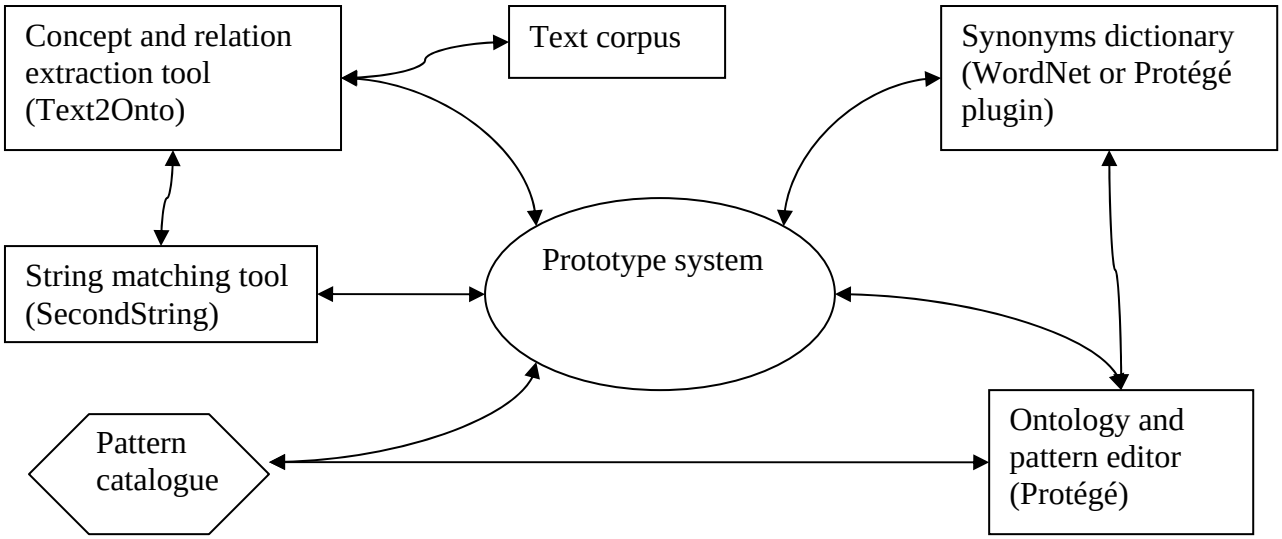

Figure 2-1 Interaction of the prototype system with external tools

2.1.1 User interfaces

The AOC prototype system does not have to interact with different categories of users. It should be accessible through a friendly and easy to use graphical interface. Different windows layouts shall be presented to the user depending on if he is constructing ODPs, if he is extracting terms and associations from a text corpus, if he is compiling an ontology, and finally if he is setting values for the different parameters involved in the ontology building process.

2.1.2 Hardware interfaces

No specific characteristics are required between the AOC prototype and the hardware components since the AOC prototype is intended for use on a local computer.

2.1.3 Software interfaces

2.1.3.1 Text2Onto interface

As described in the AOC framework in [1], a list of terms and associations shall be extract from a text corpus, and afterward those two lists are matched against the ODPs contained in the pattern catalogue. Therefore Text2Onto[1] shall be used in order to extract those terms and associations. The prototype should be responsible for, storing the terms in text files, and allow this text files to be reuse for matching extracted terms and associations against the concepts and associations in one or several ODPs.

2.1.3.2 SecondString interface

The AOC framework in [1] suggests that the terms extracted from the text corpus shall be matched against the concepts and associations in the ODPs. Therefore the SecondString[2] tool shall be used for matching the extracted terms from the text corpus against the concepts and association contained in all the ODPs that are in the pattern catalogue.

2.1.3.3 Protégé interface

Protégé[3] is an ontology editor that allows the use of many plugins and the OWL language for ontology development. As a result the prototype system shall interact with Protégé in order to reuse Protégé facilities for ontology edition, and also reuse the facilities that are available through the Protégé plugins (for instance ontology enrichment with synonyms, annotate ontology with terms, etc…). It is also suggested that the Protégé environment shall be used as an editor for the ontology design patterns.

2.1.3.4 WordNet interface

WordNet[4] lexicon shall interact with the AOC prototype in order to enrich the ontology with synonyms or suggest synonyms to the user while he is constructing an ontology design pattern. Two solutions are possible for this interaction either the prototype uses WordNet facilities directly or the prototype shall use WordNet via a plugin for Protégé.

2.1.4 Communications interfaces

No specific interfaces to communications are required between the AOC prototype and the hardware components since the AOC prototype does not intent to work over a network.

[1] http://ontoware.org/projects/text2onto/
[2] http://secondstring.sourceforge.net/
[3] http://protege.stanford.edu/
[4] http://wordnet.princeton.edu/

2.1.5 Memory constraints

No specific storage server is required for saving the ontology design patterns and the automatically constructed ontologies. Capacity is limited only by the amount of disk space available.

2.1.6 Operations

Several modes can be identified for the system, in this section they are divided in modes that uses external components and modes that perform internal tasks for the prototype system.
The identified modes of operations are:

- Ontology design patterns mode
- Terms extraction mode
- Terms matching mode
- Association generation mode
- Association conversion mode
- Association matching mode
- Pattern evaluation mode
- Ontology construction mode

2.1.6.1 Ontology design pattern mode

The ontology design pattern mode uses external components from the Protégé environment. It consists of several phases,

a) Add concepts to ODP
b) Add associations to ODP
c) Add synonyms to concepts
d) Update concepts/update associations
e) Save ODP

2.1.6.2 Terms extraction mode

Terms extraction mode uses external components from Text2Onto. It consists of several phases:

a) Set a text corpus for terms extraction
b) Extract and save terms from the text corpus

2.1.6.3 Terms matching mode

Terms matching mode uses external component from SecondString. It consists of several phases:

a) Generate a list of all concepts used in the ontology design pattern
b) Match extracted terms from the list saved in step b) of section 2.1.6.2 against the concepts from the previous step a)
c) Save the list of terms and concepts matched

2.1.6.4 Association generation mode

Association generation mode uses external components from Text2Onto. It consists of several phases:

a) Generate possible associations from the terms contained in the text corpus
b) Save the list of possible associations generated in step a)

2.1.6.5 Association conversion mode

Association extraction mode consists of several internally performed phases:

a) Generate a list of the associations that are in the ODPs

b) For all the associations saved in step b) of section 2.1.6.4, translate the terms in the possible associations to the labels of the concepts of the patterns instead of the extracted terms using the result of step of c) of 2.1.6.3

2.1.6.6 Association matching mode

Association matching mode consists of several internally performed phases:

a) Match all the possible associations extracted from the text corpus obtained in step b) of section 2.1.6.4 against the associations in the ODPs obtained in step b) of section 2.1.6.5
b) Save the associations that have been successfully matched.

2.1.6.7 Pattern evaluation mode

Pattern evaluation mode consists of several internally performed phases:

a) Compute the matching score based on the amount of terms and associations matched for each ODP
b) Compare the matching score of each ODP against the threshold
c) If the ODP score is above the threshold set by the user, save the pattern name together with the related terms and matched associations

2.1.6.8 Ontology construction

Ontology construction mode consists of several phases. The phases of this mode are:

a) For each ODP saved in step g) of section 2.1.6.7, add the concepts and related synonyms in the new ontology
b) If a concept is already in the ontology, add all new concept synonyms to the ontology
c) Add relations between the concepts
d) Redo b) and c) until all the matched terms have been used

2.1.7 Site adaptation requirements

No site adaptation is required for the integration of the AOC prototype system.

2.2 Product functions

The first version of the AOC prototype system will enable user to use the functions listed in the following sub-section (2.2.1). Other functionalities will be added in other versions of the prototype.

2.2.1 Summary of the major functions

The major functions of the AOC prototype system are listed below:

- Construction of ontology design pattern (2.2.1.1)
- Extract terms in a text corpus (2.2.1.2)
- Match extracted terms against the concepts in ontology design patterns (2.2.1.3)
- Extract associations in a text corpus (2.2.1.4)
- Match extracted associations against the relations in ontology design patterns (2.2.1.5)
- Compute a score based on the amount of terms and relations matched (2.2.1.6)
- Set a threshold for ontology design pattern selection (2.2.1.7)
- Store the accepted ontology design pattern with associated extracted terms and relations (2.2.1.8)
- Update an ontology design pattern (2.2.1.9)
- Build ontology with the selected ontology design pattern and the matched terms and associations (2.2.1.10)

2.2.1.1 Construction of ontology design pattern

This function aims at constructing an ODP via an ontology editor.

Description of the function:

The user starts by entering the name of the ODP, a short description of the ODP purpose. The user then chooses to add concepts to the ODP. For each concept created a list of synonyms is proposed, the user can add one or several synonyms from this list by selecting the synonym and validating. If not necessary to add a synonym the user can select cancel. In case the user does not want to add a synonym from the proposed list, or additionally to the synonyms added from the proposed list, the user should be able to add his owns synonyms. When the concepts are added to the ODP, the user has the possibility to add relationships among them and also define the characteristic of the association (functional association, inverse functional, etc…).

2.2.1.2 Extract terms in a text corpus

This function aims at extracting the terms that express ontology concepts in a text corpus. This extraction part concerns the first step of the ontology construction process described in [1].

Description of the function:

The researcher enters parameters required by the extraction tool Text2Onto as input and validates his choice to start the extraction process. Thus the system automatically extracts the terms and stores them in a term list corresponding to a text file. This text file is the output of this functionality and will be reused later in the process, to be matched against the concepts in the ODPs. An example of the expected content of this file could be as presented in the following Figure 2-2.

Term Label
Term1
Term2
…
TermN

Figure 2-2 Example of extracted terms list

2.2.1.3 Match extracted terms against the concepts in ontology design patterns

This function aims at contrasting the terms extracted from the text corpus and the concept in the ODPs, in order to evaluate the number of terms that are in both the pattern and the text corpus.

Description of the function:

This functionality can be used when the pattern catalogue contains at least one ODP and the text corpus contains at least one text. Before starting this functionality the user has to select; i) the path to at least one text file to add to the text corpus, ii) the path to at least one ODP to add to the pattern catalogue, iii) the string metric and the corresponding parameters (threshold for string comparison). The system should propose several matching algorithms as a list with the metric names (example of SecondString [4] TFIDF, Jaro-Winkler, SoftTFIDF, etc…). Once the user enters all these parameters he validates his choice and the system executes the matching process according to the selected parameters (string metric, term list, ontology pattern). At the end of this functionality, a new text file is created in which, the list of successfully matched terms and concepts are written together with the pattern name and the string matching score. An example of the expected content of this file could be as presented in Figure 2-3.

Term Label, Concept Label, Pattern Name, Matching Score

Term1, Concept1, Pattern1, 0.8
Term2, Concept4, Pattern2, 0.6
…
TermN, Concepti, Patternj, 1.0

Figure 2-3 Example of a list of matched terms and concepts

2.2.1.4 Extract associations in a text corpus

This functionality aims at extracting the possible associations in a text corpus. This extraction part concerns the third step of the ontology construction process described in [1].

Description of the function:

Two methods are proposed for description this function:

Method 1:

The system then uses the list of matched terms and concepts (ExTermsList) resulting from the function "Extract terms in a text corpus" see Figure 2-3, to select the terms already matched to the pattern at hand. The system then uses this list to extract associations from the text corpus. Each resulting extracted associations (one extracted association is named ExAss#) is written in a new list that is a list of "Possible associations", since they are associations in the text corpus but not necessary in the pattern. Finally the list of possible associations is saved in the text file selected by the user.

Method 2:

When using this method, the functionality can be used when the text corpus contains at least one text. The researcher enters parameters required by the extraction tool Text2Onto (name of the association extraction algorithm) as input and validates his choice to start the extraction process. Thus the system automatically extracts the associations from the text corpus and stores them in a list corresponding to a text file. This text file is the output of this functionality and will be reused later in the process for converting the association extracted to associations of concept labels. An example of the expected content of this file could be as presented in the following Figure 2-4.

Properties, Property Name, Domain, Domain Label, Range, Range Label,

Properties, A1, Domain, Term1, Range, Term2,
Properties, A2, Domain, Term3, Range, Term4,
…
Properties, Ai, Domain, Termi, Range, Termj,

Figure 2-4 Example of a list extracted associations

2.2.1.5 Match extracted associations against the relations in ontology design patterns

This functionality aims at matching the extracted associations from the text corpus to the associations contained in the ontology design patterns.

Description of the function:

For each pattern the system needs to create a list of all the "Associations in the ODPs" as shown in Figure 2-6. For each association in the "Possible association" list generated by the function in section 2.2.1.4 (see Figure 2-4), the system uses the list of terms and concept matched (see Figure 2-5) to look for to which concepts that a terms in an extracted association matches in the ODPs. And then creates a list of "Converted associations", which similar to the list of "Possible association", by changing each single term by the correct concept in the ODP as shown in Figure 2-7. Finally, the "Converted associations" list and the "Associations in the

ODPs" list are matched against each other. The matching result list is written in a new text file as the list of matched associations together with the name of the pattern where the associations where from. This text file is the output of this functionality. An example of the content of the text file could be as presented in Figure 2-8.

Term Label, Concept Label, Pattern Name, Matching Score

Term1, Concept3, 0.8
Term2, Concept4, 0.86
Term3, Concept1, 0.90
Term4, Concept2, 0.75

Figure 2-5 List of terms and concepts successfully matched

Properties, Property Name, Domain, Domain Label, Range, Range Label, Pattern Name

Properties, A1, Domain, Concept1, Range, Concept2, Pattern1
Properties, A2, Domain, Concept3, Range, Concept4, Pattern1
…
Properties, Ai, Domain, Concepti, Range, Conceptj, Patterni

Figure 2-6 List of Associations in the ontology design patterns

Convert properties, Property Name, Domain, Domain Label, Range, Range Label, Pattern Name

Convert properties, A1, Domain, Concept3, Range, Concept4, Pattern1
Convert properties, A2, Domain, Concept1, Range, Concept2, Pattern1

Figure 2-7 Example of a list of converted associations

Matched properties, Property Name, Domain, Domain Label, Range, Range Label, Pattern Name

Matched properties, A2, Domain, Concept3, Range, Concept4, Pattern1
Matched properties, A1, Domain, Concept1, Range, Concept2, Pattern1
…
Matched properties, Ai, Domain, ConceptN, Range, Concepti, Patternj

Figure 2-8 Example of a list of matched associations

2.2.1.6 Compute a score based on the amount of concepts and relations matched

This functionality aims at quantifying the likelihood of a text corpus and an ODP by computing a score based on the amount of terms and associations in the text corpus that match successfully against the ODP.

Description of the function:

The functionality allows the user to choose one formula among a list of different formulas for score computation. The user should also have the possibility to easily add a new formula to compute the score, by changing the source code. A general way to compute the score could be based on a linear combination of the percentage of terms and associations matched.

For instance, let us consider "*a*" and "*b*" two scalars such that "*a*" and "*b*" are strictly positive, and let us consider "*%Terms_matched*" and "*%Associations_matched*" two variables

representing respectively, the percentage of terms matched and the percentage of associations matched. The score can be computed using:

$$Score = a * \%Terms\_matched + b * \%Association\_matched$$

Suggested values for “a” and “b” could be:

$$a = 1/2 * \left[(Amount\_Of\_Concept\_In\_Pattern)/(Amount\_Of\_Association\_In\_Pattern)\right]$$

$$b = 1/2 * \left[(Amount\_Of\_Association\_In\_Pattern)/(Amount\_Of\_Concept\_In\_Pattern)\right]$$

The user should be able to enter his own values for “*a*” and “*b*” or select the previous suggested values.

2.2.1.7 Set a threshold for ontology design pattern selection

Once the scores for all the different ODPs and a certain text corpus have been computed, the most reliable ODPs to build the ontology automatically should be selected. As a result a threshold will be used to keep all the ODPs that exceed a definite score. This score is computed for each ODP while using the previous functionality “Compute a score based on the amount of concepts and relations matched”. The ODPs having a score below this threshold should not be considered by the system when compiling the ontology. Others should be saved together with their related list of terms and associations.

Description of the functionality:

This functionality is used after the ontology design patterns and a text corpus have already been set up in the pattern catalogue. The user chooses to set a limit for pattern scores and enters the threshold as a number. The system then saves the selected ODPs (ODPs that have a score above the threshold), the successfully matched terms and relations together so that they can be used to build the ontology. Additional information concerning the matching process is also saved; the name of the string matching algorithm used for the matching process, the formula used to compute the score of the ODP. If several ODPs are selected the system should save all of them.

2.2.1.8 Store the accepted ontology design patterns with associated extracted terms and relations

Once the selection of the ODPs have been made after the setting of the threshold, the system should keep track of the matching operations, i.e. which patterns have been matched with which text corpus (extracted terms and relations), what was the score of the matching process and, which method was used for the matching process. Those selected ODPs and extracted terms will be used afterward for building automatically the ontology.

Description of the functionality:

This functionality operates when the user sets the threshold for the ODP selection. The prototype system must store a link between the pattern file names (only for the patterns that are above the selected threshold), the list of matched terms and concepts, the list of matched associations, the name of the string metric used for the matching process, the name of the score formula used to compute the score of the ODP, and finally the computed score for the ODP.

2.2.1.9 Update an ontology design pattern

The aim oh this functionality is to permit the researcher to edit/complete/update an existing ODP.

Description of the functionality:

The user starts the Protégé environment and selects to open the ODP. A list with the ODP names is thus presented to him. The user chooses the ODP he wants to edit and validates his choice. As a result the ontology design pattern appears in the Protégé environment and the user can modify it according to the facilities offered via Protégé. The user should also be able

to update the synonyms in the ODP. When all changes have been made on the ODP, the user can save the changes by choosing the save facility of the Protégé environment.

2.2.1.10 Build ontology with the selected ontology design patterns and the matched terms and associations

After, extracting the terms and relations from the text corpus, selecting the ODPs that are efficient to build the ontology, the final stage of the process is to build the ontology automatically. One method to build the ontology is described in [1]. Once the user has set the ODP threshold a number of ODPs are either removed or accepted for the ontology building process. The accepted ones will be used for this functionality as input, together with the extracted terms and associations. The terms and associations used for this process are those ones that have been successfully matched against the concepts and associations in the ODPs.

Description of the functionality:

This functionality operates after that the matching score of each ODP have been compared to the ODP threshold, as a result only the ODPs having a score higher than the threshold are kept for the construction process. This process permits the user to choose one method to compile the generated ontology (one AOC method is presented in [1]), set heuristics for the construction. Also the system proposes to enter a name and a location for saving the new ontology. The system builds the ontology according to the selected method and save its description in an OWL file.

2.3 User characteristics

The prototype system will be used primarily by researchers of the Information Engineering research group at Jönköping University. Secondary users may be any person familiar with the automatic ontology building process presented in [1]. Secondary user may also be familiar with string matching tools, ontology editor and information extraction tools.

2.4 Constraints

The prototype system should be developed in Java since this is the most common programming language of the existing tools used for terms extraction.

The prototype system should be implemented as a plug-in for the Protégé environment.
In the AOC framework presented in [1] the ontology design pattern are equivalent to ontology since they are composed of concepts, associations, synonyms, and finally a set of constraints that can be applied to the association. Therefore the prototype shall permit to construct both the ODPs and the ontology, using the functionalities of the Protégé-OWL framework. The prototype system shall also permit to apply all the restriction conditions that apply on an ontology design pattern, to all the ontologies that are generated from this ODP.

Some of the Protégé-OWL functionalities that can be used for both the ODP, and the ontology construction, are listed below, other functionalities can be found in [6]:

- Value partitions; allow creating a list of concepts, and applying condition on the concepts of the partition.
- Restriction matrix; allow creating existential restriction on a concept or a group of concepts.
- Create class; permit to add one concept to ontology
- Disjoint classes; allow specifying that concept are disjoint from each other
- Properties; allow creating associations between two concepts of a pattern or an ontology

The prototype will check for, duplicate ODPs in the pattern catalogue, for duplicate text files in the corpus and will not allow these.

The prototype system does not need to insure confidentiality of data since we do not intent to build an ontology-based application, but the ontology itself. As a result there will not be levels of security and user profiles to use the prototype system. Consequently the prototype will be accessed without user ID and password.

2.5 Assumptions and dependencies

The prototype system should be developed for Windows XP platforms, including a Java virtual machine. Recommended Java Runtime Environment is J2SE Runtime Environment 5.0 update 6, since this version will be used for the development.
As the AOC prototype shall be developed as a Protégé plugin, the chosen Protégé version for the development is the full Protégé version 3.2 beta. This version includes Protégé-OWL editor and optional plugins.
The extraction tool Text2Onto requires as components the tools Gate and WordNet. The latest Text2Onto version (text2onto-190506) uses Gate 3.1 and WordNet 2.1, those components shall be installed as components for the AOC prototype also.

3. Specific requirements

3.1 External interface requirements

3.1.1 User interfaces

Name of item: Add a new ontology design pattern
Description of purpose: Allow a user to construct a new ontology design pattern and save it
Destination output: Save the pattern into the pattern catalogue
Valid range of accuracy: None
Units of measure: None
Timing: Every time it is needed to construct a new ontology or complete the pattern catalogue
Relationships to other inputs: None
Screen formats: None
Data formats: The data must include the concepts, associations, synonyms that have been added to the pattern during the construction process. The file format shall be text file.
Command formats: None
End messages: Pattern successfully created

3.1.1.1 SRE01: Requirement of the user interface

The user interface shall be easy to use, and also shall fulfil all the requirements specified in this document. The user shall be able to use the AOC prototype functionalities through a graphical interface.

3.1.2 Hardware interfaces

They have not been defined.

3.1.3 Software interfaces

3.1.3.1 SRE02: Integration of the AOC prototype as a Protégé plugin

The AOC prototype shall be build as a plugin for the Protégé ontology editor, so that the ontologies generated by the prototype can be edited and used by the other Protégé plugins.

3.1.4 Communication interfaces

They have not been defined.

3.1.5 Ontology design pattern and ontology construction interface

3.1.5.1 SRE01-01: Compatibility with Protégé-OWL editing interface

The functionalities of the Protégé-OWL framework shall be used to construct both the ontology design patterns and the ontologies that are generated by the prototype system.

3.1.5.2 SRO01: Compatibility with Protégé frames editing interface

The functionalities of the Protégé frames shall be used to construct both the ontology design patterns and the ontologies that are generated by the prototype system.

3.2 System features

This section defines all the features of the AOC prototype system. Each feature is described according to its purpose, the sequence of interaction between the user and the AOC prototype system for the feature and finally, associated requirement for the feature.

The sequence of interaction between the user and the AOC prototype system utilize tables that show the progression of the stimulus and response involved during the feature.

3.2.1 SRE03: Give a name to an ontology design pattern

3.2.1.1 Purpose of feature

The AOC prototype shall permit to identify a pattern by saving a name for this ODP.

Stability: Stable
Degree of necessity: Essential

3.2.1.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to save an ODP	
	2. Request for an ODP name
3. Enter a name for the ODP	
	4. Request for an ODP description
5. Confirm saving	
	6. Save the ODP name together with its description and content

Inspection: Check that the name given to the ODP does not already exist in the pattern catalogue.
Error: An error is raised if the ODP name is empty.

3.2.1.3 Associated functional requirement

No associated functional requirement.

3.2.2 SRC01: Save a short description of an ontology design pattern

3.2.2.1 Purpose of feature

The AOC prototype shall permit to add comments about a pattern by saving a short description of that ODP.

Stability: Stable
Degree of necessity: Conditional

3.2.2.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to save an ODP	
	2. Request for an ODP name
3. Enter a name for the ODP	
	4. Request for an ODP description
5. Enter a short description for the ODP	
6. Confirm saving	
	7. Save the ODP description together with its content and name

Inspection: No specific inspection.
Error: No error is raised if this field is empty.

3.2.2.3 Associated functional requirement

No associated functional requirement.

3.2.3 SRE04: Add a concept to an ontology design pattern

3.2.3.1 Purpose of feature

The AOC prototype shall permit to add concepts to an ontology design pattern.

Stability: Stable
Degree of necessity: Essential

3.2.3.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request for creating a new ODP	
	2. Open and display the ODP construction facilities
3. Add a new concept	
	4. Request for a concept name
5. Enter a name for the concept	
6. Validate the name	
	7. Add the concept in the ODP and display the concept name

Inspection: Check that the name of the concept is not already used by another concept.
Error: An error is raised if two concepts have the same name.

3.2.3.3 Associated functional requirement

SRC02: Add synonyms to a concept in an ontology design pattern
SRC03: Add user-own synonyms to a concept in an ontology design pattern

3.2.4 SRC02: Add synonyms to a concept in an ontology design pattern

3.2.4.1 Purpose of feature

The AOC prototype shall permit the user to select automatically generated synonyms to a concept and afterwards add them to an ontology design pattern.

Stability: Stable
Degree of necessity: Essential

3.2.4.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Select a concept	
2. Request to add synonyms to the concept	
	3.Compute a list of synonyms for the concept
	4. Display the list of synonyms for the concept
	5. Display a field to enter a synonym
6. Select synonyms to add from the suggested list	
7. Confirm to add the synonyms	
	8. Save the synonyms for the concept

Inspection: Check that the synonyms selected by the user are not already in the synonyms list of the concept.
Error: An error is raised if a synonym is added more than once for the same concept.

3.2.4.3 Associated functional requirement

No associated functional requirement.

3.2.5 SRC03: Add user-own synonyms to a concept in an ontology design pattern

3.2.5.1 Purpose of feature

The AOC prototype shall permit the user to add his own synonyms to a concept in an ontology design pattern.

Stability: Stable
Degree of necessity: Essential

3.2.5.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Select a concept	
2. Request to add synonyms to the concept	
	3.Compute a list of synonyms for the concept
	4. Display the list of synonyms for the concept
	5. Display a field to enter a synonym
6. Enter a label for the synonym	
7. Confirm to add the synonym	
	8. Save the synonym for the concept

Inspection: Check that the synonym entered by the user is not already in the synonyms list of the concept.
Error: An error is raised if a synonym is added more than once for the same concept.

3.2.5.3 Associated functional requirement

No associated functional requirement.

3.2.6 SRE05: Add text files to the text corpus

3.2.6.1 Purpose of feature

The AOC prototype shall permit to select and add one or several text files to the text corpus and then extract terms from those text files.

Stability: Stable
Degree of necessity: Essential

3.2.6.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to add new text file	
	2. Display the file explorer
3. Navigate on the hard disk and select the files to add to the text corpus	
4. Validate the selection	
	5. Set the selected files as text corpus

Inspection: Check that the files contained in the text corpus are text files.
Error: An error is raised if a selected file is not a text file.

3.2.6.3 Associated functional requirement

SRE06: Extract terms from the text corpus.

3.2.7 SRE06: Extract terms from the text corpus

3.2.7.1 Purpose of feature

The AOC prototype shall permit to extract terms from a text corpus.

Stability: Stable
Degree of necessity: Essential

3.2.7.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to add texts in the text corpus	
	2. Display a file explorer
3. Select the folder corresponding to the text file location	
4. Select the files corresponding to the text corpus	
5. Confirm text corpus selection	
	6. Set the selected files as text corpus
	7. Extract the terms from the text corpus
	8. Save the extracted terms in a text file

Inspection: Check that the file containing the text corpus is a text file. Check for double in the text corpus.
Error: An error is raised if the selected file is not a text file.

3.2.7.3 Associated functional requirement

SRE10: Save the list of terms and concepts matched
SRE11: Generate associations from a list of extracted terms
SRE23: Match a list of terms against a list of concepts in ontology pattern

3.2.8 SRE07: Set a threshold for string comparison

3.2.8.1 Purpose of feature

The AOC prototype shall permit to set a threshold for string comparison. It permits to set a limit for considering when a concept and an extracted term match.

Stability: Stable
Degree of necessity: Essential

3.2.8.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to set a value a for the string threshold	
	2. Request to enter a value for the threshold
3. Enter a value for the threshold	
4. Confirm the threshold value	
	5. Save the threshold value

Inspection: Check that the threshold value is a number.
Error: An error is raised if the threshold format is incorrect.

3.2.8.3 Associated functional requirement

SRE10: Save the list of terms and concept matched

3.2.9 SRE08: Select a string matching algorithm

3.2.9.1 Purpose of feature

The AOC prototype shall permit to select an algorithm for matching the extracted terms and associations against the concepts and associations in the ODPs.

Stability: Stable
Degree of necessity: Essential

3.2.9.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to select a string matching algorithm	
	2. Display a list of string matching algorithm
3. Select a string matching algorithm	
4. Confirm the selection	
	5. Set the selected algorithm for the matching process

Inspection: Check the user must choose one algorithm in the list.
Error: An error is raise if no algorithm is selected.

3.2.9.3 Associated functional requirement

SRE23: Match a list of terms against a list of concepts in ontology pattern

3.2.10 SRE09: Compute the number of terms and concepts successfully matched

3.2.10.1 Purpose of feature

The AOC prototype shall permit to evaluate the number of successfully matched terms and concepts for each pattern.

Stability: Stable
Degree of necessity: Essential

3.2.10.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to compute the terms matching score	
	2. Consider the patterns selected in SRE13
	3 For each patterns matched in SRE10, compute the percentage of terms matched for the ODP (HM = Number of concepts matched/ Number of concept in ODP)
	4. Return the percentage HM

Inspection: No specific inspection.
Error: No specific error.

3.2.10.3 Associated functional requirement

SRE10: Save the list of terms and concepts matched
SRE19: Compute the matching score

3.2.11 SRE10: Save the list of terms and concepts matched

3.2.11.1 Purpose of feature

The AOC prototype shall permit to keep track of the list of extracted terms that have been matched against the concepts of a specific ontology. The saved list shall also contain the score of the matching process.

Stability: Stable
Degree of necessity: Essential

3.2.11.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to match patterns against extracted terms	
	2. Match the terms against the concepts list generated in SRE13 and according to the algorithm set in SRE08
	3. Discard all the terms that have a matching score below the limit set in SRE07
	4. Write in a text file a list of quadruplet (extracted term(i),concept (j), matching score (i,j),pattern name)

Inspection: No specific inspection.
Error: No specific error.

3.2.11.3 Associated functional requirement

SRE14: Convert a list of generated associations to a list of associations of concepts

3.2.12 SRE11: Generate associations from a list of extracted terms

3.2.12.1 Purpose of feature

The AOC prototype shall permit to extract a list of potential associations from a text corpus.

Stability: Stable
Degree of necessity: Essential

3.2.12.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to add texts in the text corpus	
	2. Display a file explorer
3. Select the folder corresponding to the text file location	
4. Select the files corresponding to the text corpus	
5. Confirm text corpus selection	
	6. Set the selected files as text corpus
	7. Extract the associations from the text corpus
	8. Save the extracted associations in a text file

Inspection: No specific inspection.
Error: No specific error.

3.2.12.3 Associated functional requirement

SRE15: Match a list of converted associations against a list of associations in an ontology design pattern

3.2.13 SRE12: Generate a list of all associations in ontology design pattern

3.2.13.1 Purpose of feature

The AOC prototype shall permit to create a list that contains all the associations in the ontology design patterns.

Stability: Stable
Degree of necessity: Essential

3.2.13.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to set ontology design patterns for the association matching process	
	2. Consider the list of selected ontology design patterns in SRE13
	3. For all the associations in the selected patterns, write the triplet (concept name1, concept name2, pattern name) in a text file

Inspection: No specific inspection.
Error: No specific error.

3.2.13.3 Associated functional requirement

SRE15: Match a list of converted associations against a list of associations in an ontology design pattern

3.2.14 SRE13: Generate a list of all concepts in ontology design pattern

3.2.14.1 Purpose of feature

The AOC prototype shall permit to create a list that contains all the concepts in ontology design patterns.

Stability: Stable
Degree of necessity: Essential

3.2.14.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to set ontology design patterns for the concept matching process	
	2. Request to select ontology design patterns
	3. Display a list of ontology design patterns
4. Select ontology design patterns	
5. Validate selection	
	6. Consider the list of selected ontology design patterns
	7. For all the concepts in the selected patterns, write the pair (concept name, pattern name) in a text file

Inspection: No specific inspection.
Error: No specific error.

3.2.14.3 Associated functional requirement

SRE23: Match a list of terms against a list of concepts in ontology pattern

3.2.15 SRE14: Convert a list of generated associations to a list of associations of concepts

3.2.15.1 Purpose of feature

The AOC prototype shall permit to convert a list of generated association between extracted terms, to a list of associations with the concepts that matched the extracted terms.

Stability: Stable
Degree of necessity: Essential

3.2.15.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to transform terms association to concept association	
	2. Consider the list of associations from the text file generated in SRE11
	3. For all the terms in the associations, identify the concept that match to the term using, the list of concepts and terms matched generated in SRE10
	4. Write in a new text file, the list of association from SRE11 but with the matched concept labels from SRE10

Inspection: No specific inspection.
Error: No specific error.

3.2.15.3 Associated functional requirement

SRE15: Match a list of converted associations against a list of associations in an ontology pattern
SRE19: Compute the matching score

3.2.16 SRE15: Match a list of converted associations against a list of associations in ontology pattern

3.2.16.1 Purpose of feature

The AOC prototype shall permit to match a list of converted associations against all the associations in an ontology design pattern.

Stability: Stable
Degree of necessity: Essential

3.2.16.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to match extracted associations against associations in the patterns	
	2. Consider the list of converted associations (Conv1) from the text file generated in SRE14
	3. Consider the list of all associations in the ODPs (AssList) from the text file generated in SRE12
	4. Match all associations in Conv1 against associations in AssList
	5. Save all the associations that matches the two list, together with the pattern name, in a text file

Inspection: No specific inspection.
Error: No specific error.

3.2.16.3 Associated functional requirement

SRE16: Save a list of extracted associations
SRE17: Compute the number of matched associations

3.2.17 SRE16: Save a list of extracted associations

3.2.17.1 Purpose of feature

The AOC prototype shall permit to save a list of extracted associations that have been matched against all the associations in ontology design patterns.

Stability: Stable
Degree of necessity: Essential

3.2.17.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to save extracted associations	
	2. Consider the list of associations generated in SRE15
	3. For each association, replace the concept labels, by the terms that matches the concept
	4. Save in the selected file, the new list of association generated from the previous step

Inspection: Check the user must a text file for saving the associations.
Error: An error is raise if no file is selected as target file to write the relations.

3.2.17.3 Associated functional requirement

SRE17 Compute the number of matched associations

3.2.18 SRE17: Compute the number of matched associations

3.2.18.1 Purpose of feature

The AOC prototype shall permit to compute the number of associations that have been matched successfully against the associations in the ODPs.

Stability: Stable
Degree of necessity: Essential

3.2.18.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to compute the association matching score	
	2. Consider the ODPs selected in SRE13
	3 For each ODP, matched in SRE15, compute the percentage of matches (NA = Number of association matched / Number of association in ODP)
	4. Return the percentage NA

Inspection: No specific inspection.
Error: No specific error.

3.2.18.3 Associated functional requirement

SRE19: Compute the matching score

3.2.19 SRE18: Set a formula for matching score computation

3.2.19.1 Purpose of feature

The AOC prototype shall permit the user to set the values of the weights parameters that are needed in the score computation formula.

Stability: Stable
Degree of necessity: Essential

3.2.19.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to set a score formula	
	2. Request to set the value of the weights for the percentage of terms and associations extracted
3. Set values for weights	
4. Validate the weights values	
	5. Update the score computation formula with the selected weights

Inspection: The values of the weights entered by the user shall be different from zero.
Error: An error is raised if in weights values are negative or null.

3.2.19.3 Associated functional requirement

SRE19: Compute the matching score

3.2.20 SRE19: Compute the matching score

3.2.20.1 Purpose of feature

The AOC prototype shall permit to compute a score based on the matching process of the extracted terms and associations and the concepts and associations in ontology design pattern.

Stability: Stable
Degree of necessity: Essential

3.2.20.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to compute the matching score	
	2. Consider the percentage of terms and concepts matched for each ODP using SRE09
	3. Consider the percentage of matched associations for each ODP using SRE17
	4. For each ODP, compute the score considering the weights values set by the user in SRE18
	5. Save in a text file, the ODP names together with their matching score

Inspection: No specific inspection.
Error: No specific error is raised.

3.2.20.3 Associated functional requirement

SRE20: Set a threshold for ontology pattern selection
SRE22: Construct ontology

3.2.21 SRE20: Set a threshold for ontology pattern selection

3.2.21.1 Purpose of feature

The AOC prototype shall permit the user to set a threshold for the matching score so that all the ODPs that have a score above this limit are taken into account for the ontology building process. Other ODPs shall not be included for the ontology building process.

Stability: Stable
Degree of necessity: Essential

3.2.21.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to set a threshold for the ODP selection	
	2. Request a value for the threshold
3. Enter the threshold value	
4. Validate the threshold value	
	5. Save the threshold value for ODP selection
	6. For each ODP (included in the file generated in SRE19) that have a score above the threshold save in a text file; the pattern name, the matching score ,the extracted terms list , the matched associations list, the name of the string metric, the formula used to compute the score

Inspection: No specific inspection.
Error: No specific error is raised.

3.2.21.3 Associated functional requirement

SRE22: Construct ontology

3.2.22 SRE21: Update an ontology design pattern

3.2.22.1 Purpose of feature

The AOC prototype shall permit to edit an ontology design pattern and make changes on this one. The changes can be to add/remove/update the concepts, synonyms, or associations into the pattern.

Stability: Stable
Degree of necessity: Essential

3.2.22.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to open an ontology design pattern	
	2. Display a file explorer
3. Select an ontology design pattern through the file explorer	
4. Validate the selection	
	5. Display the ontology design pattern concepts, associations
6. Edit the elements (association, concept, synonym) of the ontology design pattern	
7. Request to save the updated ontology design pattern	
	8. Request for saving confirmation
9. Confirm saving	
	10. Update the file containing the ontology design pattern

Inspection: No specific inspection.
Error: No specific error is raised.

3.2.22.3 Associated functional requirement

No associated functional requirement.

3.2.23 SRC04: Select a method for ontology construction

3.2.23.1 Purpose of feature

The AOC prototype shall permit the user to select a method for ontology construction.

Stability: Stable
Degree of necessity: Essential

3.2.23.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to set ontology construction method	
	2. Display a list of available AOC method
3. Select one of the AOC method	
4. Confirm the selected method	
	5. Construct the ontology according to the selected AOC method

Inspection: No specific inspection.
Error: No specific error is raised.

3.2.23.3 Associated functional requirement

SRE22: Construct ontology

3.2.24 SRC05: Select heuristics for ontology construction

3.2.24.1 Purpose of feature

The AOC prototype shall permit the user to select heuristics for ontology construction.

Stability: Stable
Degree of necessity: Essential

3.2.24.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to set heuristics for ontology construction	
	2. Display a list of heuristics for ontology building
3. Select heuristics among the suggested list	
4. Confirm the heuristics selection	
	5. Construct the ontology according to the selected heuristics

Inspection: No specific inspection.
Error: No specific error is raised.

3.2.24.3 Associated functional requirement

SRE22: Construct ontology

3.2.25 SRE22: Construct ontology

3.2.25.1 Purpose of feature

The AOC prototype shall construct automatically ontology from a set of extracted terms and associations, a set of heuristics and finally a set of ontology design patterns.

Stability: Stable
Degree of necessity: Essential

3.2.25.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to construct ontology	
	2. Request to select a location to save the new ontology
	3. Display a file explorer
4. Selection of a location for the ontology	
	5. Request to enter a name for the new ontology
6. Set a name for the new ontology	
	7. Consider the ODPs having a score above the ODP threshold set in SRE20
	8. For each pattern consider the list of matched terms and associations
	9. Build the ontology according to the selected method in SRC04, and heuristics in SRC05
	10. Save the ontology name together with the concepts, associations, and synonyms generated

Inspection: No specific inspection.
Error: No specific error is raised.

3.2.25.3 Associated functional requirement

SRE10: Save the list of terms and concepts matched

3.2.26 SRE23: Match a list of terms against a list of concepts in ontology pattern

3.2.26.1 Purpose of feature

The AOC prototype shall permit to match a list of extracted terms against all the concepts in an ontology design pattern.

Stability: Stable
Degree of necessity: Essential

3.2.26.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to match terms against concepts in patterns	
	2. Consider the list of extracted terms (TermList) from the text file generated in SRE06
	3. Consider the list of all concepts (ConList) in the patterns from the text file generated in SRE13
	4. Match all terms in TermList against the concepts in ConList
	5. Save the all the terms and concepts that have a matching score above the string threshold set in SRE07

Inspection: No specific inspection.
Error: No specific error.

3.2.26.3 Associated functional requirement

SRE19: Compute the matching score

3.2.27 SRE24: Set predefined values for the weights parameters of the score computation formula

3.2.27.1 Purpose of feature

The AOC prototype shall permit the user to choose suggested values for the formula used to compute the matching score for the patterns. In this case the system is responsible for computing the values of the weight parameters and afterwards the matching score.

Stability: Stable
Degree of necessity: Essential

3.2.27.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to set a formula for the matching score computation	
2. Request to set automated values for the formula parameters	
	3. Compute the value of the parameters the formula
	4. Update the score computation formula with the computed weights

Inspection: The computed values for the weights shall be different from zero.
Error: An error is raised if the computed weights values are null.

3.2.27.3 Associated functional requirement

No associated functional requirement.

3.2.28 SRE25: Add association in ontology design pattern

3.2.28.1 Purpose of feature

The AOC prototype shall permit the user to add association between the concepts in an ontology design pattern.

Stability: Stable
Degree of necessity: Essential

3.2.28.2 Stimulus/Response sequence

User	**AOC Prototype System**
1. Request to construct ontology design pattern	
2. Selection of ontology properties	
3. Request to add ontology associations	
	4. Request to enter association name
5. Enter a name for the association	
6. Validate association name	
	7. Request to select the concepts that are linked by the association
8. Selection of the concept names	
	9. Request to add a relation type
10. Selection of a type of association	
11. Validate the association creation	
	12. Add the association to the ODP and display the association name

Inspection: The user should not be able to add an already existing association.
Error: An error is raised if the user adds an association already existing in the pattern.

3.2.28.3 Associated functional requirement

No associated functional requirement.

3.3 Performance requirements

3.3.1 SRE26: Number of terminals to be supported by the prototype

One terminal shall be sufficient to install and use the AOC prototype system and its required components.

3.3.2 SRE27: Number of simultaneous users to be supported by the prototype

No simultaneous access is required for the AOC prototype system.

3.3.3 SRE28: Amount and type of information to be handled by the prototype

The AOC prototype system shall handle alphanumeric data type. Tough the amount of information is not fixed since it depends on the size of the ontology to construct, the amount

of ontology design patterns, the amount of synonyms evolved in the ontology design patterns, an order of magnitude for the amount of information could be several 100MB.

3.4 Software system attributes

3.4.1 SRE29: Requirement on the prototype system maintainability

The AOC prototype system shall be designed and documented such that any person familiar with the AOC method presented in [1] is capable of using and maintaining AOC prototype system.

The prototype shall be designed so that the functionality of the AOC prototype can be enlarged by plugging new components. Also, the functionalities of the prototype shall be adaptable by a minimum of programming effort.

All the code which will be implemented for the prototype system shall be written in Java.

Software Design descriptions Document for the Automatic Ontology Construction Prototype System

Student: Ludovic Jean-Louis
Teacher: Eva Blomqvist

Document Evolution		
Indices	Date	Comments
A	06/10/19	Initial version
B	07/01/22	Update of the complete document
C	20/02/07	Update of section 6.1

Table of contents

List of figure

1. Introduction

1.1 Purpose

The present document is a statement of the design of the automatic ontology construction prototype system. The document aims at providing all necessary explanations to achieve all the, automatic ontology construction prototype system, requirements stated in the document named ***Requirement Specifications Document for the automatic ontology construction prototype*** [3]. This document will discuss how to divide the prototype in modules that can work together and how the prototype will interact with the user when this later uses some modules.

1.2 Scope

The prototype system will help in validating the general framework for automatic ontology building presented in [1]. A succeeding goal is to reduce the time and effort required to build ontologies through the use of an automated method.

1.3 Definitions, acronyms, and abbreviations

- **AOC**: Automatic ontology construction.
- **ODP**: Ontology design pattern.
- **GUI**: Graphical user interface.
- **Text corpus**: "*A large and structured set of texts*"[1].
- **Term**: We consider a term as a group of words that possibly refers to an explicit concept in a text corpus.
- **Concept**: We consider a concept as "*an abstract idea or a mental symbol, typically associated with a corresponding representation in language*"[2].
- **Association/relation**: We consider an association or relation as a link between two concepts or between two terms.

1.4 Overview of the document

This document describes the design specification for the AOC prototype system. It is divided into 5 parts. Part 2 gives a description of the prototype system decomposition into design entities. In part 3 all relationships between the design entities are presented together with the system resources needed. Part 4 gives an overview of the knowledge required by the system developer to deal with the design entities. Part 5 gives a detailed description of the design entities listed in part 2.

2. References

[1]. Blomqvist, E. (2005) Fully Automatic Construction of Enterprise Ontologies Using Design Patterns: Initial Method and First Experiences. Lecture Notes in Computer Science p1314-1329
[2]. IEEE Std 1016-1998 IEEE Recommended Practice for Software Design Descriptions
[3]. Software Requirement Specifications Document for the Automatic Ontology Construction Prototype System

[1] www.wikipedia.org
[2] www.wikipedia.org

3. Decomposition description

3.1 Module description

The AOC prototype system can be divided into different modules as presented in the following Figure 3-1.

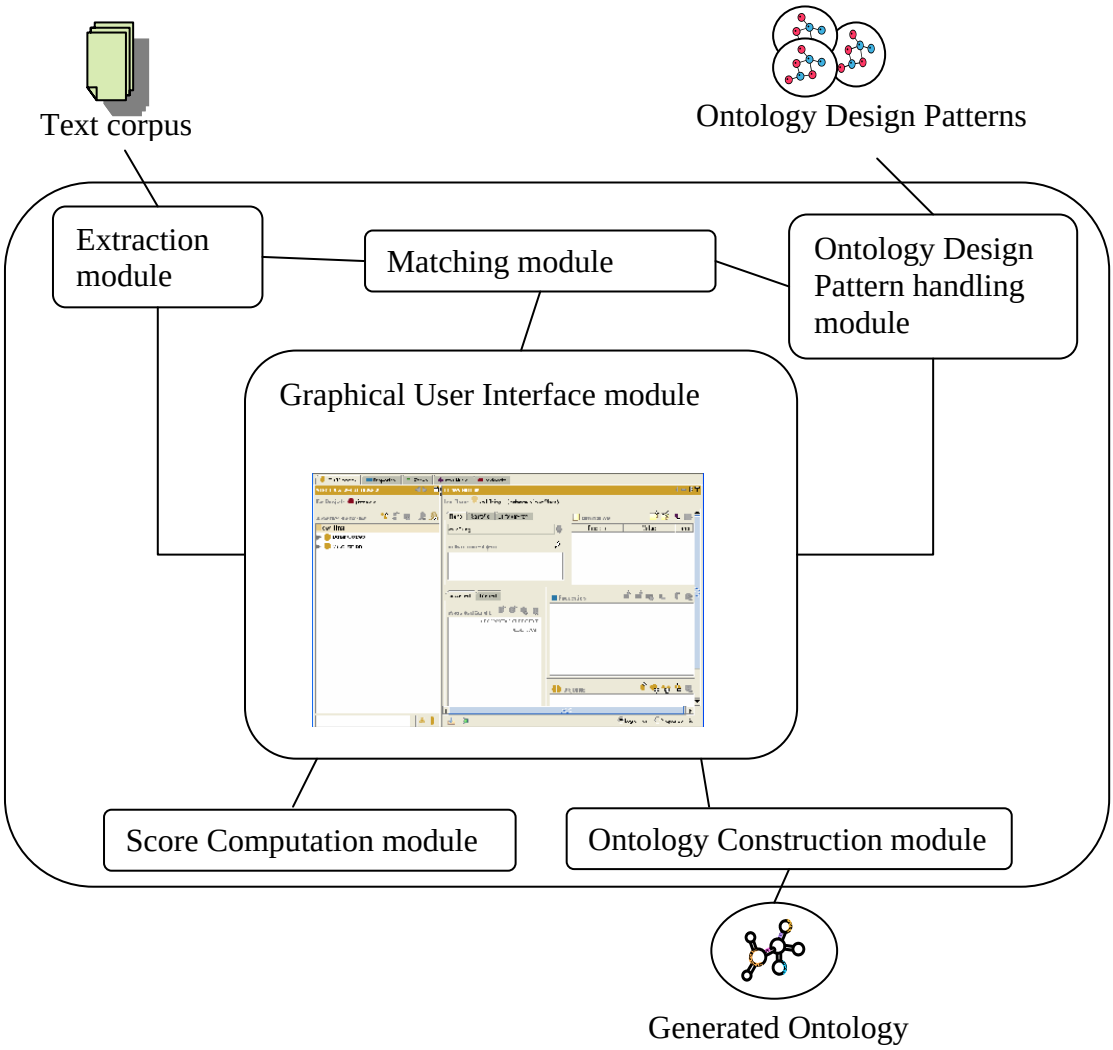

Figure 3-1 Architecture of the prototype system

3.1.1 M1: Extraction module description

Identification	M1Ext
Type	Module
Purpose	This module allows extracting terms and associations from the text corpus. It satisfies the requirements SRE08, and SRE13.
Function	This module provides methods for extracting terms and associations from a text corpus. The module functionalities are called by the module M6GUI when constructing a new ontology.
Subordinates	No subordinates.

3.1.2 M2: Matching module description

Identification	M2Match
Type	Module
Purpose	This module allows matching extracted terms against concepts in ODPs, converting extracted associations to associations of concepts, matching of converted associations against associations in ODPs. It satisfies the requirements SRE09, SRE10, SRE11, SRE12, SRE14, SRE15, SRE16, SRE17, SRE18, and SRE19.
Function	This module provides methods for matching the terms and associations extracted by using M1Ext and match them against the concepts and associations in the ODPs. The module functionalities are called by the module M6GUI when constructing a new ontology.
Subordinates	This module uses the functionalities of M5ODP for reading the content of the ODPs.

3.1.3 M3: Score computation module description

Identification	M3ScoreComp
Type	Module
Purpose	This module allows calculating the matching score of the extracted terms and associations against the ODP's concepts and associations. This module satisfies the requirement SRE20, SRE21, SRE22, and SRE27.
Function	This module provides methods for calculating the matching score of each ODP in the pattern catalogue. The module functionalities are called by the module M6GUI when setting a formula to compute the score, settings weights values for a score formula, setting the ODP threshold.
Subordinates	No subordinates.

3.1.4 M4: Ontology construction module description

Identification	M4OntConst
Type	Module
Purpose	This module allows constructing a new ontology from the matched terms and associations and their related ODPs. This module satisfies the requirements SRE24, SRE25, and SRE26.
Function	This module provides methods for adding concepts and associations to the generated ontology based on the accepted ODPs, and their matched terms and associations. The module functionalities are called by the module M6GUI when constructing a new ontology.
Subordinates	No subordinates.

3.1.5 M5: Ontology design pattern handling module description

Identification	M5ODP
Type	Module
Purpose	This module allows constructing ODPs and adding synonyms to the concepts in the ODPs. This module satisfies the requirement SRE03, SRC01, SRE04, SRE05, SRE06, SRE14, SRE15, and SRE29.
Function	This module provides methods for adding concepts and associations to the ODPs, and adding synonyms to the ODP concepts by using the Protégé-OWL facilities. Also the module is used to store the concepts and associations of the ODPs into text files.
Subordinates	No subordinates.

3.1.6 M6: Graphical user interface module description

Identification	M6GUI
Type	Module
Purpose	The user interface provides a graphical representation of the prototype system functionalities. This module is involved in the accomplishment of the requirements SRE01, SRE02, and SRE07.
Function	The GUI permits the user to interact with the previous modules functions and set different parameter values.
Subordinates	M1Ext, M2Match, M3ScoreComp, M4OntConst, M5ODP.

3.2 Concurrent process description

The execution of the AOC prototype system functionalities does not involve any concurrent processes.

3.3 Data

This section aims at describing the structure of the data used in the different module of the AOC prototype system. Since the data used by the prototype system functionalities are stored in text files, the data will be decomposed according to the following format; i) the variables name, ii) the name of the class handling the variables, iii) the semantic of the variables.

3.3.1.1 M1 data

Variables	Class handling the data	Semantic of the data
corpus	ExtractText2Onto	Represents the text corpus.
pom	ExtractText2Onto	Represents a probabilistic ontology model containing the extracted terms and associations.
ac	ExtractText2Onto	Represents an algorithm controller containing the algorithms used for extracting the terms and the associations from the text corpus.
concepts	ExtractText2Onto	Represents a list containing all the extracted terms from

		the text corpus.
relation	ExtractText2Onto	Represents a list containing all the extracted associations from the text corpus.

3.3.1.2 M2 data

Variables	Class handling the data	Semantic of the data
nbConceptMatch	TermsAndConceptMatch	Represents the amount of concept match for an ODP.
termsTable	TermsAndConceptMatch	Represents a string table containing the extracted terms.
conceptTable	TermsAndConceptMatch	Represents a string table containing the ODP concepts.
patternTable	TermsAndConceptMatch	Represents a string table containing the ODP names.
matchScore	TermsAndConceptMatch	Represents a double containing the computed matching score of two strings.
AlgoName	TermsAndConceptMatch	Represents the name of the string metric chosen for matching the extracted terms against the ODP concepts.
propertyName	AssociationConversion	Represents the label of the association that has to be converted.
termAssDomain	AssociationConversion	Represents the association domain of the association to be converted.
termAssRange	AssociationConversion	Represents the association range of the association to be converted.
conceptAssDomain	AssociationConversion	Represents the converted association domain label.
conceptAssRange	AssociationConversion	Represents the converted association range label.
NbAssMatched	AssociationMatching	Represents the number of associations successfully matched.
PatternAsso	AssociationMatching	Represents a table containing both domain and range of the ODP associations.
convertedAsso	AssociationMatching	Represents a table containing both domain and range of the converted associations.
patternName	AssociationMatching	Represents a table containing the name of the ODPs.

3.3.1.3 M3 Data

Variables	Class handling the data	Semantic of the data
formulaName	ComputeScore	Represents the name of the formula set to compute the score.
threshold	ComputeScore	Represents the value of the ODP threshold.
score	ComputeScore	Represents the matching score for an ODP.
Nbconcepts	ComputeScore	Represents the number of concepts in an ODP.
NbAsso	ComputeScore	Represents the number of associations in an ODP.
NbConMatched	ComputeScore	Represents the number of distinct concept matched for an ODP.
NbAssoMatched	ComputeScore	Represents the number of distinct association matched for an ODP.
PercenConceptMatched	ComputeScore	Represents the percentage of concept matched for an ODP.
PercenAssMatched	ComputeScore	Represents the percentage of association matched for an ODP.

3.3.1.4 M4 data

Variables	Class handling the data	Semantic of the data
owlModel	CompileOntology	Represents an OWL model containing the concepts and associations to write in the OWL output file.
property	CompileOntology	Represents an association in the OWL model.
Cls	CompileOntology	Represents a concept in the OWL model.
PatternName	CompileOntology	Represents the name of the ODP accepted for the AOC process.
conceptLabel	CompileOntology	Represents the label of an ODP concept to add to the new ontology.
relationLabel	CompileOntology	Represents the label of an ODP association to add to the new ontology.
relationDomain	CompileOntology	Represents the domain label of an ODP association to add to the new ontology.
relationRange	CompileOntology	Represents the range label of an ODP association to add to the new ontology.

3.3.1.5 M5 data

Variables	Class handling the data	Semantic of the data
m_model	ListPatternContent	Represents an OWL model used to read the content of the ODPs.
OntClass	ListPatternContent	Represents a list containing the concepts of an ODP.
OntProperty	ListPatternContent	Represents a list containing the associations of an ODP.
PatternFileName	ListPatternContent	Represents the location of the ODP on the hard disk.

3.3.1.6 M6 data

Variables	Class handling the data	Semantic of the data
CorpusListModel	AutoOntCons	A list that contains the text file in the text corpus.
PatternsListModel	AutoOntCons	A list that contains the ODP in the pattern catalogue.
ConceptAlgoList	AutoOntCons	A list that contains the algorithms for extracting terms from a text corpus.
AssociationAlgoList	AutoOntCons	A list that contains the algorithms for extracting terms from a text corpus.
PatternThreshold	AutoOntCons	Represents the value of the ODP threshold.
StringThreshold	AutoOntCons	Represents the value of the string threshold.
StrMetric	AutoOntCons	Represents the name of the string metric.
ScoreFormulaName	AutoOntCons	Represents the name of the formula set to compute the matching score.
OntologyName	AutoOntCons	Represents the name of the generated ontology.
aScoreParam	AutoOntCons	Represents the value of the “a” parameter of the score formula.
bScoreParam	AutoOntCons	Represents the value of the “b” parameter of the score formula.

4. Dependency description

4.1 Intermodule dependencies

The GUI module (M6GUI) provides several variable values for the other modules (M1Ext, M2Match, M3ScoreComp, M4OntConst, and M5ODP). As a result the GUI shall offer

several lists and fields in order to provide the parameters of the methods with the appropriate values. The GUI shall permit to gather values for the following parameters;

- Texts to add to the text corpus.
- ODPs to add to the pattern catalogue.
- String matching threshold.
- ODP threshold.
- Name of a string metric.
- Name of a formula for computing the matching score, and the corresponding values for the weights of the formula (in case the weights are needed).
- Name to save the generated ontology.

4.2 Interprocess dependencies

The AOC prototype system is not intended to communicate with a server as a result no interprocess dependencies have been identified.

4.3 Data dependencies

All the important data such as, extracted terms and associations, concepts and associations from the ODPs, the list of terms and ODP concepts matched, etc. are stored separated in text files, by following a specific syntax as presented in the requirement specification document [3]. Also no integrity constraints apply on the data used by the prototype system.

5. Interface description

In this section we describe the interface for each module and present the methods implemented in the modules.

5.1 Module interface

In this section we present the purpose of the methods implemented in each module (M1 to M6) that can be reused by other modules.

5.1.1 M1 interface

Identification	M1Ext
Function	This module provides methods for extracting terms and associations from a text corpus. The module functionalities are called by the module M6GUI when constructing a new ontology.
Interface	The functionalities implemented in this module are: • ExtractTermsAndRelations; extract terms and relation of terms from the text corpus. • WriteTerm; write an extracted term in the text file containing all the extracted terms for the text corpus set by the user. • WriteRelation; write an extracted relation in the text file containing all the extracted relations for the text corpus set by the user.

5.1.2 M2 interface

Identification	M2Match
Function	This module provides methods for matching the terms and associations extracted by using M1Ext and match them against the concepts and associations in the ODPs. The module functionalities are called by the module M6GUI when constructing a new ontology.

Interface	The functionalities implemented in this module are: • ListConceptInPatterns; retrieve the label of all the concepts in the ODPs contained in the pattern catalogue and write them together with the ODP name in a new text file. • ListAssociationsInPatterns; retrieve the label of all the relations name, relations domain and relations range in the ODPs contained in the pattern catalogue and write them together with the ODP name in a new text file. • SetAlgorithm; set a string metric for matching the extracted terms against the concepts in the ODPs. • SetThreshold; set a value for the string matching threshold. • MatchConcepts; match the extracted terms against the concepts in the ODPs according to the selected string threshold, and string metric. • WriteScore; write in a new text file the list of the terms and concepts successfully matched together with their matching score and the name of the ODP used during the matching process. • GetNbConceptMatch; returns the number of concepts that have been matched for a specific ODP. • BestMatch; retrieve the best matching score of a specific extracted term against several concepts of a same ODP. • FindConcept; retrieve the concept that match the best a specific extracted term. • ConvertAssociation; replace the domain and range labels of an extracted association by their best match concept in the ODPs. • WriteAssociation; write in a new text file the different parts of an association (association label, association domain, association range, ODP name). • MatchConvAssociation; match the domain and range of the converted association against the domain and the range of the associations in the ODPs. • WriteMatchedAssociation; write the successfully matched associations in a new text file. • GetNbAsssociationMatched; returns the number of associations matched for a specific ODP.

5.1.3 M3 interface

Identification	M3ScoreComp
Function	This module provides methods for calculating the matching of each ODP in the pattern catalogue. The module functionalities are called by the module M6GUI when setting a formula to compute the score, settings weights values for a score formula, setting the ODP threshold.
Interface	The functionalities implemented in this module are: • DistinctMatchedConcepts; returns the number of distinct concepts that have been matched against the extracted terms. • NumberOfPatternConcepts; return the number of concepts in a specific ODP. • NumberOfPatternAssociation; return the number of associations in a specific ODP. • NumberofMatchedAssociation; return the number of associations in ODP that have been matched against the extracted associations.

	• AutoScore; calculate a matching score according to the "*Automated weight values*" formula. • LinearScore; calculate a matching score according to the "*Basic linear combination*" formula. • SetPatternThreshold; set a value for the selection of the matching score • WriteScore; write in a new text file the matching of the ODP that are above the threshold set.

5.1.4 M4 interface

Identification	M4OntConst
Function	This module provides methods for adding concepts and associations to the generated ontology based on the accepted ODPs, and their matched terms and associations. The module functionalities are called by the module M6GUI when constructing a new ontology.
Interface	The functionalities implemented in this module are: • ConstructOntology; compile the ontology from the list matched concepts and associations in the ODPs, the heuristics set, and the construction method set. The constructed ontology is saved in an OWL file.

5.1.5 M5 interface

Identification	M5ODP
Function	This module provides methods for adding concepts and associations to the ODPs, and adding synonyms to the ODP concepts by using the Protégé-OWL facilities. Also the module is used to store the concepts and associations of the ODPs into text files.
Interface	The functionalities are reused from the Protégé-OWL ontology editor facilities: • Add class; add a concept to an ODP. • Add property; add an association between two ontology concepts. • Add disjoint property; add a disjoint property among a set of ontology concepts. • Add synonyms; add synonyms to the ODPs. • ListConceptInPatterns; retrieve the label of all the concepts in the ODPs contained in the pattern catalogue and write them together with the ODP name in a new text file. • ListAssociationsInPatterns; retrieve the label of all the relations name, relations domain and relations range in the ODPs contained in the pattern catalogue and write them together with the ODP name in a new text file.

5.1.6 M6 interface

Identification	M6GUI
Function	The GUI permits the user to interact with the previous modules functions and set different parameter values.
Interface	The user interface is composed of several buttons linked to the functionalities of the previously presented module; • Add text file; add a file to the text corpus. • Add ontology design pattern; add an ODP to the pattern catalogue. • Set pattern threshold; enter a value for the ODP selection process.

	• Set string metric; choose a string metric and set a value for the string threshold. • Set a score formula; choose a formula for computing the matching score. • Construct ontology; start the AOC process according to the different parameters required. • New ontology; initialize all the parameters to construct a new ontology. Several list boxes; • Text corpus; permit to visualize the content of the text corpus. • Pattern catalogue; permit to visualize the content of the pattern catalogue. • Concept extraction algorithms; permit to visualize the algorithms set for the term extraction process. • Association extraction algorithms; permit to visualize the algorithms set for the association extraction process.

5.2 Process interface

No description is required for process interfaces.

6. Detailed design

In this section we present a detailed description of the modules functionalities. For visualizing this description, sequence diagrams have been used. In order to understand how the functionalities of the AOC prototype perform, a first sequence diagram is used to describe all the main actions involved in the ontology construction process, then several sub-diagrams are used to give further information concerning those actions. Also the following class diagram (Figure 6-1) gives and overview of all the classes and their methods.

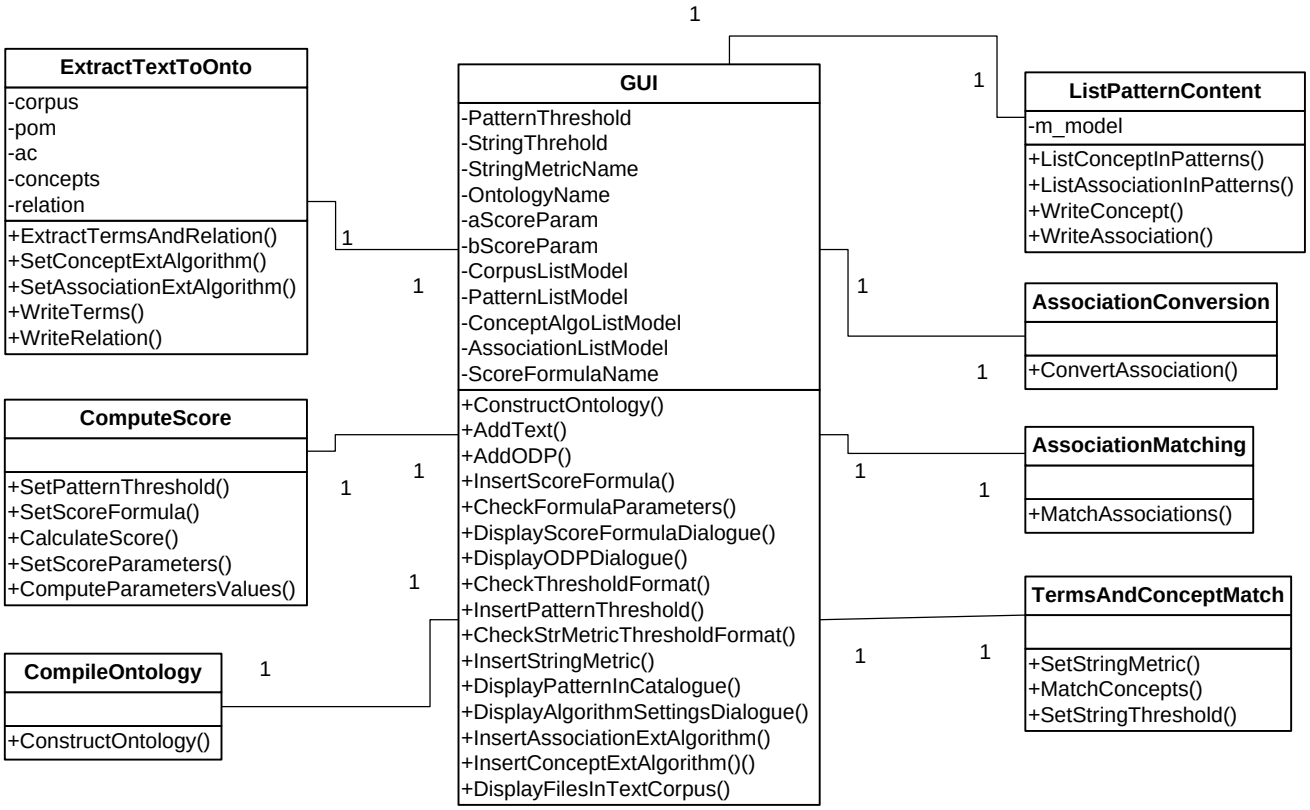

Figure 6-1 Class diagram of the prototype system

6.1 Ontology construction detailed design

The sequence diagram (Figure 6-2) gives a general overview of the operations accomplished during the AOC process.

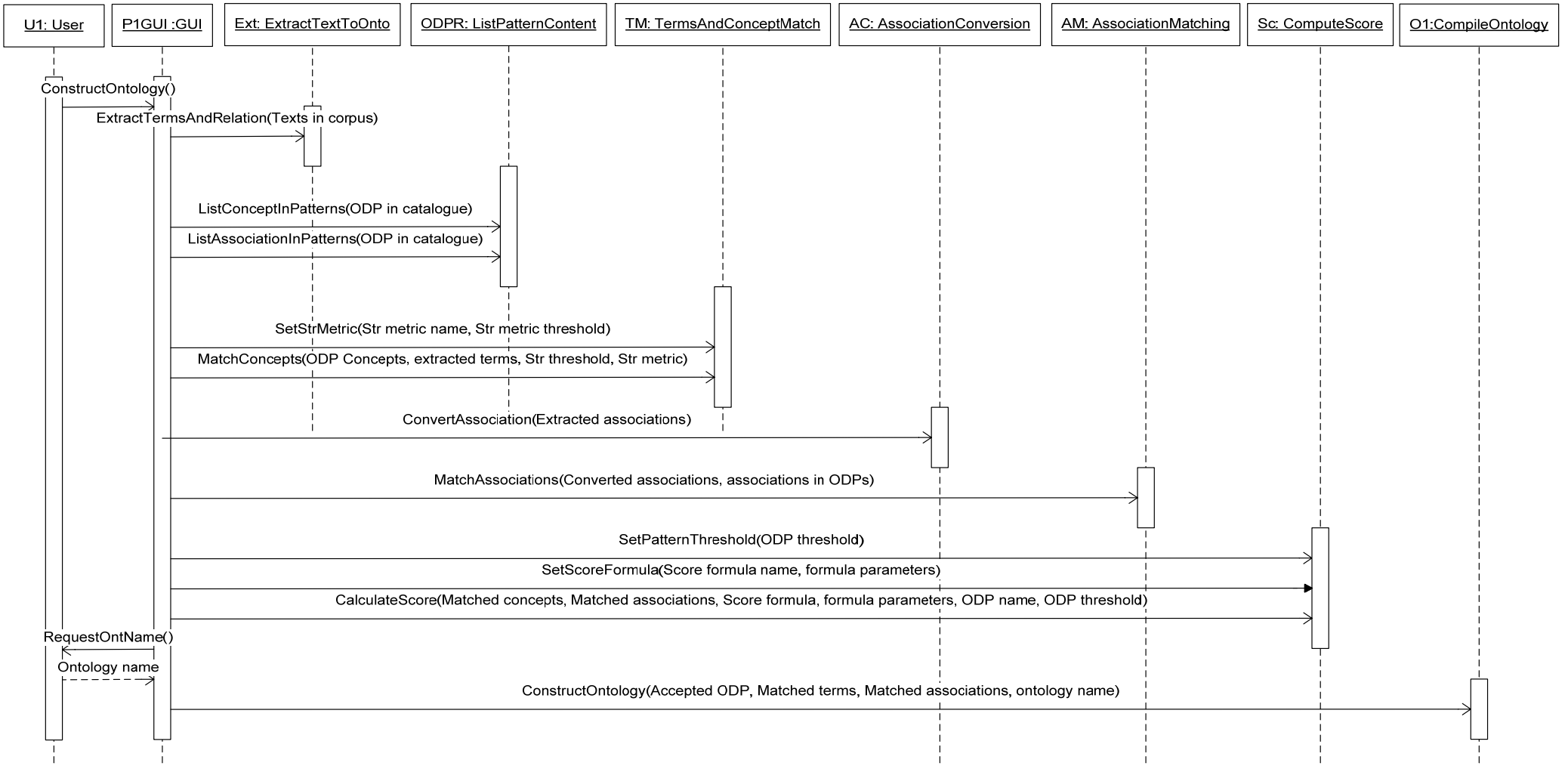

Figure 6-2 Ontology construction sequence diagram

6.1.1 Terms and association extraction detailed design

The following sequence diagrams (from Figure 6-3 to Figure 6-6) describe the interaction between the user and the prototype system for extracting terms and associations from a text corpus, add text to the text corpus, set algorithms for terms and associations extraction from the text corpus.

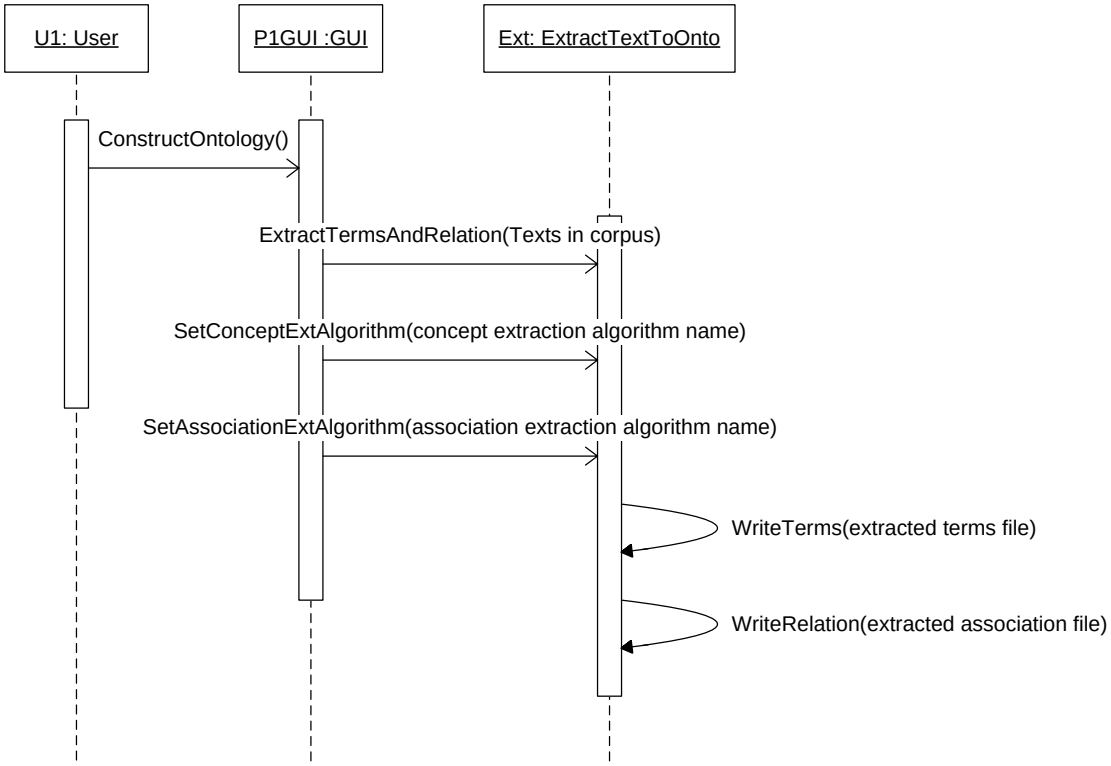

Figure 6-3 Terms and associations extraction sequence diagram

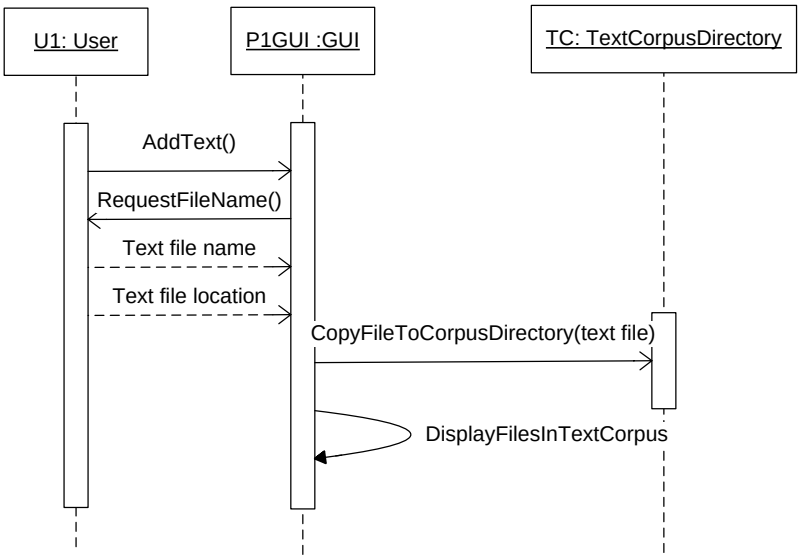

Figure 6-4 Add text files to the text corpus sequence diagram

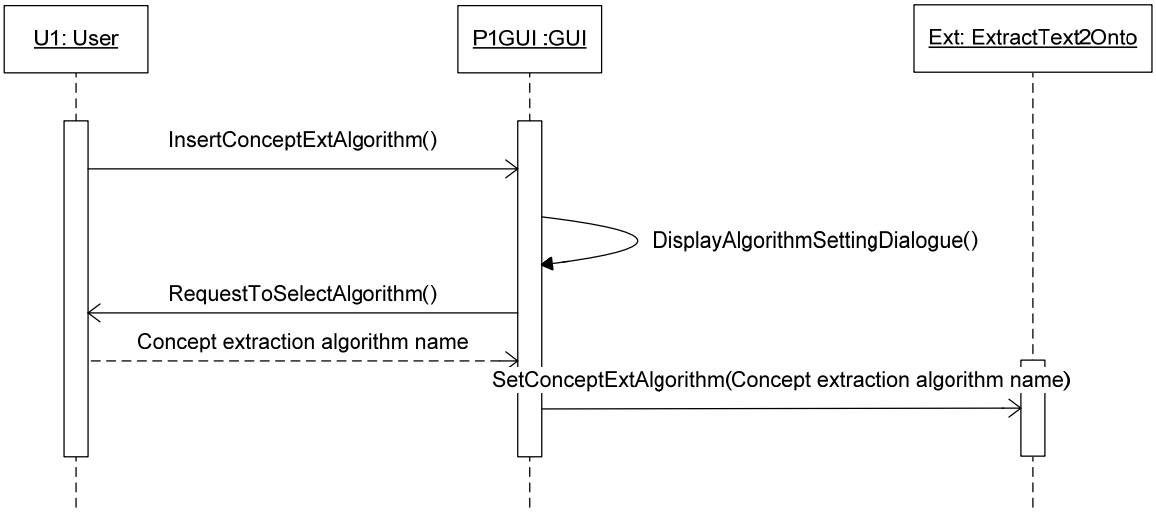

Figure 6-5 Set terms extraction algorithm sequence diagram

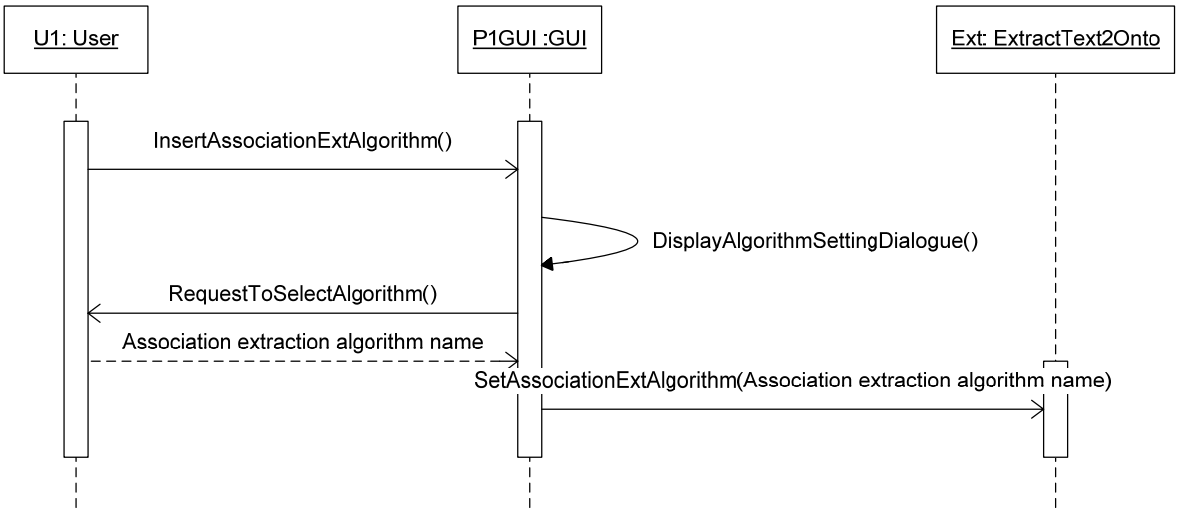

Figure 6-6 Set associations extraction algorithm sequence diagram

6.1.2 List pattern content detailed design

The following sequence diagrams (from Figure 6-7 to Figure 6-9) describe the actions performed for retrieving the concepts and associations in the ODPs and storing them in text files. Before listing the content of the ODPs it is necessary to have ODP in the pattern catalogue, as a result we first present the sequence diagram for adding ODP to the pattern catalogue and then the diagrams for reading their content. Describe how to read the ODP content refers to describe the actions involved in both "ListConceptsInPatterns" and "ListAssociationsInPatterns".

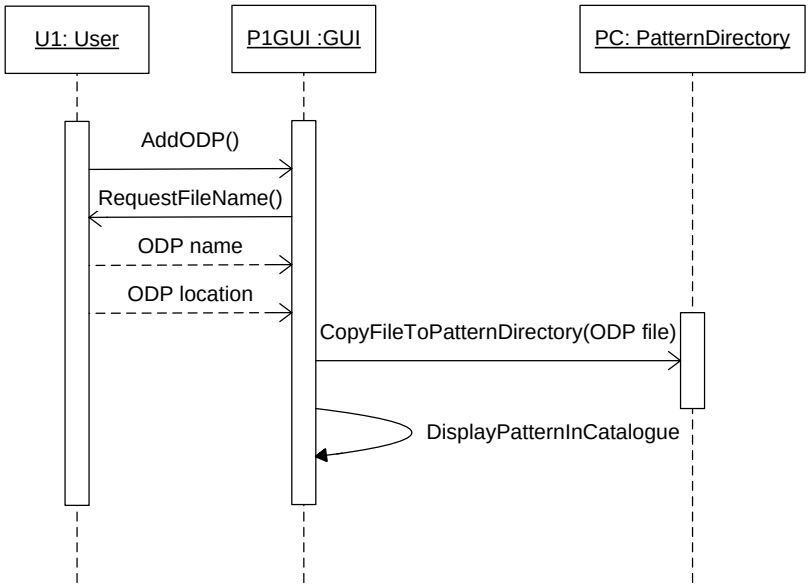

Figure 6-7 Add ODPs to pattern catalogue sequence diagram

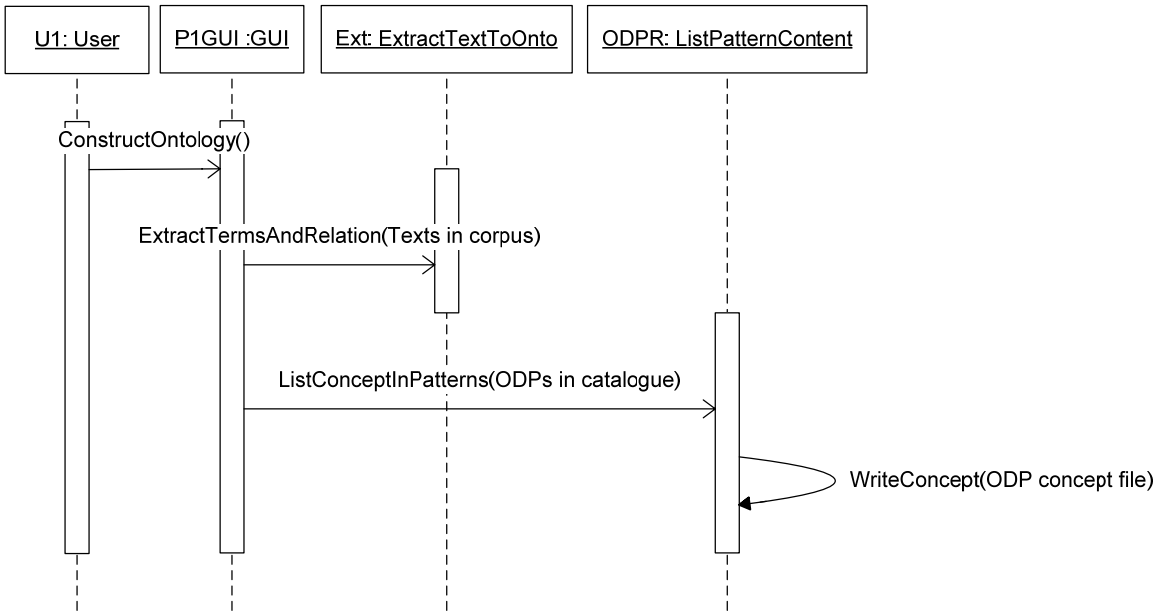

Figure 6-8 List concepts in ODPs sequence diagram

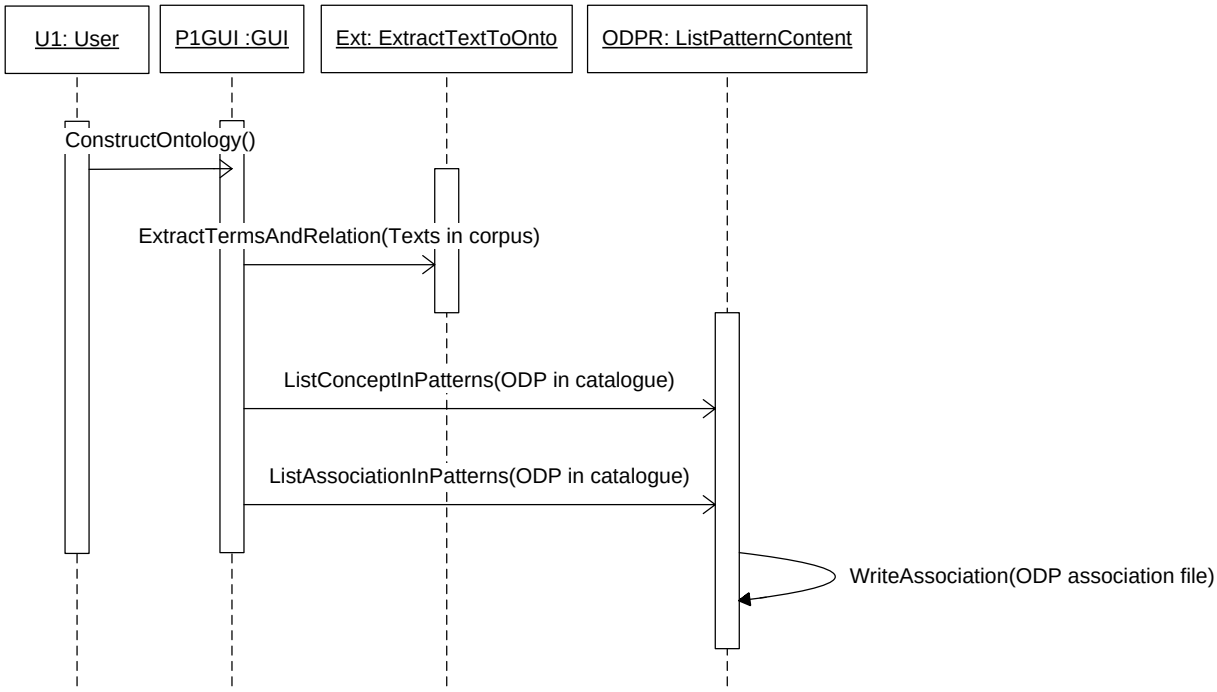

Figure 6-9 List associations in ODPs sequence diagram

6.1.3 Set string matching algorithm detailed design

The following sequence diagram describes the actions performed for choosing a string metric and set the string matching threshold.

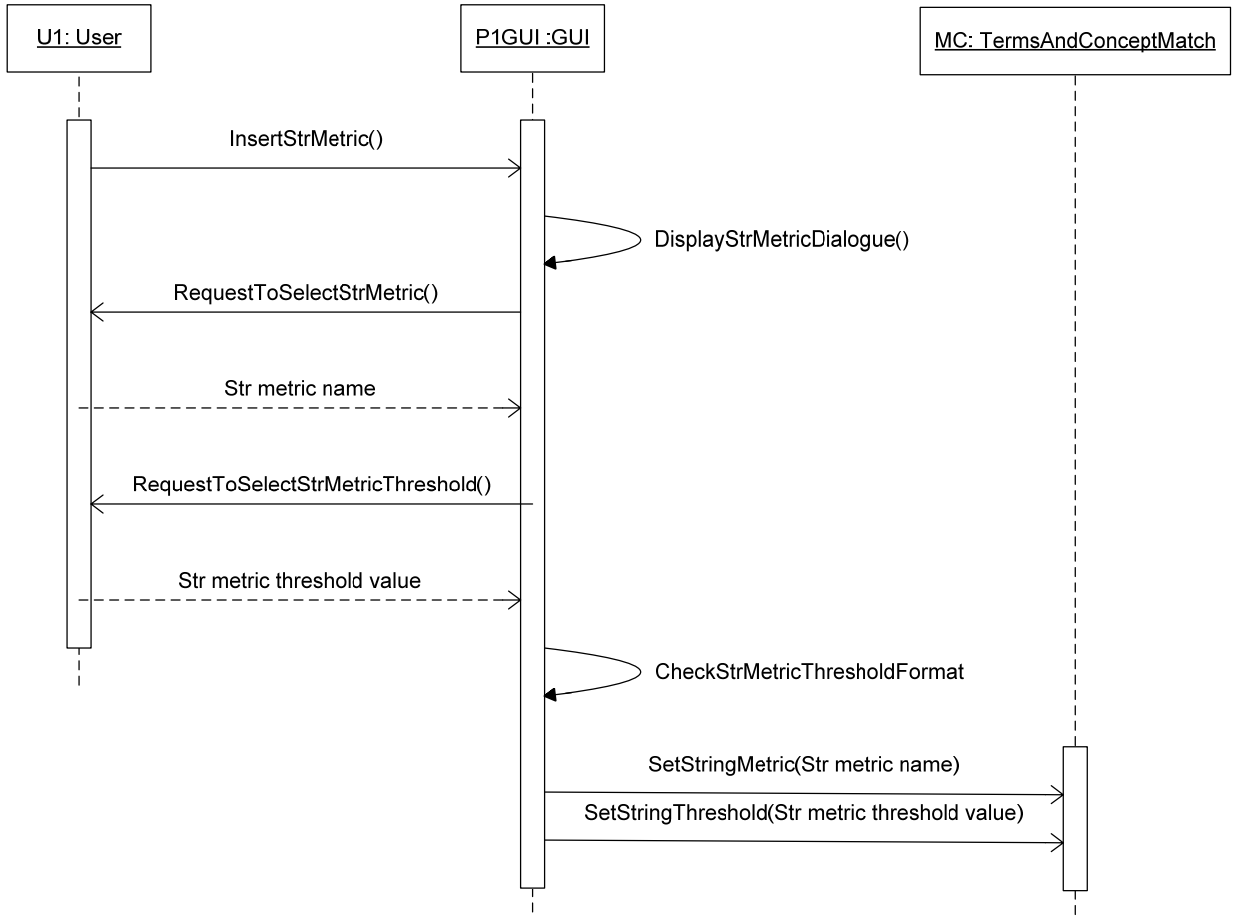

Figure 6-10 Set string metric sequence diagram

6.1.4 Set ontology design pattern threshold detailed design

The following sequence diagram describes the actions performed for setting a threshold value for the matching score of the ODPs selection.

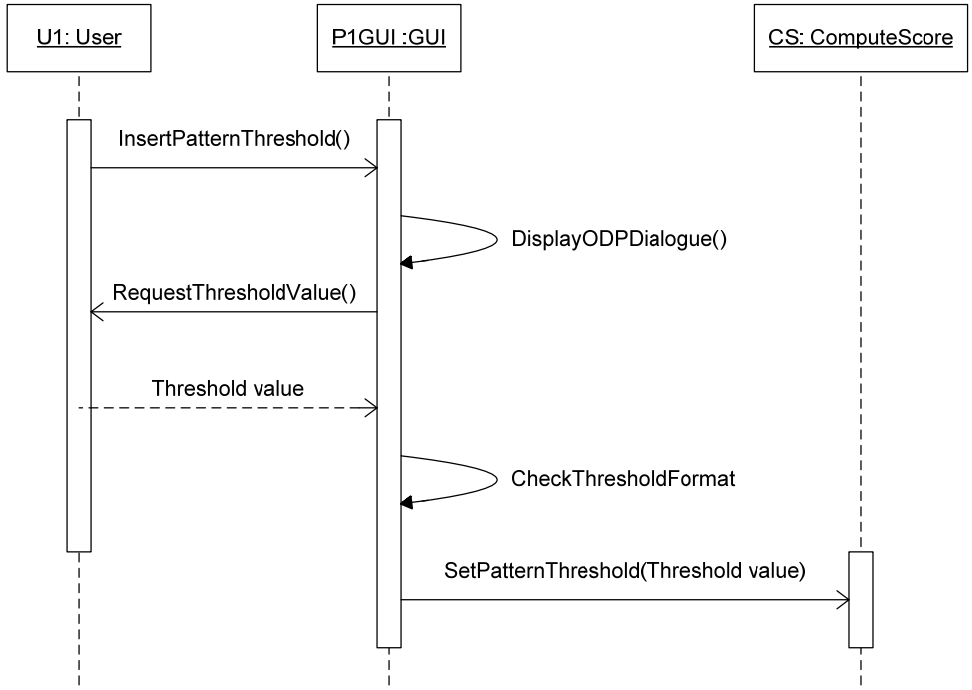

Figure 6-11 Set ODPs threshold sequence diagram

6.1.5 Score formula settings detailed design

The following sequence diagrams (Figure 6-12 and Figure 6-13) describe the actions performed for setting a formula for computing the matching score of the ODPs against the extracted terms and associations. In case the user chooses the “Linear combination” score formula, the later must enter values for the parameters “a” and “b” of the formula. If the formula chosen is “Automated weights values”, the user does not have to enter values for the parameters, since they are computed automatically. In this section we present the sequence diagram for the “Linear combination” formula and the “Automated weights values” formulas.

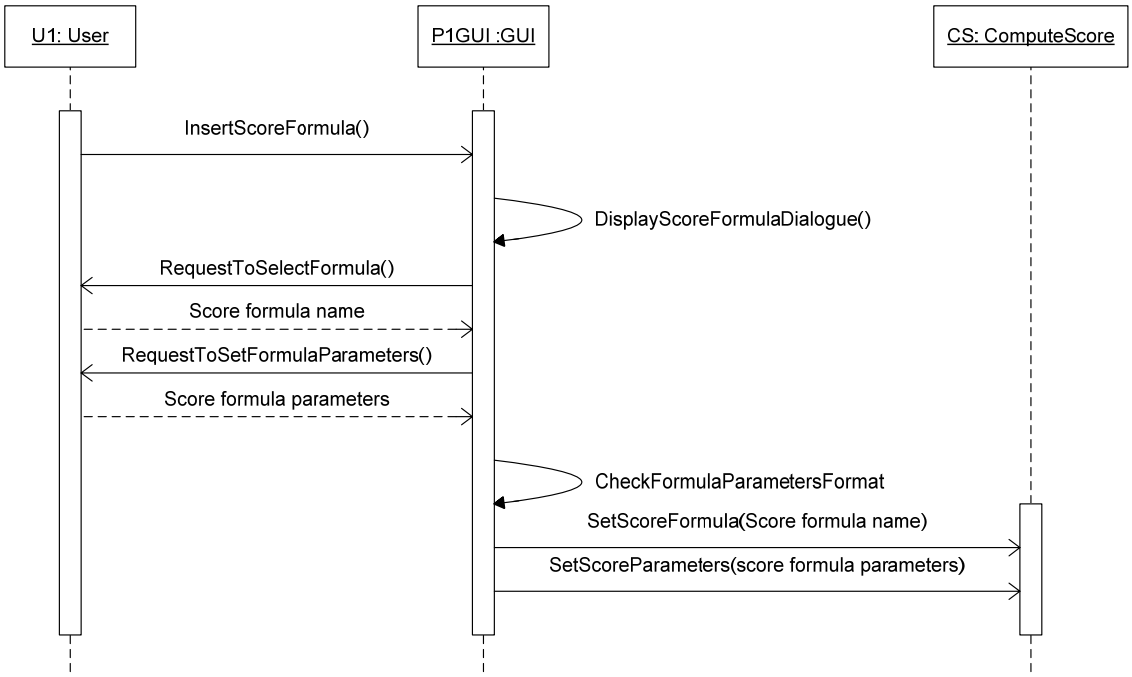

Figure 6-12 Setting the score formula "Linear combination" sequence diagram

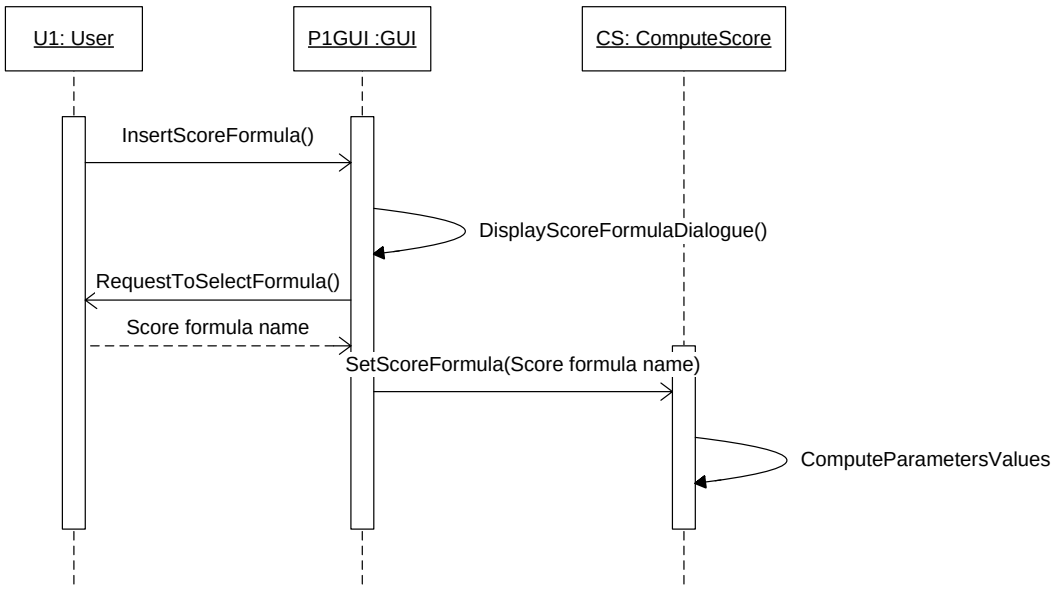

Figure 6-13 Setting the score formula "Automated weights values" sequence diagram