Utrecht University

Ontological Traceability using Natural Language Processing

A master thesis presented by

Edder de la Rosa Benitez

Submitted to the
Department of Organization and Information
in partial fulfillment of the requirements for the degree of

Master of Science

in

Business Informatics

Supervisors:
Prof. Dr. Sjaak Brinkkemper
Dr. Fabiano Dalpiaz

Utrecht, Netherlands, July, 2019

Dedication

I dedicate this thesis to my family with a unique feeling of gratitude to my mother, N. Leonora Benitez, whose education legacy is the most precious that I could have. My brother, Noel de la Rosa, has never left my side.

I also dedicate this dissertation to my many friends, Galia, Paola, César, Carlos and José Manuel, who has always been a constant source of support and encouragement during the challenges of my life.

Acknowledgements

I would first like to thank my thesis advisor, Sjaak Brinkkemper of the Department of Information and Computing Sciences at Utrecht University. The door to Prof. Brinkkemper office was always open to discuss ideas and debate theories about the thesis. He steered me in the right direction whenever he thought I needed.

I would also like to acknowledge Fabiano Dalpiaz, in the role of the second advisor for the valuable feedback with very effective and helpful comments on this thesis.

I am also grateful to the members of the GRIMM project group lead by Sjaak, where we met with fellow master students. To be part of this project was an enriching experience and provided a great environment to discuss and share ideas, especially to Sabine Molenaar, Tjerk Spijkman, Abel Menkveld and Jeroen Venema for sharing your knowledge and feedback.

Furthermore, I must express my very profound gratitude to my mother and my brother for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Finally, a special thanks to my friends, who supported me in the whole process of this study and incented me to strive towards my goal.

Ontological Traceability using Natural Language Processing

by

Edder de la Rosa Benitez

Abstract

The software development process is continuously evolving to find the right balance between the real-life problem side with the requirements engineering and the architecture of the software as part of the solution.

While the software architecture has to deal with different types of artifacts using different notation, the requirement engineering has, on top, the complexity of the ambiguity of the real world as most of its artifact use natural language to capture the required functionality. To find a solution to link these two areas and the artifacts among them is one of the practical problems that look to narrow the gap between the expected solution and the actual solution.

The use of ontologies is an accepted theory as a solution to connect these two areas and their respective artifacts through the creation of trace links. Such trace links can track a requirement among the different artifacts. However, the effort to generate such trace links can be time consuming and not beneficial for the time of development.

This research project aims to propose a solution to automate the software traceability through the use of a conceptual representation of a software artifact. An artifact in natural language is represented as a sub-ontology and find its match in a Product Ontology using natural language techniques and tools.

To demonstrate such theory, a proof-of-concept is created to extract an ontology from a software artifact and fins trace match and trace links among other artifacts. The results when testing the concepts is remarkable and suitable, and the level of acceptance in a segment of the software industry is quite promising.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Problem Statement and Context

Many practitioners have neglected software traceability over the last decades. Apparently because a misconception of the topic or ignoring the benefits. The effort, cost, discipline and time dedicated to this task to create and maintain trace links may contribute to this impression. It can be quite high in a rapidly evolving software system if performed fully manually (Cleland-Huang, Gotel, Huffman Hayes, Mäder, & Zisman, 2014).

The Center of Excellence for Software and Systems Traceability (CoEST) takes the definition of Software traceability as:

> "the ability to describe and follow the life of a requirement in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use), and through periods of ongoing refinement and iteration in any of these phases" (Gotel & Finkelstein, 1994)."

In a requirements engineering context, traceability is about the understanding of software requirements from high to low level, specifications, goals and the relation between different layers of information.

The requirement engineering community has performed a large part of traceability research to try to prove its importance for the software development and impact on the quality. Higher quality in software can lead to fewer inconsistencies, omissions and, therefore, fewer defects that can be beneficial for the development time, especially in testing and maintenance phases (Winkler & von Pilgrim, 2010).

There is, therefore, the need to reduce such inconsistencies that the requirement engineering is not only responsible. (Kaiya & Saeki, 2006) proposed an analysis method

based on domain ontology techniques that can be used to abstract concepts in requirements.

One of the goals of the requirements analysis is to develop automated high-quality requirements specifications supported by tools and methods. However, to automate this process is still a big challenge; most of the problems rely on the fact that requirements documentation are usually written in natural language (Kaiya & Saeki, 2006).

With more and more business adopting Agile development methods, practitioners follow an approach to formalise the software development process methodologically using semi-structured notations. Scrum artifacts such as product backlogs are examples of a semi-structured notation. A product backlog is a hierarchised list of requirements organised top-down from high-level requirements to the lower-level, e.g.jobs to be done, epics and user stories. These groups of requirements are still written natural language (Robeer et al., 2016) but with a particular standardised format.

Despite the benefits of these methods to standardise the software development, there are still gaps that inhibit the automation of the traceability of specified requirements with their respective atifact. An artifact is any program (i.e. source code), models (i.e. graphs, data models, process models, simulation models, etc.) related to the software development process. One gap identified is the ability to automate the tracking of a requirement and a target artifact to simulate human cognition process. The goal is to transform those requirements into a specific artifact with a more formal notation than natural language. For instance, when a developer needs to implement a new user story, fist, he needs to evaluate if there already exist the feature or group of features related to that requirement. If no documentation exists, then he needs to go directly to the code and identify the places where a change or new code is required. However, the developer could miss part of the code that not easy to reach, and the requirement not complete.

Natural language processing, text mining and machine learning techniques are research fields that have explored the extraction of information from unstructured text. Using terminology like lexicons, controlled vocabulary, thesauri and ontologies as knowledge resources. There is a clear need for more extensive models to associate linguistic information to ontologies. Those models should capture how concepts and relations are learned, allowing people to relate them to their own linguistic and cognitive system (Buitelaar, Cimiano, Haase, & Sintek, 2009).

There are new approaches classified as model-driven engineering (MDE) that share concepts and have different levels of abstraction. These approaches in software consider models as central artifacts and used not only for documentation purposes and represent different parts of the requirements. At the same time, they are a simplified part of a

system with the finality to share the vision of the stakeholders of the same type, either technical or functional (Da Silva, 2015).

Software artifacts are constructs with certain syntaxis, semantics and pragmatics. They contain notation, rules, exceptions and contexts where each linguistic element, statement or graphical element can or can not use. These artifact notations have different levels of formality that range from informal, semi-formal to formal. Natural Language, for example, is considered an informal representation of the reality as the language can be challenging to analyse. It does not follow a defined format besides the syntax and grammar. Opposite to this, state diagrams or Petri-nets follow a notation that has a precise semantics and, therefore, a formal representation of the reality that allows a formal analysis (Bass, Clements, & Kazman, 2013).

One formal approach in finding better interrelation between requirements and different software artifacts is through ontologies. Ontologies can act as an intermediate step between the syntaxis, semantics and pragmatics of the natural language and the target artifact and vice-versa, identifying the "ubiquitous software traceability" defined as:

> "Ubiquitous Software Traceability is software traceability that is always there, without ever having to think about getting it there, as it is built into the engineering process, traceability has effectively, 'disappeared without a trace.' Achieved only when traceability is established and sustained with near zero effort" (Cleland-Huang, Gotel, Huffman Hayes, Mäder, & Zisman, 2014).

The context of this study is placed using the Requirements Engineering for Software Architecture model (RE4SA) developed by researchers from Utrecht University as part of the GRIMM project. In the model, an epic story has its counterpart of a module and a user story with a feature 1.1.

Two initial problems we can identify. First, how to trace requirements in the software product and second, how to identify the dependencies of such traces between different artifacts.

Martens (2018) created an approach of traceability method through sub-ontologies using linguistic terms. A sub-ontology in this case is a sub-group of a product ontology. The similarity of terms is determined using a scoring technique when comparing semantic similarity among two artifacs. This semantic similarity is provided as an evidence of a trace link. However, it is not difficult to determine a match between two sub-ontologies when they are not in the same level of granularity.

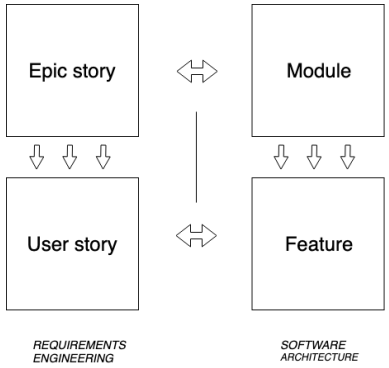

Figure 1.1: The RE4SA model by Utrecht University

A broadly accepted definition of an ontology used in this study is:

> "An ontology is an explicit specification of a conceptualisation. An ontology specifies the concepts, relationships, and other distinctions that are relevant for modelling a domain." (Gruber, 1995)

To build domain knowledge ontologies can be applied to different domains (Uschold & Gruninger, 1996). This domain is specific to the real-life problem that represents, for instance, health, research, finance, ICT, among others. The ontology domain provides a context idea about what kind of information is expected to find in an ontology. Thereby, the importance to know the context of the ontology to facilitate the classification of the concepts when similar terms are observed.

1.2 Research objective

The main goal of this research is to explore natural language tools and techniques that can be suitable for ontological traceability in a rapidly evolving software system environment using agile methods. The main hypothesis of this research is that there are linguistic patterns and models from an annotated text in agile atifacts that can facilitate the identification of ubiquitous software traceability through ontologies using supervised learning.

The main deliverables of this research are a conceptual model of an NLP analysis for ontological traceability and a proof of concept of the model that best suits the structure of agile artifacts that include epics and user stories. This proof-of-concept will provide evidence about the advantage and disadvantages of the model in the automation process of identifying traces between artifacts.

1.3 Research problem

With relative new approaches to reduce the risk of failure when developing software, it is common to find continuous and shorter development cycles. However, there is more pressure to deliver value to the stakeholders in each cycle. Such value is measured with the level of usefulness of the delivered increment in each cycle.

This continuous delivery also can imply to have more dependencies among requirements and among different software architecture artifacts.

Dependencies among requirements can be seen when a new requirement gets in conflict with a set of requirements previously implemented either because previous requirements are not completed or simply contradicts them. This is not very easy to see if there is no a good knowledge about the architecture of the product.

Dependencies among artifacts can be identified when, for instance, when a requirement requests a change in a feature or a set of feature but such features can depend from other features. For instance, an *save file* feature can be dependent of *edit file*. This dependency may not be clear as they can be at the same level but in a different location.

The following research problem aims to contribute to software engineering and stakeholders in a software development context that can be reflected in a higher value of software and, as a consequence, deliver value to the stakeholders.

> Improve software quality through automated ontological learning for software, supported by linguistic and neural network models and techniques to identify the existing ubiquitous traceability — all in the aim to streamline the software development process and deliver value to the stakeholders.

1.4 Research questions

This research elaborates over software traceability through ontological concepts that can be extracted from artifacts in natural language. In order to scope this study and reflect the goal, we define the following research questions:

> **MRQ:** "What kind of relevant ontological information can be extracted from annotated patterns in software artifacts using natural language processing that facilitates to trace links between them in the Software development process?"

Based on the goal of this research question, and to delimit the scope of this research, it is divided into five sub-questions:

RQ_1**:** "How can relevant ontological information from system development artifacts written in natural language be extracted?"

RQ_2 "What patterns from an annotated text in system development artifacts written in natural language can be identified?"

RQ_3**:** "What NLP techniques are suitable to identify sub-ontologies in an annotated text in software artifacts written in natural language text using NLP?"

RQ_4**:** "How can the identified sub-ontologies be related to a software development artifact ontology?"

RQ_5**:** "How can a direct trace link be derived from a sub-ontology to specific system development artifacts?"

1.5 Vision

The vision of this thesis and the proof-of-concept is to explore the integration in a DevOps environment using requirement engineering, software architecture and agile methodology. This exploration is performed using GitLab as a DevOps platform and how to integrate natural language processing techniques to find traces among artifacts as shown in 1.2. The integration of software architecture applications is not part of this study but described as part of the vision. The boxes indicated in light grey in the right-hand side of the figure represent the current platform landscape. The boxes in the black background in the left-hand side (annotation, pattern recognition and PDO information extraction) represent the vision areas of future development and the boxes with a light blue background are the main objectives of this research.

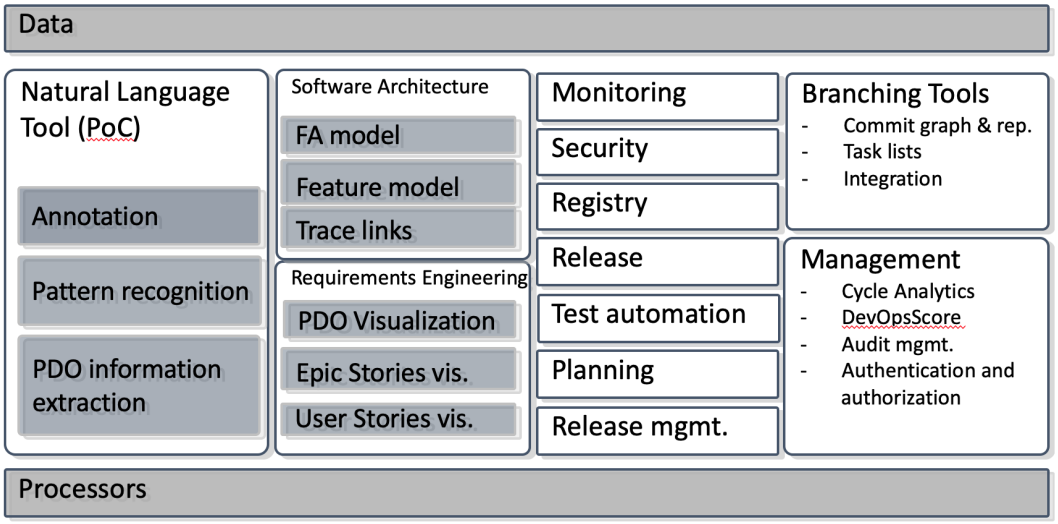

Figure 1.2: Vision of landscape in a GitLab environment.

Chapter 2

Research Approach

To help to answer these questions, an explorative analysis in the current literature is performed together with a Scrum artifact to answer RQ1 and RQ2. Two artifacts are produced as a result of this study. The first artifact is a conceptual model that comprises the patterns that a machine can interpret in order to find matches and traces to a defined ontology from a specified ontology domain to answer RQ3. Several artifacts are created to automate and support the analysis.

The second artifact is a proof-of-concept that contains this model or can be trained with a different model to find the sub-ontology related and help to answer RQ4. Finally, data analysis is performed to derive the direct link between the source artifact and the target artifact, for instance, a scrum artifact (i.e. user stories) and an architectural artifact (functional architecture model, feature model, etc.).

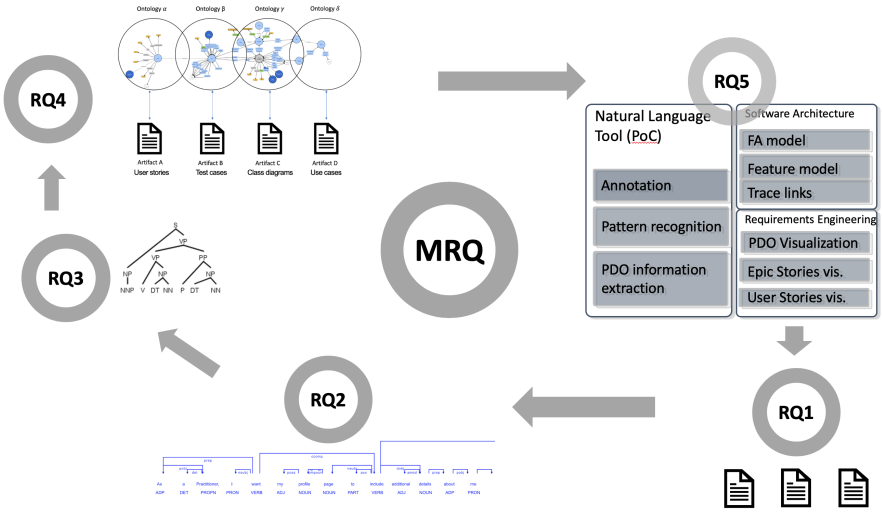

Figure 2.1: Research questions in the problem context. Annotations from spaCy.

2.1 Research method

In order to answer the research questions part of the engineering cycle will be performed described by Wieringa (2014). The engineering cycle consists of four main steps: treatment design, validation, implementation and evaluation (see figure 2.2). The treatments for this study will be created. This research focuses on a problem investigation, treatment design and validation. As such, this study is not intended to conduct the whole engineering cycle, but only the three steps mentioned.

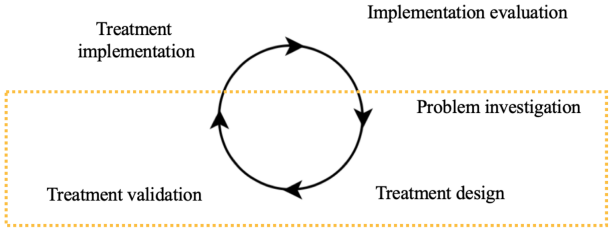

Figure 2.2: The engineering cycle adapted from Wieringa(2014). Only the indicated part is used in this research

To validate this study hypothesis, this study carries out one cycle of the empirical cycle as stated by Wieringa (2014). Figure 2.3 provides an overview of the empirical cycle applied to this research, it is divided into five phases that can be cyclically executed. This study performs only one cycle and analyses the results. The following sections a detail of each phase is explained more in detail using a process deliverable diagram (PDD) notation as described by van de Weerd and Brinkkemper (2009) to facilitate the formalization of the study. The activities or steps are described in figures 2.5 to 2.9 and tables 2.5 to 2.9.

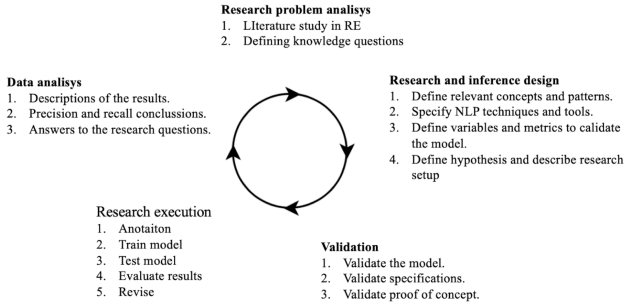

Figure 2.3: The empirical cycle adapted from Wieringa(2014).

The main goal of each research question can be seen as follows.

RQ_1**:** "*How can relevant ontological information from system development artifacts written in natural language be extracted?*"

The first challenge faced is to define how relevant ontological information will be extracted from the text. Relevant information is that information that is usefully towards the identification of main concepts and relations. As a first step, it is necessary to determine what type of artifacts are more suitable to get the expected goal, Selected artifacts need to be in a natural language notation. For this study jobs, user stories and epic stories will be used as corpus. The result of this process is to generate a corpora ready to be analyzed as shown in table 2.1 and the nlp method suitable to extract ontological information.

Input	Process	Output
Software development artifact	Text extraction from the artifact relevant to this study and predefined language. Determination of natural language method and technique that helps to provide relevant expected information.	Text in natural language notation in defined language (corpora). Natural language method and technique.

Table 2.1: Extraction process parameters and processing.

RQ_2 "*What patterns from an annotated text in system development artifacts written in natural language can be identified?*"

Extracted text from selected artifact is most of the time in an informal or, in the best case, in a semi-formal notation form. To solve this problem annotation in text is used to add metadata information to text. This metadata is required to process natural language and is a basis of this study as it is an essential part of processing text. As described in table 2.2 a corpora is obtained from the artifact and annotation process is performed. As a result,

Input	Process	Output
Corpora	Annotation	Annotation metadata

Table 2.2: Text annotation process parameters and processing.

In order to facilitate the understanding of natural language to a machine through software, it is necessary to annotate this text. In the case of scrum artifacts like epics and user stories, these have certain notation rules and structure. The assumption here is that there exist certain patterns that can be identified from an annotated text in those artifacts. Each artifact can be represented as an ontology and each ontology artifact is part of a product ontology in a product software development as shown in figure 2.4.

> RQ_3**:** "*What NLP techniques are suitable to identify sub-ontologies in an annotated text in software artifacts written in natural language text using NLP?*"

The use of different techniques and tools of natural language processing are the main means to solve this question. In this case, most frequent patterns need to be identified in order to identify the affinity of text to an ontology using automatic annotation in text. Such annotation has to be grouped and classified in order to expose these patterns. These patterns are defined as classes and comprise one or several tags. spaCy software provides different NLP libraries that combined with development languages like Phyton provide adequate means for this purpose. Table 2.3 describes the parameters to process this step.

The analysis is carried out in several iterations. Each iteration consist in the creation of an artefact and an analysis of the information obtained to identify relevant patterns. If there is enough information from the patterns obtained then iterations conclude and the flow of the research continues. One hypothesis of this study is that at least two annotation techniques are necessary to to find relevant ontological information from a software artefact. Therefor, at least two iterations are expected from this analysis.

Input	Process	Output
Annotation metadata	Pattern recognition	Text ontological affinity

Table 2.3: Pattern recognition process parameters and processing.

> RQ_4**:** "*How can the identified sub-ontologies be related to a software development artifact ontology?*"

Natural language processing annotated text can be integrated to a certain sub-ontology that leverage from external knowledge. However, it is not completely clear how to relate such annotation into respective sub-ontologies. As shown in figure 2.1

this question explores this aspect and aims to generate a proof-of-concept that helps to demonstrate the use of this research to answer it. The input for this is the intended ontology generated from the natural language processing and categorized as a sub-ontology. This sub-ontology is mapped to the general ontology to identify the elements of this sub-ontology into the general ontology.

Input	Process	Output
Ontological affinity	Semantic classification trees and comparison	Intended ontology

Table 2.4: Ontological classification process parameters and processing.

RQ_5**:** "*How can a direct trace link be derived from a sub-ontology to specific system development artifacts?*"

RQ5: Once trace matches are identified to a related sub-ontology it is possible to derive the link between both artifacts.

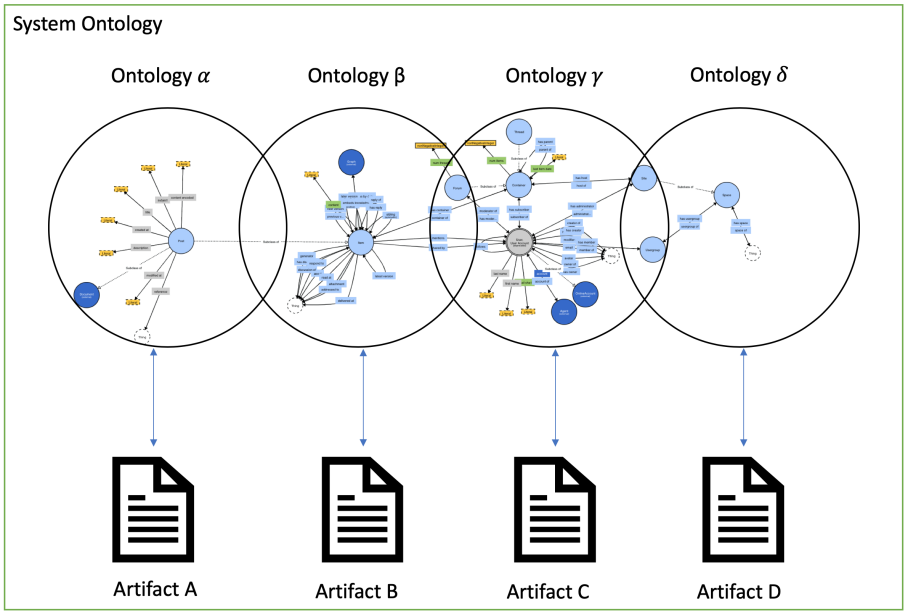

Figure 2.4: Artifact ontologies as part of a product ontology.

2.2 Problem investigation

This study starts with a problem research analysis in order to understand further about the focus of the study performing a literature study with different levels of refinement

in order to select relevant literature, identify missing knowledge, build a theory and create the research questions. Having natural language processing, ontology traceability and requirement engineering as discovery fields and the gaps traceability in requirement engineering as the focus of the problem. Figure 2.5 shows a graphical description of this phase with its respective deliverables.

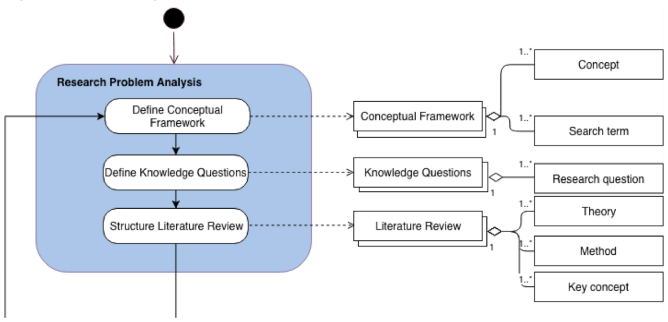

Figure 2.5: PDD of the first phase. The main deliverable of this phase is a literature review with the state of the art theories, methods and key concepts relevant for this study.

MS ID	Activity	Sub-activity	Description
MS1	Research Problem Analysis	Define conceptual framework	Structure a framework of how to organize ideas related to traceability problems and their relation with ontologies and NLP.
		Define knowledge questions	Identify a gap in the requirement engineering and define the research questions.
		Structure literature review	Support the identified gap using literature and identify research trends and theories.

Table 2.5: Activity table related to the PDD in problem investigation phase.

2.3 Research and inference design

In this phase a design of the model using the identified NLP techniques and supported by tools is carried out. In order to identify the accuracy of the model, several measures

and metrics are specified. It also specifies what artifacts are used, the necessary required datasets wither to train and to execute the model.

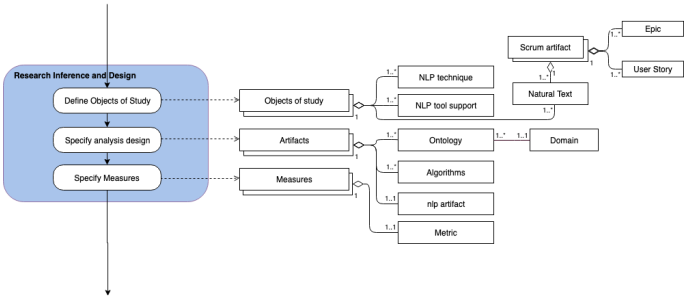

Figure 2.6: PDD of the second phase.

MS ID	Activity	Sub-activity	Description
MS2	Research inference design	Define objects of study	Identify techniques and methods relevant for this study,software artifacts and most suitable tools to support the study.
		Specify model	Using the selected techniques and method, patterns are preliminarily identified, and a conceptual model is created.
		Specify measures	Measures and metrics are defined to calculate the accuracy of the model.

Table 2.6: Activity table related to the PDD in inference design phase.

2.4 Validation

The main objective of this phase is to reduce the risk of validity with regards to the objects of study, models, measurements and design inferences and make sure that all elements are in line to the purpose of the research and relevant to achieve the main goal of the study.

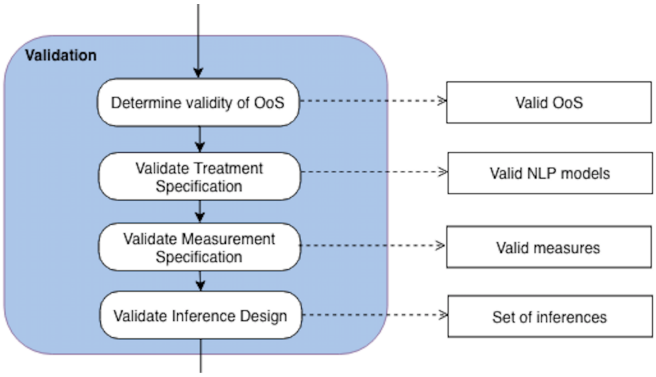

Figure 2.7: PDD of the validation phase.

MS ID	Activity	Sub-activity	Description
MS3	Validation	Determine validity of OoS Validate treatment specification Validate measurement specifications Validate inference design	Support why the selected objects are more suitable for the study. Request ML and linguistics experts to validate the model. Ensure that measures actually measure for what is intended to. Request feedback from the research community to see if the inference design is valid or not.

Table 2.7: Activity table related to the PDD in validation phase.

2.5 Execution

This phase, as shown in figure 2.8, is a combination with the Annotation Development Process cycle model (MATTER) from Pustejovsky and Stubbs (2012) into the empirical cycle. The goal of this phase is to build a model that can be used to identify ontologies, links and traces from jobs, epics and user stories to later on implemented in a proof of concept to trace the information contained in those artifacts and describe those ontologies and relations and demonstrate the feasibility of the model.

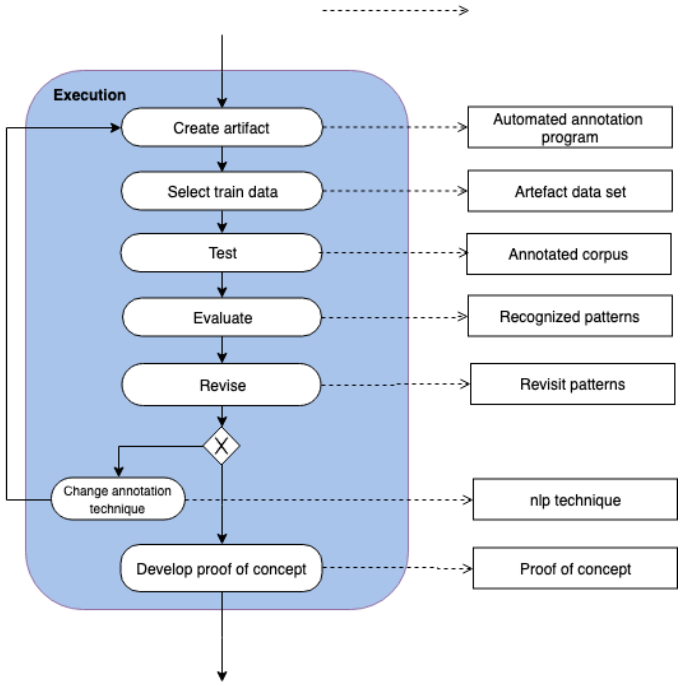

Figure 2.8: PDD of the execution phase.

MS ID	Activity	Sub-activity	Description
MS4	Execution	Annotate	Create an annotation scheme that encodes structural descriptions and properties of the text. A corpus is generated that can be processed by a machine.
		Train	Selected algorithm is trained over an annotated corpus with a target ontological model.
		Test	The algorithm is tested with the data set.
		Evaluate	A standardized evaluation of the result is carried out.
		Revise	Revisiting the model and annotation specification and make more robust and reliable the use of the algorithm.
		Change model	Adapt the model to improve its accuracy and robustness.
MS5		Proof of concept	Using the final model, a proof of concept is developed to make possible and interaction with the model.

Table 2.8: Activity table related to the PDD in execution phase.

2.6 Data analysis

This phase, described in figure 2.9, is comprised of four activities dedicated to describing the results obtained from the measures, evaluate if it is possible to make generalizations to the hypothesis, make the possible conclusions with regards to the objective of the study and provide an answer to the knowledge questions.

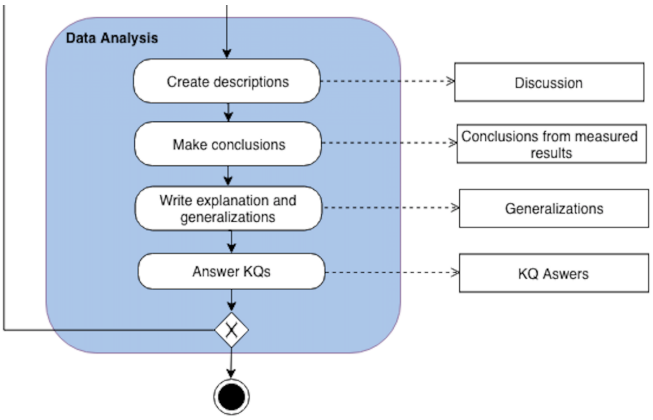

Figure 2.9: PDD of the data analysis phase.

MS ID	Activity	Sub-activity	Description
MS6	Data Analysis	Create descriptions	A set of descriptions are created to state a discussion based on the results.
		Make conclusions	Final conclusions are generated from the data analysis.
		Write an explanation and generalization	Explanations of generated conclusions and definition how easy they can be generalized to other cases.
		Answer KQs	Answers to knowledge questions are provided to conclude the study.

Table 2.9: Activity table related to the PDD in data analysis phase.

2.7 Literature research protocol

The literature research is divided into four steps in order to identify correctly the essence of the study an is reflected in figure 2.10. First, terminologies, group researchers, conferences technologies and initial papers are identified in regards to the area to be explored. There is an initial approach to NLP tools from the literature that starts to be explored at the same time and that can support the present study.

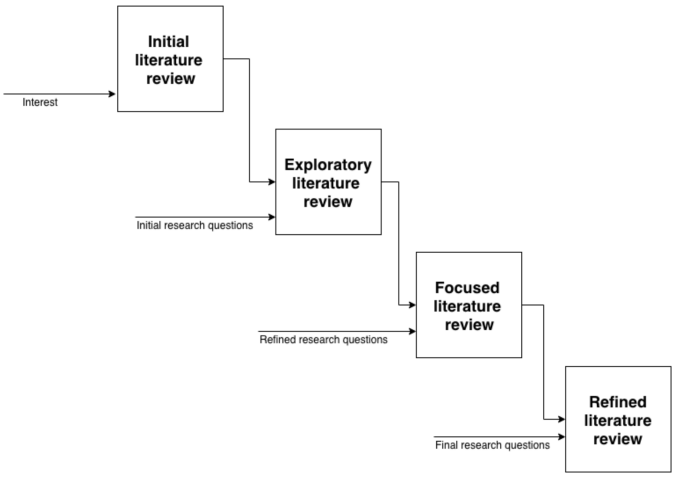

Figure 2.10: Literature research process, adapted from Liston (2006).

The second phase is an exploratory where specific theories are explored and find relations between them in order to find common concepts, theories and models that can share common characteristics related to the topic explored. The third step is a focused literature review with refined research questions. For this review, several search concepts are identified to find specific literature. This search was based in following search terms:

- “Natural Language Processing” AND (”Ontology learning” OR “Software traceability”).
- Ontology AND “Model-driven engineering” AND “Software traceability”.
- “Natural Language Processing” and “Requirements Engineering”.
- “Semantic trees” or “Syntactic trees”.
- “Ambiguity” and “Word2vec”

For the fourth step, there is already an idea of the research questions and, therefore, it is possible to do a refinement of the literature found. In order to do this refinement,

the literature is classified and selected. The classification is based on if it is specific and highly related to the topic or if it is generic and background information. For the first classification, there is not much literature. In order to ensure the quality of the research, this literature is analyzed and approved or provided by a specialized researcher in the area. For the second classification, the literature with more than 150 citations is considered for the research. This literature should have been published in a recognized journal or conference.

2.8 Research Plan

This study is organised in two main phases and five deliverables. The first phase covers what is called pro problem analysis of the research. The second phase includes the research in four sub-phases that integrates the inference design, validation, execution and data analysis as previously explained in the research method.

2.8.1 Phase I

The first phase is dedicated to integrate a framework to collect and clarify concepts, generate the research questions and structure a literature study. A project proposal integrates the first milestone with identified gaps in the research literature and state theory of how to reduce those gaps.

2.8.2 Phase II

The second phase of this study comprises an inference design, validation, execution and data analysis as primary activities. During this phase, four milestones are delivered. The first milestone (MS2) contains the design of the study providing preliminary conceptual models and the measures used. The second milestone (MS3) is delivered once the objects of study, conceptual model, measures and inferences. A third milestone (MS4) considers a conceptual model that will be analysed after one or more cycles of the annotation development. Once analysis reaches the expected goal, a proof-of-concept (MS5) taking the conceptual model is created to identify ontologies, relations and, as a consequence, traces.

Finally, the last milestone is delivered containing the final discussion, analysis, conclusions and answers to the research question. This last deliverable concludes this study and is handed over for the respective revision.

Chapter 3

Literature review

This literature study is explores the most relevant concepts identified in this activity. The table 3.5 shows an extract of the main hypothesis, key concepts and results of each research study. This literature will be considered with some additions during the process following the same protocol of evaluation. This literature can also be found in the bibliography section.

3.0.1 The RE4SA model

As introduced before, the context of this study is placed using the Requirements Engineering for Software Architecture model (RE4SA) (Blessinga, 2018) shown in figure 3.1. This model has three levels of granularity and a counterpart in both sides, requirements engineering and software architecture.

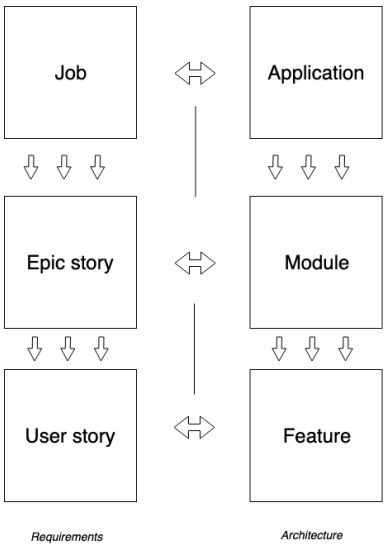

Figure 3.1: The RE4SA model

From this model, a Job to be done is represented as the application in the architecture side.

At the same time, an epic story is derived in user stories that increase the level of granularity of the requirement. In case of a module and feature, a group of features integrates a module.

We define an epic as:

> "A large user story that cannot be completed in one iteration of the Agile method, follows a predefined format template, one motivation, is triggered by one problem to be solved and describes an expected outcome."

In order to perform an analysis of artefacts in natural language, We define a user story as:

> "A sentence statement in a defined template that is composed by one role, one means and a goal or benefit. A means is composed by at least one action verb, a subject, a direct object and optionally adjectives and indirect objects."

3.1 Ontologies

Ontologies have been adopted as a suitable representation of concepts and its relations, providing considerable support in the identification of traces. Lucassen et al. (2015) propose an approach to identify links in existing source code and documentation by inferring implicit relationships between concepts and the use of ontologies. Liu et al. (2011) and Maedche and Staab (2001) created a way to automate the process of learning with ontologies through a defined architecture and machines.

One of the main problems identified in requirements engineering is that it deals with gathering functionality desired by the respective stakeholders. Stakeholder has different understandings and point of views of the problem that can generate a certain level of ambiguity between terminologies used. Happel and Seedorf (2006) observed that ontologies could be used as an approach to describe such requirements that are mostly in natural language. An ontology model provides a better understanding of the domain and provides aid in the requirements management and traceability. Furthermore, authors like Rolland and Proix (1992) and Bernstein and Kaufmann (2006) highlight the necessity and benefit of tool support in the area of requirements engineering using linguistic approach methods.

Zhang et al. (2008) proposed a method to automate the finding of traceability links between source code and natural language documents. An ontology-based method designed to model patterns and their relations between entities in an ontology. This approach uses standard OWL-DL for capturing the semantics in a domain discourse. Cleland-Huang et al. (2014) provides some examples of what is the benefit of identifying traces in the artifacts. For instance, the identification of potential side effects from newly introduced user stories on existing ones and in the whole software. At the same time, he describes a process perspective in the identification of traces with a traceability life-cycle.

Martens (2018) presents a systematic approach for the creation of links using an ontology and a set of linguistic terms extracted from software artifact instances. The extraction of these linguistic terms is fundamental for this study. Natural language processing provides specific techniques to support different real-life problems. Problems like questioning and answering systems, memorisation, machine translation, speech recognition and document classification (Pustejovsky & Stubbs, 2012) (Happel & Seedorf, 2006).

3.2 Natural Language Processing

Chowdhury (2003) started to sport the importance of the study of NLP with previous studies of NLP text systems by Haas (1996), Mani (1999), Smeaton (1999), and Warner (1987). In the area of natural language processing Cambria and White (2014) divides NLP into three different studies, syntactic NLP, semantic categories and pragmatics. Hirschberg and Manning (2015) published an analysis of the actual status with NLP and highlighted the challenges and difficulties of its use related to semantics, contexts and knowledge.

3.2.1 Syntactic NLP

The first category, syntactic NLP, involves categorisation based on unambiguous words for keyword spotting. This category refers to a lexical affinity using probabilistic methods to determine the affinity of arbitrary words already spotted and the use of statistical methods like machine learning algorithms to support the affinity determination. Berland and Charniak (1999) proposed a method to extract parts of objects using statistical methods. However, this approach, by itself is minimal. Keyword spotting relies in its reliance upon the presence of obvious words that only surface features of the prose. For instance,

a document related to a noun may not reference this noun in a sentence and might not be retrieved by a keyword-based search engine.

The basic unit of the syntactic is the *word*. We take the definition from Crystal (2011) as:

> "A unit of expression which has universal intuitive recognition by native-speakers, in both spoken and written language."

Words can be classified according to the properties that share between them. In this case, we take the definition of *word-class* as:

> "An application in linguistics and phonetics of the general use of this term, to refer to a set of entities sharing certain formal or semantic properties (Crystal, 2011)."

One word classification most commonly used is the categorisation of such words according to their grammatical properties called part-of-speech.

A combination of different words that ordered with a syntactic structure from a grammar of certain language form a sentence. We take the definition of a sentence as:

> "The largest structural unit in terms of which the grammar of a language is organized (Crystal, 2011)."

3.2.2 Semantic NLP

The second category relates the semantics, where different techniques that leverage in external or internal knowledge like intrinsic semantics of document methods. Gabrilovich (2009) published a method for semantic interpretation of unstructured natural language texts. Gabrilovich (2009) built an NLP engine that utilises a specifically developed context-free lexicon, lexemes though tokenization and interpret these lexicons. Williams (2013) analyzed how to obtain identifier names using simple syntactic patterns and syntax identifiers from natural English inconsistent ways using part-of-speech. In this area, Rindflesch and Fiszman. (2003) proposed a method to interpret linguistic structures to identify concepts and its relation with a more general idea.

3.2.3 Pragmatics NLP

The third category refers to the semantics that goes from lexical to compositional semantics involving different methods like disclosure structure, argument-support hierarchies

and common-sense reasoning (AI). Maynard et al. (2008) provide promising results from different experiments in the area of information extraction from text. Usually combining at least two NLP techniques and rule-based approaches or machine learning using defined measures and metrics for its validation.

Pustejovsky and Stubbs (2012) defines pragmatics in natural language annotation as:

> "The study of how context of text affects the meaning of an expression, and what information is necessary to infer a hidden or presupposed meaning."

In this study, the context is the software development using RE4SA model and the required software artefacts by the model.

3.2.4 Natural Language Understanding

Pattern recognition from natural text is an important task in the process of NLP understanding. Muter et al. (2019) described linguistic patterns through an analysis of a large collection of user stories by identifying the action verbs and a template at user story task level.

Kuhn and De Mori (1995) built a method to obtain natural language understanding using semantic rules. These rules are trained, and classification is performed using semantic classification trees, decision trees and machine learning techniques. Soricut and Marcu (2003) supports the approach of use parse trees, lexical, syntactic trees and discourse parser to create a discourse model.

Soricut and Marcu (2003) provides two probabilistic models to identify elements of discourse building discourse parse trees. These parse trees are used to segment the discourse dividing the problem into lexicalized syntactic trees and discourse trees. Once the segment of discourse is identified, then it can be related to the particular part in an ontology.

A semi-automatically extract domain terminology is used by Quirchmayr et al. (2018) to extract relevant feature information from an artifact, in this case, user manuals. NLP techniques are applied to such arifact to obtain feature-relevant information. The results are measured, and different techniques evaluated.

3.2.5 Convolutional neural networks in NLP

One concept used recently by natural language processing is the use of neural networks. The need to solve complex problems and find a solution based on the human brain works generating models of cognition is the basis of this concept.

The idea of a neural network is to mimic *qualitative reasoning* by manipulating encoded symbols (A. S. Lapedes & Farber, 1988). The brain automatically does this qualitative reasoning by reasoning all elements in real-world such as space, time propose, time etc.

To mimic such brain functionality, a neural network divides each element of the real world into different hidden layers. Each layer contain a group of nodes or neurons that their task is to receive an input, perform some calculations and produce an output.

This architecture gives the flexibility to perform simultaneous tasks that can work in the same problem in parallel.

The use of neural networks has been significantly accepted in the creation of statistical models instead of using the traditional linear model due to its good performance. A. Lapedes and Farber (1987) used this nonlinear perspective with neural networks in the prediction of non-linear signals. Churchland and Sejnowski (1990) aboard this problem from a philosophical perspective and the use of a neural network model in speech processing, where you have a first layer that is a word in the real world. a *hidden layer* that performs a transformation and the output layer that provides the results.

Neural networks are recently used to solve different real-world problems like face recognition (Lawrence et al., 1997), visual document analysis (Simard et al., 2003) and most recently to model natural language sentences (Kalchbrenner et al., 2014) or classify them (Kim, 2014) and many other applications. All these examples share a common architecture as convulsive CNN.

A convolutional neural network (CNN) has several layers with several nodes or neurons. When a neuron receives some input, this performs the designated calculation, and the output serves as in input to another neuron as can be seen in figure 3.2

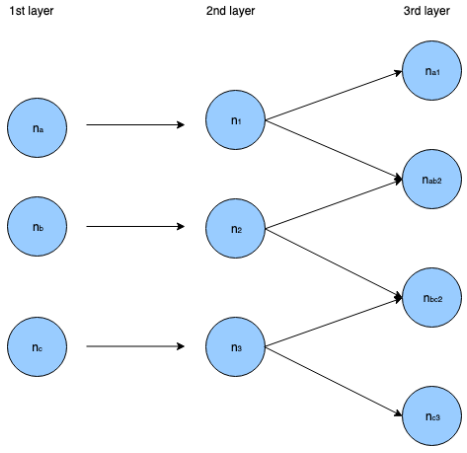

Figure 3.2: Convolutional neural network representation

The use of a CNN can be very convenient in solving the problem of similarity of terms. Two terms can be similar but may not refer to the same thing, therefore, ambiguous.

The input of a CNN should be something that can be used to do further calculations. The use of bag-of-words can handle this problem.

In the bag-of-words, each word in real life is represented as a vector. This vector is in a repository of vectors that build vocabulary. As a result, you have a repository of vectors containing the vocabulary.

Mikolov et al. (2013) proposed a model with an architecture of vector representation known as continuous bag-of-words (CBOW) and skip-gram models (one the opposite to the other). These models try to maximise the classification of a word-based of another word in the sentence; in other words, predict the probability of a word based on the context or vice-versa. These models use a CNN to its processing, and the results provided seems acceptable in terms of performance of the architecture model.

3.2.6 Tool support

There are different libraries to support NLP activities in which we can find StanfordNLP toolkit, NLTK for python, SyntaxNet and spaCy. StanforNLP provides a state-of-the-art libraries developed in Java and widely used in the NLP comunity. CoreNLP architecture provides different techniques form annotated objects like tokenization, sentence splitting, part-of-speech tagging, morphological analysis, entity recognition, syntactic parsing, conference resolution, sentiment classification (Manning et al., 2014). This tool also provides libraries in Java for visualization of the features, for instance, POS tagging, entity recognition or syntactic parsing between others.

Language processing feature	Functionality
Tokenization	Tokenizes the text into a sequence of tokens.
Sentence Splitting	Splits a sequence of tokens into sentences.
Part-of-speech tagging	Labels tokens with their part-of-speech (POS) tag, using a maximum entropy POS tagger.
Morphological analysis	Generates the lemmas (base forms) for all tokens in the annotation.
Entity recognition	Recognizes named (PERSON, LOCATION, ORGANIZATION, MISC) and numerical (MONEY, NUMBER, DATE, TIME, DU- RATION, SET) entities.
Syntactic parsing	Provides full syntactic analysis, including both constituent and dependency representation, based on a probabilistic parser.
Conference resolution	Implements mention detection and both pronominal and nominal conference resolution.
Sentiment	Sentiment analysis with a compositional model over trees using deep learning

Table 3.1: CoreNLP language processing features.

Natural Language Toolkit (NLTK) is a tool that works together with Python with advanced algorithms for NLP. It owns distinct qualities like simplicity, consistency, insensibility and modularity. It has been broadly used in universities and different research

projects. Table 3.1 lists the most important modules. This tool also provides visualisation libraries. However, these are not as sophisticated as in CoreNLP.

Language processing feature	Functionality
Accessing corpora	Standardized interfaces to corpora and lexicons
String processing	Tokenizers, sentence tokenizers, streamers.
Collocation discovery	t-test, chi-squared, point-wise mutual information.
Part-of-speech tagging	n-gram, back-off, Brill, HMM, TnT.
Classification	Decision tree, maximum entropy, naive Bayes, EM, k-means.
Chunking	Regular expression, n-gram, named entity.
Parsing	Chart, feature-based, unification, probabilistic, dependency
Semantic interpretation	Lambda calculus, first-order logic, model checking.
Evaluation metrics	Precision, recall, agreement coefficients.
Probability and estimation	Frequency distributions, smoothed probability distributions.
Applications	Graphical concordance, parsers, WordNet browser, chatbots.
Linguistic fieldwork	Manipulate data in SIL Toolbox format.

Table 3.2: NLTK language processing features.

spaCy is an open-source library for NLP written in Python and Cython. It provides libraries oriented to production usage. Therefore, the flexibility of integration with other technologies is higher than with research-oriented libraries. Table 3.3 lists the main features of this technology. This tool also provides some basic visualization in features like POS, tokenization, entity recognition and dependency parsing. Nevertheless, these are just necessary libraries that offer a simply plain output the requested feature, but it can be easily integrated with a Python environment.

Language processing feature
Neural network models
Integrated word vectors
Multi-language support
Tokenization
Part-of-speech tagging
Sentence segmentation
Dependency parsing
Entity recognition

Table 3.3: spaCy language processing features.

Al Omran and Treude (2017) made a series of experiments within state-of-the-art NLP libraries assuring the level of agreement between them to assign a part-of-speech and the accuracy. The libraries taken for this study are Google's SyntaxNet, Stanford CoreNLP, NLTK Python library and spaCy using three different artifact sources related to Java programming language: StackOverflow, GitHub ReadMe files and Java API documentation.

The findings from the study, the source with the highest ratio of correctly identified token POS tags came from StackOverflow. For general POS tagging, NLTK achieved the highest agreement in tokenization. Manual annotation POS tags reached 92% compared with the NLTK. For specific POS tags, the manual annotation comparison yield the highest agreement in the case of spaCy with a 90% of agreement, for instance the ord "Length" in a sentence "statement String Length", both spaCy and NLTK agreed to take it as a noun (NN) while other libraries tagged the word as a proper noun (NNP).

The general finding from the study showed that spaCy library provides the best overall performance on the data analyzed, table 3.4 shows a comparison table about the accuracy of each library and manually annotated text. This study provides relevant information to determine what tool can be more suitable for the research in hand.

Source	Comparison	Identical tokens %	Identical token/POS_g	Identical token/POS_s
Stack Overflow	Manual vs. Stanford	97.63	89.21	83.42
	Manual vs. SyntaxNet	96.36	89.61	85.56
	Manual vs. spaCy	98.14	**92.29**	**89.89**
	Manual vs. NLTK	**99.60**	88.38	81.71
GitHub ReadMe	Manual vs. Stanford	97.54	84.70	78.14
	Manual vs. SyntaxNet	97.36	85.67	**82.61**
	Manual vs. spaCy	94.12	84.54	79.62
	Manual vs. NLTK	**98.20**	**86.55**	77.12
Java API Doc.	Manual vs. Stanford	96.14	84.32	77.89
	Manual vs. SyntaxNet	98.29	78.52	75.36
	Manual vs. spaCy	97.11	**90.79**	**78.42**
	Manual vs. NLTK	**99.48**	90.55	77.17

Table 3.4: Table of accuracy based on comparing the manual annotation with the output of the four libraries (Al Omran and Treude, 2017).

Taking the most relevant sources of the literature review of this study, we classify them by level of relevance from high to low to high, it is possible to identify the most relevant concepts of each publication, the central hypothesis formulated and the methods used to get the results. An overview of this classification can be seen in table 3.5. The main results are summarized to evaluate its relevance.

Table 3.5: Literature classified by relevance.

Year	Reference	Main hypothesis	Key concepts	Method(s)	Results	Relev
(2019)	Muter et al.	Patterns in linguistic analysis in a large collection of user stories.	Requirements engineering, user stories, backlog items, natural language processing, sprint tasks.	Stanford Part-of-Speech	7 elementary action verbs identified and a template for task labels.	High
(2018)	Martens	Ontological approach can ultimately achieve ubiquitous software traceability.	Ontological traceability for software Ontology, Ubiquitous Software Traceability, Software traceability, NLP	PDO traceability method, Ontology mapping	PDO Traceability Method appeared to be an effective method for the creation of trace links, using an ontology and a set of linguistic terms extracted from software artifact instances. As soon as the user stories or feature tests become slightly more complex, the UTA misses quite a few trace links.	High
(2015)	Lucassen et al.	Quality user story framework consisting of 14 quality criteria that user stories should strive to conform to.	User stories, User story quality, Language Processing (NLP) techniques	NLTK grammatical tagger.	A framework for higher quality user stories. A conceptual model of a user story. An initial set of relationships to indicate here the stories lack quality	High
(2015)	Hirschberg and Manning	Analysis of the actual status in natural language processing	Machine translation, speech recognition, spoke, dialogue systems, conversational agents, linguistic structure analysis, machine reading, text-to-speech	N/A	Big improvement in speech recognition. ML and deep learning will lead to further substantial progress in NLP. The tough problems of semantics, context, and knowledge will require discoveries in linguistics and inference.	Med
(2014)	Cleland-Huang et al.	A prior body of work to highlight the state-of-the-art in software traceability and to present compelling areas of research that need to be addressed.	Software traceability, Software engineering	Traceability information model (TIM)	Set of research directions in: Traceability Strategizing, Trace creation, Trace Maintenance, Trace Integrity and Visualizing Trace Data	Med
(2014)	Rong	Patterns in linguistic analysis in a large collection of user stories.	Requirements engineering, user stories, backlog items, natural language processing, sprint tasks.	Stanford Part-of-Speech	7 elementary action verbs identified and a template for task labels.	High

Year	Reference	Main hypothesis	Key concepts	Method(s)	Results	Relev
(2014)	Kalchbrenner et al.	Use of convolutional neural network cab be easily applicable to any language and does not rely in parse trees	DCNN, sentiment prediciton, quesiton classification, sentence models, *k*-Max pooling, n-Gram order, feature graph	Network machines and folding	A high perfomance network on question and sentiment classification	High
(2014)	Kim	CNN model for sentimen analysis and question classification	Pre-trained word vectors, multi-channel model, single-channel model, non-static representation, word2vec, CNN	Deep-learning	A CNN with one layer of convolution performs remarkably well	High
(2013)	Williams	An analysis of POS tag patterns in ontology identifiers and labels	Extraction of POS tags and general syntactic pattern analysis	Natural language processing and part-of-speech	Identifier names follow simple syntactic patterns; each type of identifier can be expressed through relatively few patterns; and the syntax of identifiers differs from natural English inconsistent ways.	High
(2013)	Mikolov et al.	Use of advanced techniques that improve accuracy in a word similarity task with lower computational costs	Similarity, high-quality word vectors, multiple degree similarity, recurrent neural net language model, bag-of-words and skyp-gram models	Neural network model	High maximization accuracy and quality vectors using simple models with simple model architectures	High
(2012)	Pustejovsky and Stubbs	Word2vec model can be derived to different problems.	Word2vec, bag-of-words model, Skip-Gram model, maximization, softmax, computational efficiency, neural networks, negative sampling	Formula derivation	The use of a simplified context definition in a multi-word context	High
(2014)	Cambria and White	Survey article of Overlapping curves on Syntactics, Semantics, and Pragmatics	NLP, Syntactics, Semantics, Pragmatics	Mentioned: Statistical, stochastic graph-based	Based on evolution of NLP according to three different paradigms, namely: the bag-of-words, bag-of-concepts, and bag-of-narratives models, NLP research is gradually shifting from lexical semantics to compositional semantics and offered insights on next-generation narrative-based NLP technology.	Med
(2011)	Liu et al.	Explore existing methods that can be beneficial to extract knowledge and develop biomedical ontologies.	Ontology, Ontology learning From text, Ontology Enrichment, Information extraction, Natural Language Processing	Symbolic, Statistical, Symbolic. Ontology learning systems: ASIUM, DODDLE II, HASTI, KnowltAll, MEDSYNDIKATE, OntoLearn, STRING-IE, Text-To-Onto Text20nto, TIMS, WEB- ¿ KB	Fully automated acquisition of ontology by machines is not likely in the near future. Symbolic methods suffer the limitation of coverage and applicability due to the requirement of manual acquisition and codification of lexical knowledge for each domain. Statistical methods, in general, cannot provide linguistic insight on their own, a human expert is required to make sense of the results.	Med
(2009)	Gabrilovich	Propose a novel method, called Explicit Semantic Analysis (ESA), for fine-grained semantic interpretation of unrestricted natural language texts.	high-dimensional space of concepts, semantics, domain-specific world knowledge, Lexical databases, Semantic Interpreter, Term frequency, Link Structure, high-dimensional space of concepts, Feature generation, Text categorisationcategorization, Text classifiers, Labeled feature vectors, Inter-article links, Bag-of-words	Latent Semantic Analysis (LSA). Text categorization, Singular Value Decomposition (SVD). Explicit Semantic Analysis. Inverted Index Pruning	The concept-based representation allows generalizations and refinements to address synonymy and polysemy partially. ESA results in significant improvements in automatically assessing semantic relatedness of words and texts.	High
(2008)	Maynard et al.	Describe a method for term recognition using linguistic and statistical techniques, making use of contextual information to bootstrap learning. And investigate how term recognition techniques can be useful for the wider task of information extraction, making use of similarity metrics and contextual information.	information extraction, ontology population, term recognition, Ontology population, Semantic Network, Information Extraction, tokeniser, sentence splitter, POS tagger, gazetteer, finite state transduction grammar, Orthomatcher, Measures and metrics, BDM metric	Boundary words, NC- Value method, GATE, ANNIE, Learning Accuracy metric	How NLP techniques can be adapted to the wider task of information extraction. While term recognition generally uses primarily statistical techniques, usually combined with basic linguistic information in the form of part-of-speech tags, information extraction is usually performed with either a rule-based approach or machine learning, or a combination of the two. Experiments with a new evaluation metric have shown auspicious results and clearly, demonstrate a better evaluation technique than the Precision and Recall metrics used for traditional (non-ontology-based) information extraction applications.	High

Year	Reference	Main hypothesis	Key concepts	Method(s)	Results	Relev
(2008)	Zhang et al.	A novel approach to re-establishing traceability links between existing source code and documentation to support software maintenance.	Traceability links, Traceability, Software Maintenance, Ontology, Text Mining, Deep links, Ontology Alignment	Not mentioned	The approach allows inferring implicit relations between discovered concept instances. The linked ontologies provide the capability to perform queries across the boundary between a programming language and natural language.	Med
(2006)	Bernstein and Kaufmann	Controlled natural languages offer to bridge the gap between the end-user and the logic-based scaffolding of the semantic web. Propose a tool that allows users to edit and query ontologies in a language akin to English: GINO	Formal logic, ontology editing tool, querying disconnection, Semantic Web, natural language interfaces NLI, multi-level grammar, grammar compiler, incremental parser, ontology-access layer, parse tree, static grammar rules, OWL	Not described	The evaluation with six end users provides some evidence that novice users are capable of virtually flawlessly add new elements to an ontology. Also, users were confused with major Semantic Web elements.	Med
(2003)	Novichkova et al.	Create a biomedical domain-oriented NLP engine called MedScan that efficiently processes sentences from MEDLINE abstracts and produces a set of regularized logical structures representing the meaning of each sentence. The engine utilises a specially developed context-free grammar and lexicon.	Lexicon, Lexeme, Semantic frame, Tokens, word descriptors, Semantic tree, Semantic nodes, Attribute slots	Active chart parser algorithm (Allen, 1994), semantic interpretation in LFG (Sells, 1985)	MedScan performance is satisfactory for the real-time MEDLINE processing with a coverage rate of 34%.	High
(2003)	Soricut and Marcu	Two probabilistic models to identify elementary discourse units and build discourse parse trees	Parse trees, discourse segmentation, lexical, syntactic trees, discourse parser	Discourse model	Sophisticated discourse parsing model to yield discourse trees at an accuracy level that matches near-human levels of performance.	High
(2003)	Chowdhury	Research activities in NLP.	natural language text processing systems, text summarization, information extraction, information retrieval, domain-specific applications	Statistical, Finite State	Experiments performed show promising results using NLP	Med
(2003)	Rindflesch and Fiszman.	Propose a methodology for interpreting linguistic structures that encode hypernymic propositions, in which a more specific concept is in a taxonomic relationship with a more general concept.	Semantic processing, Knowledge representation, Information extraction	SemSpec	Crucial information is provided by semantic groups from the Semantic Network and hierarchical relationships, but a lot of space for improvement form the error analysis.	Low
(2001)	Maedche and Staab	Ontology learning architecture	Ontology-learning process, concept extraction, Lexical entry, Ontology Engineering, Workbench, OIL (ontology interchange language), DAML-ONT (DAML ontology language)	Ontology-learning	A promising architecture for ontology learning that crosses borders of disciplines.	Med
(1999)	Berland and Charniak	A method for extracting parts of objects from wholes (e.g. ”speedometer” from ”car”)	Semantic, lexicon	Statistical Methods	Given a very large corpus the method finds part words with 55% accuracy for the top 50 words as ranked by the system.	Low
(1995)	Kuhn and De Mori	A new method to build natural language understanding	Speech understanding, semantic classification tree, natural language, decision tree, machine learning	Semantic rules learning	Semantic rules can be learned automatically from training data, yielding successful NLU for applications.	High
(1992)	Rolland and Proix	The need of support of RE with tool based linguistic approach	Linguistic approach, R.E. support, natural language analysis, conceptual schema	Case of case linguistic approach REMORA methodology	Generalization of the linguistic approach using CASE tool	High

Chapter 4

Linguistic Analysis

4.1 Annotation techniques

The way that humans expresses their thoughts through a language can be rather complex to understand as a whole. Therefore, it is important to divide text written in natural language in chunks that can be named in tokens that can be classified as a part-of-speech, then hierarchized in part of a discourse and subsequently can do sentiment analysis in a narrative classification. This chapter describes a model to extract relevant information in the context of the RE4SA. The main purpose of the annotation is to generate a collection of machine-readable texts (subsequently called corpus) that have been produced in a natural language environment.

4.1.1 Layers of Linguistic Description

To retrieve information inherent in the text, NLP divides annotation into different layers and differentiates different levels of granularity. The lowest level refers to the syntactics at a word level or named as a bag-of-words. One level higher the semantics layer or bag-of-concepts and the pragmatics or bag-of-narratives as the top layer as seen in figure 4.1. The present work is an analysis of the syntactic and semantic level. We consider the results as a possible solution to a pragmatic problem, analysing the results of the artifact created and see how behaves taking into account the context.

Different techniques can be used in each layer with the finality to extract information. One theory of this work is that the combination of at least two techniques can provide significant results in the task of finding patterns in software artifacts written in natural text. Between these techniques, it is possible to find the widely accepted tokenization process, the part-of-speech tagging (POs tag), the dependency parsing, entity

recognition between others.

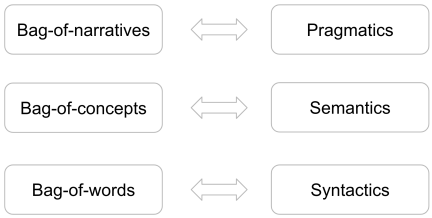

Figure 4.1: Layers of linguistic description

Syntactic annotation analysis

The main purpose of the syntactic level is to split the text into individual words defined as tokens. This demarcation of text is used to classify them in different classes where each class can represent more than one lexeme. In linguistics, a lexeme is a unit of lexical meaning in a given language. For instance, in English the words *read*, *reads*, *reading* belongs to the same lexeme in different forms. One of these forms is used in dictionaries (as lemmas or citation form). Each token can receive a name that is categorised defined by the syntactic language rules and can be defined as a member of a group called part-of-speech. However, at this level, each token has no meaning in the real world as the semantics of the language provides this meaning and, therefore, it will be used in the semantic layer. In our example, the word *read* can be tagged as verb and *book* as a noun.

POS tagging makes use of a catalogue and categories of speech classes. The most commonly identified are verb, noun, adverbs, preposition, pronoun or conjunction. A tag identifies each class — for instance, VB, NN, RB, IN, PRP, CC, respectively. For instance, the statement *I want to describe myself on my page* can be split into nine tokens. Each token classified in a POS tag can generate the following sequence of tags *PRP VBP TO VB PRP IN PRP JJ NN*. A set of these sequences produce a corpus of a full-text artefact. Taking the definition of *copus* as:

> "A collection of machine-readable texts produced in a natural communicative setting" (Pustejovsky & Stubbs, 2012). A corpus is said to be "representative of a language variety." (Leech, 2014).

Each POS tag is mapped in a general way of a part-of-speech class, as seen in figure 4.2, giving a POS tag the lowest level in the speech, It is possible to identify when, for example, a pronoun is personal, possessive determiner or possessive pronoun.

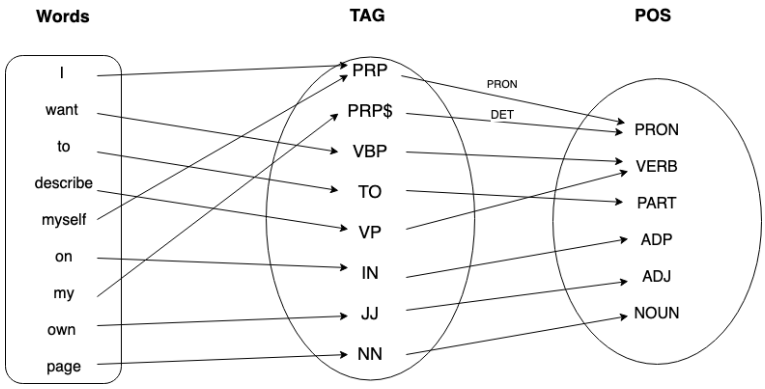

Figure 4.2: Example of part-of-speech tagging

However, all are classified as a pronoun. The same occurs with nouns and compound nouns and plural nouns.

It is essential to classify the tokes in the correct POS tags to provide more accurate results and create more precise applications using these techniques. For instance, question-answering systems, machine translation systems or in this case entity recognition. Therefore, it is essential to select the most convenient tool to perform such a task to reduce the number of inaccurate tags that can be a result of ambiguous terms. As described before, spaCy has been tested with good overall performance compared with other state-of-the-art linguistic tools (Pustejovsky & Stubbs, 2012)

It is possible to define a syntactic parse tree and analyse the structure of a statement. From a bottom-up perspective, Each token word of a sentence constitutes the leaves of the tree. Each leave has one and only one POS tag as a parent node. One or more POS tags can belong to a part-of-speech word class. On or more part-of-speech classes is subordinated by a phrase. Each phrase can be classified as a noun phrase or a verb phrase. These phrases belong to a sentence node as can be seen in the definition of the parse tree in figure 4.3.

As an example for a manual annotation we can extract from a product backlog a user story from a portal that provides Agile training "*As a site member, I want to describe myself on my page in a semi-structured way so that others can learn about me*". To perform an annotation, the sentence needs to be divided into words and assign a POS tag class to each word or token. Once all tokens are tagged and based on our definition or syntactic parse tree, it is possible to generate a parse tree as shown in figure 4.4.

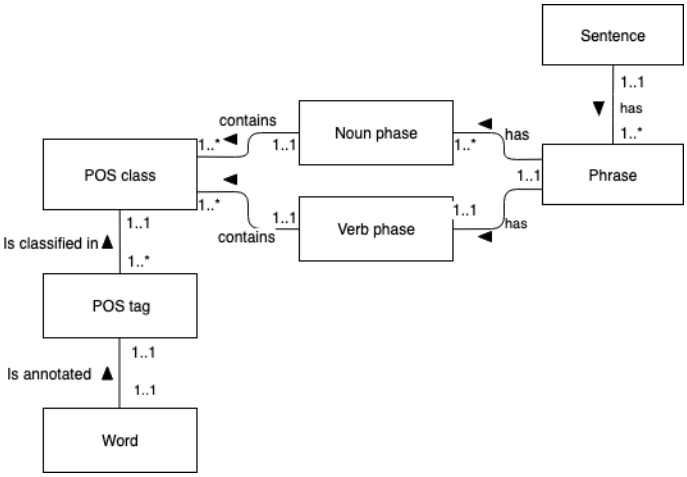

Figure 4.3: Syntactic parse tree meta model

Syntactic pattern recognition

One of the hypothesis is that there are patterns in software artefacts written in natural language that exist innocuously. However, we do not think explicitly in that. We use our empiricism together with syntactic language rules and exceptions to write sentences or, in some cases, a semi-structured approach like in the case of user stories. However, how to find these patterns and in what percentage these patterns cover the overall corpus.

To represent such patterns, a specific notation is used. The symbols used to describe such notation can be seen in table 4.2.

Pattern identification algorithm

POS tagging with spaCy tagger libraries are then performed. The POs tags of each sentence were written line by line in a new file. Starting with the POS tags followed by the number of coincidences separated by a character identifier (i.e. ;). The words that are represented by such tags, e.g. ”PRP VBP PRP$ NN TO VBI; 10; want my rating to show”.

A POS tag classification are performed in order to identify patterns. A POS class is defined as:

> a sequence of POS tags in annotated text that contains at least one or more than one POS tag.

A POS tag classification program was developed to make an in-depth syntax analysis of POS tag patterns and automate the process. First, the first POS tag of the first sentence is selected and added to a list of POS classes identified. However, To avoid POS tags that may not be relevant, the following criteria are considered:

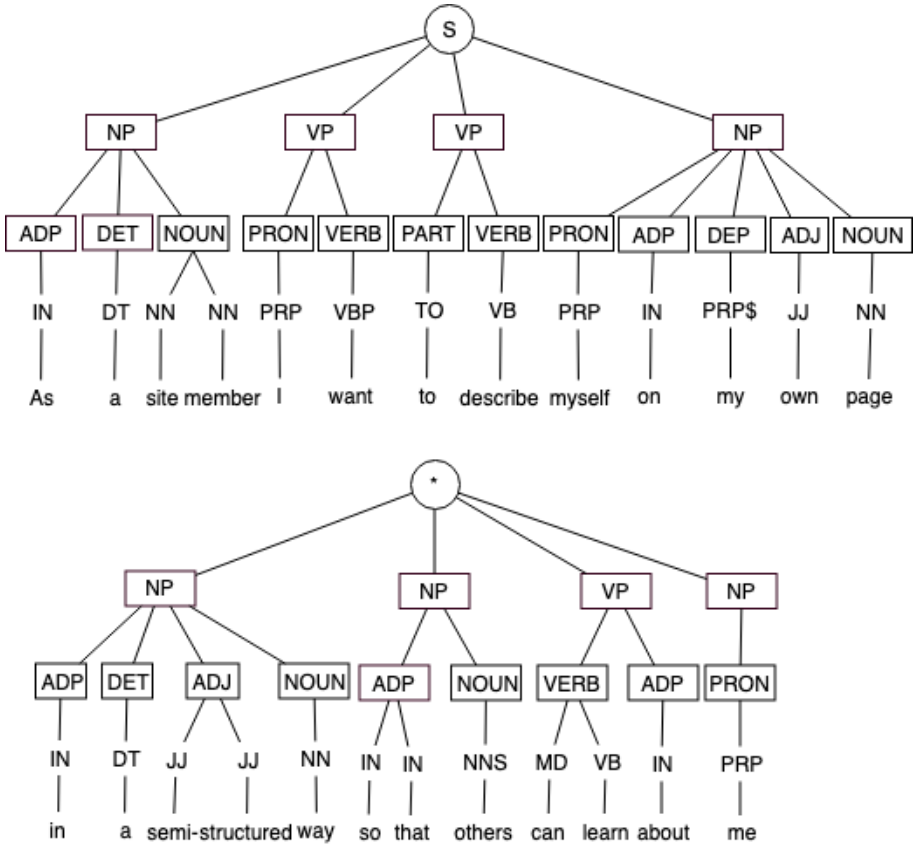

Figure 4.4: Syntactic parse tree model

1. The POS class contains at least one type of noun or verb tag in any form.

2. The POS class does not include a space tag.

3. The POS class does not include punctuation tags.

Then, the algorithm starts looking for such POS class over all the corpus and count the number coincidences of each class found. First, a POS class is formed, taking the first element *i* for the row. Then, the POS class is checked if it was previously selected and processed. If the POS class has not to be chosen yet it is compared to the same number for tag elements throughout the corpora. Once this is completed, the next class is selected. The next class is formed taking current POS tag and the following consecutive tag in a row for each annotated sentence. The process continues for each sentence taking 10 POS tags as a maximum limit for a POS class and considering in between POS tags as classes as well. Once this limit is reached, then the algorithm starts forming classes from the position $i + 1$ of each row in the corpus, starting from *i* as the first POS tag of the sentence to the *n* number of tags in a row. As a result, an accurate amount of

English POS					
TAG	POS	Description	TAG	POS	Description
,	PUNCT	punctuation mark, comma	RB	ADV	adverb
.	PUNCT	punctuation mark, sentence closer	RBR	ADV	adverb, comparative
JJ	ADJ	adjective	RBS	ADV	adverb, superlative
JJR	ADJ	adjective, comparative	VB	VERB	verb, base form
JJS	ADJ	adjective, superlative	VBD	VERB	verb, past tense
MD	VERB	verb, modal auxiliary	VBG	VERB	verb, gerund or present participle
NN	NOUN	noun, singular or mass	VBN	VERB	verb, past participle
NNP	PROPN	noun, proper singular	VBP	VERB	verb, non-3rd person singular present
NNPS	PROPN	noun, proper plural	VBZ	VERB	verb, 3rd person singular present
NNS	NOUN	noun, plural	WDT	ADJ	wh-determiner
PDT	ADJ	predeterminer	WP	NOUN	wh-pronoun, personal
POS	PART	possessive ending	WP$	ADJ	wh-pronoun, possessive
PRP	PRON	pronoun, personal	WRB	ADV	wh-adverb
PRP$	ADJ	pronoun, possessive			

Table 4.1: English part-of-speech with spaCy tagset.

Symbol	Description	Example
( )	element delimiter	
*	zero or more occurrences	(NN)*
+	one or more occurrences	(NN)+
|	alternative	IN |ON
?	previous character is optional	NNS?

Table 4.2: POS tag pattern notation

coincidences of each class can be obtained and used as an input for the identification of patterns.

It is necessary to order the classes by the number of coincidences to find the most representative classes of the whole corpus.

Once the most representative classes are identified, they are simplified in an expression using the notation described in table 4.2.

Semantic annotation analysis

The syntactic analysis gives an idea from the lowest level of the linguistic analysis and the patterns that people use to describe requirements in English language. However, these patters are not related to a meaning but only yo a phrase that can be a noun phrase or a verb phrase. Therefore, it is necessary to analyze the corpus from a semantic level.

It is necessary to identify the discourse of a sentence. To have a better understanding of what is a discourse, this can be defined from its elements and represented as a tree. We take the definition of discourse as:

> "A discourse structure is a tree whose leaves correspond to elementary discourse units (edu)s, and whose internal nodes correspond to contiguous text spans (called discourse spans) (Erdmann, Maedche, Schnurr, & Staab, 2000)".

Taking the previous example in figure 4.9, it is necessary to create the discourse structure of the sentence. Erdmann et al. (2000) characterizes each node by a rhetorical relation such an *(*attribute) or an *(*enabler). Within the rhetorical relation is also labeled as *(*satellite) or *(*nucleus). An arrow links the *(*satellite) to the *(*nucleus) and is labelled by the rhetorical relation. In figure 4.5 it is possible to see this example in a discourse tree. The horizontal lines correspond to text spans, vertical lines to *(*nucleus) and the arrows labelled with corresponding rhetorical relation.

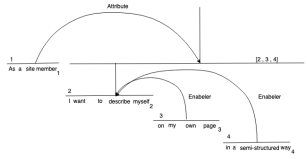

Figure 4.5: Discourse structure of a sentence

This disclose model is still in a sentence-level that is very abstract and generic. Therefore, it has to be segmented into smaller parts to divide the problem. For this, a discourse segmentation and discourse parsing is used.

Discourse segmentation is the segmentation of text in non-overlapping segments. Combining the lexycalized syntactic parse model, the parse tree model and the discourse structure in such a way that it is possible to identify the the links between segments. As in our example, in figure 4.6, it is possible to identify three segments each segment with two sub-segments as can be ween in figure 4.6. The first segment from left to right an

attribute relationship is created between a satellite and the nucleus, in this case, the nucleus is a subordinated clause SBAR (Clause introduced by subordinating conjunction). The other bub-segments are linked to the nucleus as enablers.

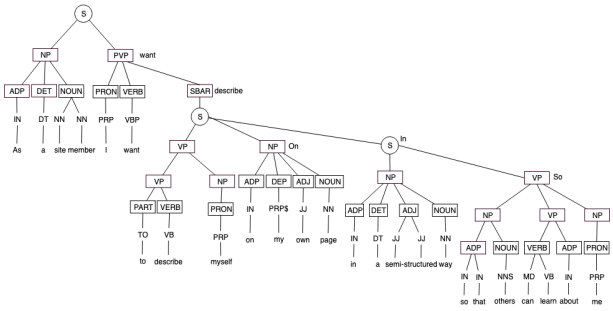

Figure 4.6: Discourse segmentation using lexicalized syntactic trees.

The semantic annotation is necessary at this level. To identify the information to the concepts and how these concepts are related between them. NLP provides different techniques to perform this annotation like dependency parsing or lemmatization. For this study, a dependency technique is used to identify patterns to extract desired information. Dependencies can be identified from the tree, for instance, the words *(*want, describe on, in) and *(*so) generate a dependency over a segment and a dependency within the same phrase that belongs. The word *(*want) has a dependency on a nominal subject (nsub) form the pronoun *(*I). *(*Myself) has a dependency of a direct object (dobj) form the verb *(*describe), *(*My own page) has a object of a preposition (pobj) form *(*on).

As seen in the discourse tree, a discourse structure of a sentence can act as a unit of a group of words (noun phrase, verb phrase). Those words are related to a grammar function concerning the other words of the sentence (e.g. subject, direct object, indirect object). To help in the task of discourse parsing a dependency parse tree is used to model the discourse of a sentence. Dependency trees are represented by directed arch that connect a word head with a dependent. Each arch has a head with a dependent with a semantic dependency relation and these arches from a rooted tree as seen in figure 4.10.

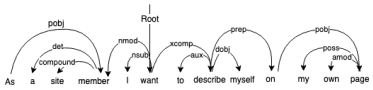

Figure 4.7: Dependency tree.

The use of parse trees helps to facilitate the reading of the semantic dependencies. And at the same time, it is possible to detect patterns in dependency trees. However, it

gets complicated if there are patterns that do not form any pattern and only generates noise in the analysis. Based on the discourse segmentation tree, it is possible to see that phrases that are noun phrases are more relevant to take into the analysis and splitting in noun chunks.

The dependency pattern analysis is carried out taking the training data sets used in the syntactic analysis and reading only noun chunks form the parse tree. The dependency annotation is performed to the three different data sets using a developed artefact.

Pragmatics analysis

The main goal of this study is at the level of pragmatics. The proof-of-concept indents to facilitate this analysis by answering the main research question.

Syntactic and semantic analysis are evaluated in a context to try to solve a real-life problem; in this case, how to trace requirements through trace links between software artifacts.

Chapter 7 describes the results of the proof-of-concept are described, and the analysis and discussion of them in chapter 8.

4.1.2 Case analysis

To identify such patterns and facilitate the analysis, two artefacts are developed to automate and aid in such task performed in three different cases.

The artefact used for this analysis is a data set of user stories. Each user story is included as a sentence.

The first case contains a collection of 97 user stories. These user stories are requirements from a system to manage training and sponsors in a website.

The second case includes a data collection of 55 user stories form a poker game system.

The third case is a sample data set with 210 user stories from a system to facilitate storage and sharing of research information.

4.1.3 Preliminary results

POS-tag analysis

Taking the classes ordered from the higher number of coincidences and looking for the most representative, We obtained a distribution positively skewed with a long tail with

those patters with less representative in the corpus as shown in figure 4.9, where a) is the distribution count of POS classes of a data set of 97 user stories, b) 55 and c).

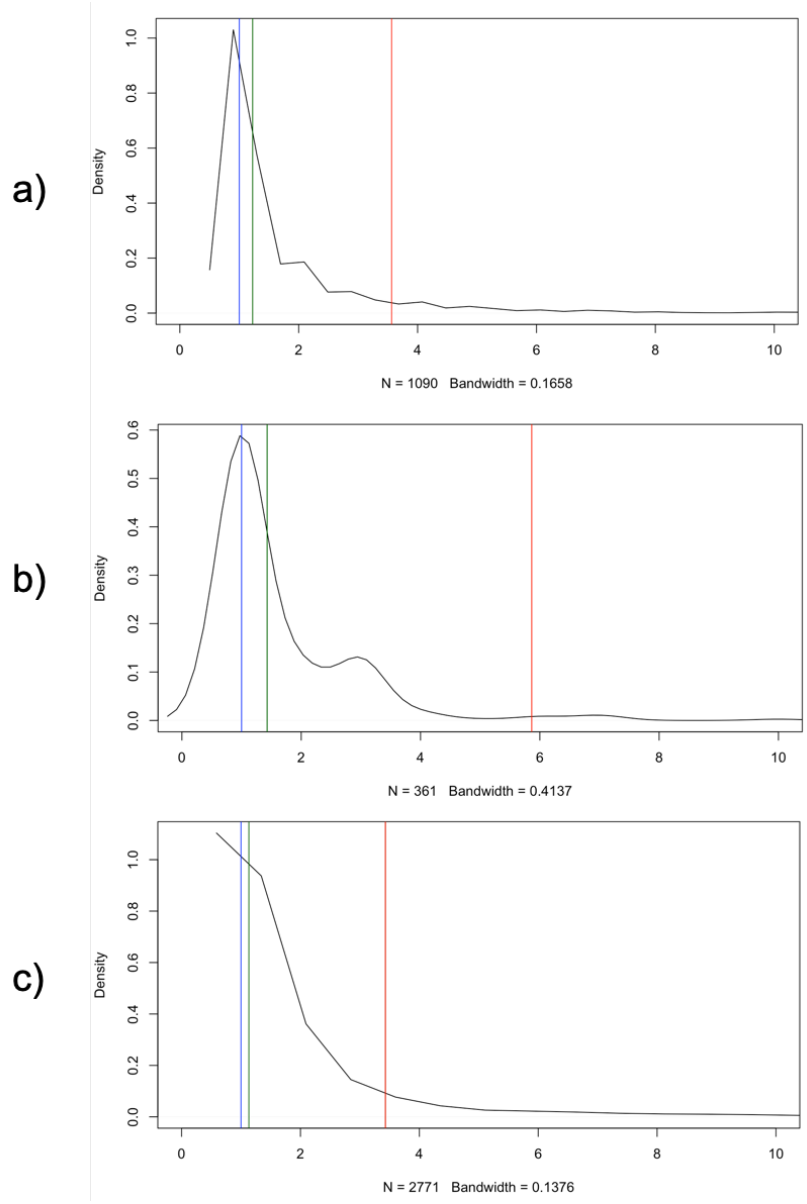

Figure 4.8: POS tag classes distribution, where *x* axis represents all classes ordered by number of coincidences and the *y* axis the density.

From the first data set, a total of 1,090 different classes were identified with a total of 3,884 coincidences. When ordered by frequency from the class with most number of coincidence to the class with the least, the top 132 contained 2,612 of the coincidences. This give us a 67.25% concentration of the whole corpus and the rest of the classes counted less than 7.

From the second data set, a total of 361 different classes were identified with a

total of 2,117 coincidences. Ordered by the frequency from the class with most number of coincidence to the class with the least, the top 51 contained 1,666 of the coincidences. This give us a 78.69% concentration of the whole corpus.

From the third data set, a total of 2,771 different classes were identified with a total of 9,503 coincidences. Ordered by the frequency from the class with most number of coincidence to the class with the least, the top 313 contained 6,662 of the coincidences. This give us a 70.10% concentration of the whole corpus. Each class in the top 40 had more than 30 occurrences in the whole corpus.

It is possible to look for specific patterns in selected POS classes taking the observed annotation. The table 4.3 show the identified patters and their frequency.

Pattern	Frequency
IN* DT NN+ PRP*	1.353
PRP* VBP TO VB*	758
(NN+ |NNP) PRP	392
IN DT NN+	380
TO VB DT*	299
JJ NN |NNS	75
DT NNP	47
NN CC	23

Table 4.3: Table of pattern frequency

From the patterns determined above, it is possible to identify specific characteristics of the language. The pattern 2, for instance, contains the POS class ”PRP VBP”, where PRP is a personal pronoun like ”I” and VBP is a verb in a non-third person singular present, for instance, ”want” or ”like”. This is more like a desire or wish of the personal pronoun. In the context of a requirement is only the confirmation the something is required or not. Therefore, it can be discarded as an action verb.

Another observation is, for instance, in patterns one and four starts with the POS tag IN, that is a preposition or subordinating conjunction. This is very logical as every user story has the preposition ”As” and it can be considered as part of the user story template. As a result, it is discarded as part of the patterns.

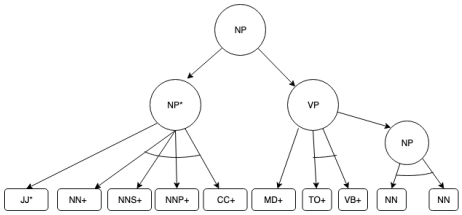

Figure 4.9: Pattern parse tree model

There is also a common relation between patterns. For instance, there is a possibility that pattern 1 and 2 can be connected through the PRP and patterns 2 and 3 through the POS class "TO BV". If we take all these common classes and leaving out POS tags that are not part of a template, personal pronouns and verbs that are not in an infinitive form, the patterns can be simplified as follows:

1. JJ* (NN+ — NNS+ — NNP+ — CC+)*
2. (MD+ (TO+ — VB*)*) VB* ((NN)+ — (NNS)+)*

The representation of these patterns represented in figure 4.9, where the crossed curve among several arrows indicates a logical operator OR. They represent 80% of the corpus. These patterns answer the research questions RQ_1 and RQ_2 partially. The model needs to be completed going up one level with a semantic analysis in the following section.

Dependency analysis

From the first case, a total of 864 dependencies are found. 84% of the dependencies are concentrated in three main groups, nominal subjects (nsub), objects of possession (pobj) and direct object (dobj). Within the group nsub it is possible to observe that in 94.6 % the header of the dependency is a VERB with three main POS tags, VBP, VB and VBZ as can be seen in table 4.4.

For the second case, the same tendency can be observed. With a total of 390 dependencies 93.33% of the dependencies are nominal subjects, objects of possession and direct object. Within the group nsub in 88.03 % the header of the dependency is a VERB with three main POS tags, VBP, VB and VBZ as can be seen in the table 4.5.

For the third case, with a total of 1188 dependencies, 93.01% of the dependencies are nominal subjects, objects of possession and direct object. Within the group nsub in 98.03 % the header of the dependency is a VERB with three main POS tags, VBP, VB, BVN and VBZ as can be seen in table 4.6.

POS Head-tag	Dependency	Sum	Abs. prop.	Rel. prop.
	nsub	250	28.94%	
ADP-IN	pobj	290	33.56%	
	dobj	187	21.64%	
VERB	nsub	259	29.98%	
VERB-VBP	nsub	74	8.56%	28.57%
VERB-VB	nsub	137	15.86%	52.90%
VERB-VBZ	nsub	34	3.94%	13.13%
FOR	dobj	32	3.70%	17.11%
OF	dobj	40	4.63%	21.39%
ON	dobj	30	3.47%	16.04%
TO	dobj	35	4.05%	18.72%

Table 4.4: Dependency tree analysis

POS Head-tag	Dependency	Sum	Abs. prop,	Rel. prop.
	nsub	142	36.41%	
ADP-IN	pobj	113	28.97%	
	dobj	109	27.95%	
VERB	nsub	142	36.41%	
VERB-VBP	nsub	71	18.21%	50.00%
VERB-VB	nsub	47	12.05%	33.10%
VERB-VBZ	nsub	7	1.79%	4.93%
FOR	dobj	0	0.00%	0.00%
OF	dobj	0	0.00%	0.00%
ON	dobj	0	0.00%	0.00%
TO	dobj	0	0.00%	0.00%

Table 4.5: Dependency tree analysis

POS Head-tag	Dependency	Sum	Abs. prop.	Rel. prop.
	nsub	357	30.05%	
ADP-IN	pobj	452	38.05%	
	dobj	297	25.00%	
VERB	nsub	354	29.80%	
VERB-VBP	nsub	212	18.85%	59.89%
VERB-VB	nsub	117	9.85%	33.05%
VERB-VBN	nsub	10	0.84%	2.82%
VERB-VBZ	nsub	9	0.76%	2.54%

Table 4.6: Dependency tree analysis

4.2 Similarity and ambiguity

Software requirements are mostly collected in artifacts written in natural text. The new methodologies and methods like Agile and scrum have contributed to a better organization and standardisation of them. However, there are still areas that can be improved as

humans tend to be unconsciously ambiguous.

One solution to this problem is trying to create more strict rules when generating requirements like the use of templates or tools that help to reduce the ambiguity of terms. However, this can become very difficult to handle due to the fact that the new methodologies require continuous iterations. Each iteration in small cycles with different groups or teams integrated by people interacting in the same context.

People think differently and may have a different vocabulary or express itself in a certain way due to other external factors like culture or different mother language. Assuming that somehow ambiguity exists and will be challenging to get rid of it. Another solution proposed to solve this problem is through the use of probabilistic models that can resolve in an efficient way such ambiguity introduced.

The bag-of-words models seem that provide a right solution in the representation of a vocabulary. This model uses a repository of vectors in a matrix that represents the vocabulary.

To have a repository of word vectors by itself, however, is a starting point in the classification or prediction of the context. The context of a word is defined by its neighbors in a sentence. If a word shares quite often the same neighbors we can say that there is a context for this word. If we can establish the what is the probability of having a context given a word or having a word given a context.

The *Word2vec* models seem that provides an effective solution to this problem. These models use a repository of vectors in a matrix that represents the vocabulary and a repository matrix of context vectors that contain all the vectors that surround a word in question.

One of the sub-research questions (RQ_4) states that how a sub-ontology can be identified in a software development artifact. This can be performed comparing each term of the sub-ontology by each term in the PO. However, it was assumed that ambiguity is there and at the moment we have to live with it.

If two concepts are syntactically equal, we can say that we are talking about the same concept. But, what happens when two concepts are very closely related?. If one person introduced the ambiguity, then it is easy to solve the problem. When two or more people introduce an ambiguous concept, then the solution is more complicated as maybe a discussion is necessary to agree in a resolution.

When new requirements are intended to be implemented, there is a high possibility that may contain terms that may be similar and the same and similar but not the same. This study proposes to create a model using *Word2vec* architecture to solve this problem.

The model is integrated by the two dimensions used by *Word2vec* vocabulary and

context, plus another class dimension called domain context. This third dimension will be in charge to resolve an ambiguity once an ambiguity was detected in the first two dimensions.

The model then will consist of three layers that process word vectors, context words and domain context words as classifiers, as shown in figure 4.10. As a result, we would be able to resolve if a concept is similar or not based on a general context and a domain context.

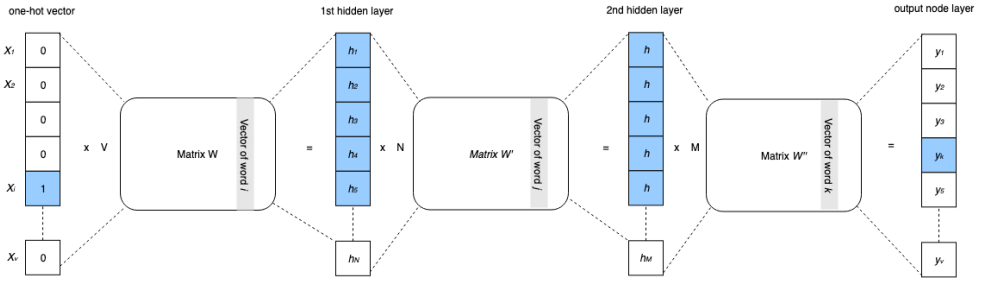

Figure 4.10: Ontological ambiguity resolve model.

Taking the model used by Mikolov et al. (2013) we have our general knowledge context vectors (*W*) and we add a second class that is our domain context vectors (*W"*), we have the following *softmax* formula as:

$$u_j = hW'(._j))W''(._k)) \tag{4.1}$$

$$P(w_{o,j}|w_I) = y_j = \frac{exp(u_j)}{\sum_{j'=1}^{V} exp(u_{ji}))} \tag{4.2}$$

A model trained with GloVe vectors in Common Crawl and OntoNotes with 685k vectors is used as general vocabulary and general context. A custom model trained with Gensim with vectors generated in the domain of the system is used as a domain context using Gensim's neural network in python libraries to generate the model.

The domain context is determined by the domain ontology taking as a base the total set of user stories that is used to generate the product ontology.

Chapter 5

Ontology generation

This chapter intends describe the pragmatic problem and make a formalization of the analysis performed during the development of this research using different artefacts and create the basis for the ontology generation.

Based on such formalization, it is intended to generate a sub-ontology of different artefacts to find trace matches to finally establish trace links between various software development artefacts written in natural language.

5.1 Trace links and trace match

It is a usual software development practice to start making corrections or adding new functionality directly in the code without a systematic method, knowledge of the original authors of the code or simply the loss of track of versions.

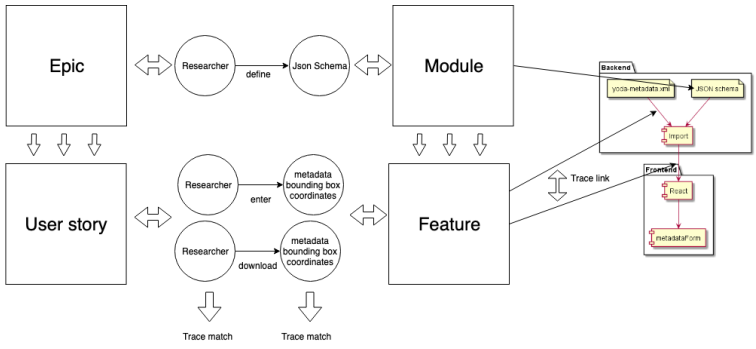

Figure 5.1: Example of trace match and trace link in RE4SA.

An example is taking one of the cases of the analysis, having one feature from the realised architecture and manually generating the ontology and another sub-ontology

from a user story. It is possible to detect a match in the ontology and as a consequence, an existing link between different artefacts.

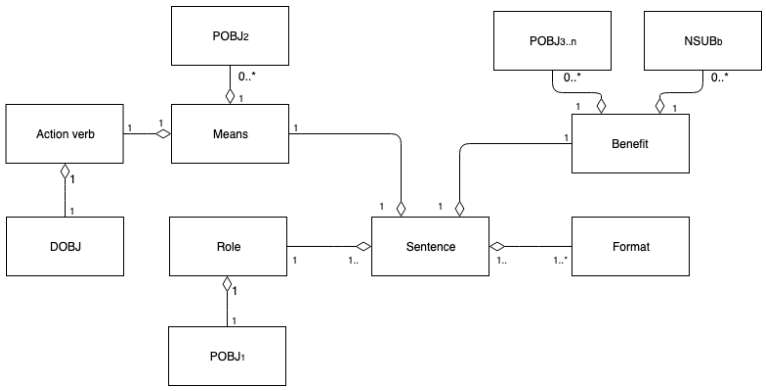

Figure 5.2: Dependency conceptual model of a user story

This trace link can indicate that there is an event that affects such features that are interrelated by such trace link as can be seen in figure 5.1. An event can be derived by a change, correction or creation of new feature.

The conceptual model of the dependencies can be seen in figure 5.2. The role is derived from one and only one object of preposition dependency — the action verb from the nominal subject. The means from several or no direct object dependencies and the benefit can have a combination of direct objects and nominal subjects that basically contain non-action verbs that describe more benefits of the action verb, e.g. ”so others can learn about me”.

The format of how user stories facilitate the application of such statements in a well-formed user story that consists of only one role, one action and one direct object with the option to have several adjectives or indirect objects as defined by van de Weerd and Brinkkemper (2009):

> “A user story μ is a 4-tuple μ = { r, m, E, } where r is the role, m is the means, E = e1, e2,... is a set of ends, and f is the format. A means m is a 5-tuple m = { s, av, do, io, adj } where s is a subject, av is an action verb, do is a direct object, io is an indirect object, and adj is an adjective.”

5.2 Formalization

To been able to compare concepts of a sub-ontology and been able to trace links between artefacts, first, it is necessary to generate such sub-ontology. Each sub-ontology can be generated by processing the ontological information extracted from the corpus

and identifying a relationship between the ontological information using the different patterns identified in the analysis.

The combination of such patterns from syntactic to semantic layers can provide enough information to map them into concepts and relations in an ontology.

From the semantic tree, each POS category is part of a phrase. Such phrase can be a noun phrase or verb phrase that are part of a segment in a statement. Therefore, we define a phrase as:

Definition 1. *Phrase*

Let be w *a word,* p *a phrase.*

$w \in p$ *A word is an element of a phrase*

np (p *) A phrase* p *is a noun phrase*

vp (p *) A phrase* p *is a verb phrase*

$$\forall\, w\, \exists\, p\, (\, np\, (\, p\,) \vee vp\, (\, p\,)\,) \rightarrow (\, (\, w \in p\,)\,)$$

Each word is identified by a POS tag and each POS tag is part of a part-of-speech category. Then, a word is assigned according to its semantic function.

Definition 2. *POS and POS tag*

Let be w*, a word,* t *a POS tag and* c *a category,*

pos(w *,* t*) A word* w *has a POS tag* t

category(t *,* c *) A POS tag has a catogory* c

$$\forall\, w\, \exists\, p_t\, (\, pos(w,\, t) \wedge category(t,\, c)\,)$$

In the same way, there exists a class that contains a group of POS tags that can be part of a pattern that holds similarities with other POS classes. Therefore we can define:

Definition 3. *Frequent POS class*

Let be P_1*,* P_2 *pattern 1 and pattern 2,* t *a POS tag,* c *a POS category and* A_c *an annotated corpus,*

$t \in P$ *A POS tag is an element of a pattern*

$$\forall\, p_t \,\exists\, t \in A_c$$

$$\exists\, t \,(\, (\, tc \in P_1 \,) \vee (\, t \in P_2 \,) \rightarrow (\, t \in A_c \,) \,)$$

From a dependency tree, there exists a dependency in a phrase that is defined as a head and dependent.

Definition 4. *Dependency*

Let be np a noun phrase, h a head and dp a dependent.

$$\forall\, np \,\exists\, dependency \,(\, h \,,\, dp \,)$$

Example: "As a site member"

dependency (As, a site member)

pobj (As, a site member)

As stated before, a dependency has a head, dependency and dependent elements. Therefore, we can define the following corollaries:

When a dependency is a pobj (object of preposition) and the segment is before an action verb then the dependent segment is a source concept.

Corollary 1. Let be p_i a noun phrase, p_j a verb phrase, av_p is an action verb phrase, h a head and dp a dependent.

$$\exists\, p_i \in (\, p_1, p_2, \ldots\, p_n \,) \text{ where } (\, \mathrm{i} < \mathrm{j} \,) \wedge (\, p_j = av_p \,)$$

$$\text{dependency}(\, p_i \,) = \text{dobj}(\, h \,,\, dp \,)$$

For all dependencies dobj (direct object) and where the head POS category is a *VERB* and POS tag a *VB* then we can say that the head is an action verb and the dependent is a target concept.

Corollary 2. Let be w a word, $dobj$ a dependency, h a head and dp a dependent.

cat(w, $verb$) A word has a category VERB

tag(w, vb) A word has a tag VB

$\forall\, dobj \,\exists\, w$ *dependency* (h , dp) where $h \in$ (*cat(w, $verb$)* $\wedge$ *tag(w, vb)*)

$$\Rightarrow h \equiv \text{action verb} \wedge dp \equiv \text{target concept}$$

When the dependency is a nsub (nominal subject) and the head POS category is a *VERB* and POS tag a *VBP* then the head is a non-action verb. There can be only one action verb in a sentence.

Corollary 3. Let be w a word, $dobj$ a dependency, h a head and dp a dependent.

cat(w, $verb$) A word has a category VERB

tag(w, vbp) A word has a tag VBP

$\forall\ dobj\ \exists\ dependency$ (h , dp) where $h \in$ (*cat(w, $verb$)* $\wedge$ *tag(w, vbp)*)
$\Rightarrow \exists!$a(a = h) $\rightarrow$ a $\neq h$

When the dependency is a pobj (object of preposition), the head is a POS(ADP) the tag(IN) and the segment is after the action verb then the dependent is a target concept or a further description of the concept or properties.

Corollary 4. Let be $pobj$ a dependency, h a head, tc a target concept, np_i a noun phrase, av_p is an action verb phrase, and dp a dependent.

cat(w, adp) A word has a category ADP

tag(w, in) A word has a tag IN

$\forall\ pobj\ \exists\ dependency$ (h , dp) where $h \in$ (*cat(w, adp)* $\wedge$ *tag(w, in)*) $\rightarrow$
$dp = tc \Rightarrow \exists\ p_i \in$ (p_1, p_2, ... p_n) where (i $>$ j) $\wedge$ ($p_i \neq av_p$) $\wedge$ $dp \in p_i$

An ontology is integrated by concepts and relations between those concepts. The representation of the relation between two concepts of an ontology can be defined as:

Definition 5. *Ontology relation*

Let be c a concept.

$\exists\ C$ *: Set(Concept).(*$\forall c\ \exists$*.action(*c_s *,*c_t*)* $\wedge$ $c_S \in C$*) where* c_s *is a source concept and* c_t *is the target concept.*

To represent such statement in logical predicates containing ontological information, we first define a source and target concepts. Two concepts are connected and related between them with the action verb as can be seen in figure 5.3. The source and the target concept can be the same concept.

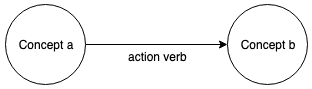

Figure 5.3: Representation of ontological axioms.

Taking the web ontology language we can define that:

Definition 6. *Web Ontology OWL A web ontology is a set of axioms* $\forall a \in D$ $\sigma = \{ a_1, a_2, a_3, ... a_n \}$ *where* a *is an axiom and D is the domain.* $\exists a \in \{ c, o, d, k, d_t, a_s \}$ *where* c *is a class,* o *is a object property,* d *is a data property,* k *is a key,* d_t *is a data type definition and* a_s *is an assertion.*

The representation of the axioms in figure 5.3 in the same definition but with the restriction that the concepts are different and from the definition, it can be the same concept as well.

Different artefacts have different ontologies but share similar concepts that are basically the same concept. The same occurs with sub-sets of the same artefact. For instance, user stories that have not been implemented yet and made modifications to the current implemented functionality generate a sub-ontology that already exist in the PDO. We define these concepts as:

Definition 7. *Concept*

Let be c *a concept,* A *an artefact,* o *an ontology,* O_s *a sub ontology,* O_p *a product ontology and* O_d *a product ontology domain.*

$$\exists\, c \in O_s \subset O_p \subset O_d \text{ where } O_s \subseteq A$$

At the same time, there are concepts that belong to the artefact sub-ontology but are still not present in the PDO. We can consider this functionality and features that will be integrated into the system. These new concepts are defined as:

Definition 8. *Sub-ontology*

Let be c *a concept,* A *an artefact,*o *an ontology,* O_s *a sub ontology and* O_p *a product ontology and* O_d *a product ontology domain.*

$$\exists\, o \in O_s \not\subset O_p \subset O_d \text{ where } O_s \subseteq A$$

In order to identify such sub-ontologies, it is necessary to create the ontology of the artefact or the sub-set of the artefact to facilitate the process of comparison of objects. Having created such ontology, it is necessary to compare the concept with each concept of the product ontology one by one in order to identify similarities and establish that an existing ontology has been found. When two concepts have equal names or similar names and share the same properties as data types, we can say that they are the same concept.

In the context of the RE4SA, the relation of two concepts can represent a feature or set of features when processing user stories as an artefact and modules or sub-modules when processing Epic stories. However, there are concepts that do not have a direct relationship with a feature or a module, such as the case of quality requirements, for instance, a requirement to reduce the time of a feature to process or the amount of storage required during the execution. This type of requirements needs to be somehow described in the concept to know what other relationships and concepts as by itself remains abstract and difficult to automate.

In software development, there is a practice to keep a linguistic relationship with the name of the feature and the coded functions or objects and methods in object-oriented programming(OOP) this is in order to facilitate reading the code in a logical way. This linguistic relation can be seen, for instance, in the case three epic stories contain the name of the module to which they refer the set of user stories in the form of a noun. As a result, it is possible to establish a trace link between the concept and the module either in the functional model or in the code.

In the same way, a feature or group of features can be linguistically related to a concept in the ontology. Therefore, a trace match can be established.

Definition 9. *Let be f a feature, F a group of features, r a relationship, A an artefact, c a concept, O_s a sub-ontology and O_p a product ontology.*

$$(\forall\, f \in F \wedge F \in A\ ) \exists\ (\ c \wedge r\ ) \in O_s \subset O_p \Rightarrow TraceMatch(c, r, F)$$

When two concepts of different artefacts share the same match, then we can say that there is a trace link between them.

Definition 10. *Let be A an artefact, c a concept, r a relationship, O_s a sub-ontology and O_p a product ontology.*

$$(\forall\, f \in F \wedge F \in A\ ) \exists\ c \in O_s \subset O_p \Rightarrow TraceMatch(c, F)$$

5.3 Algorithms

The previous formalization is the base to create logical expressions and the algorithms to generate and merge ontologies. Such algorithms are used in the proof-of-concept built as part of this study.

5.3.1 Train model

To create domain context first, it is necessary to build the vocabulary to such domain. The vocabulary is represented as a matrix of vectors of each word and stored in a model.

To establish the context of the vocabulary, text data about the domain contain implicit ubiquitous information about the context of each word. This implicit information is present according to the neighbors of each word. The model is trained with such information in a context matrix with previous built vocabulary. With more train data there are higher probabilities to have a better context of the domain.

The detailed sequence is described in the algorithm 1.

Algorithm 1: Train Domain Model

Data : The artifact instance in natural text

Result: A trained model

```
1 Read artifact instance
2 split artifactData in lines dataLines
3 for all dataLines do
4     split dataLines in words wordData
5     for all wordData do
6         build vocabulary with word vector
7     train(model) with dataLine
8 return model
```

5.3.2 Sub-ontology extraction

The extraction of ontological data has the artifact in natural text and divided by sentences, each sentence expected to be in a different line.

The ontological information is extracted from each line in form of axioms and classified according to the different patterns. The classification can be as a concept or relationship. Once it is classified then the axiom is added to the sub-ontology and continue the extraction process with the rest of the lines.

This extraction from a software artifact is summarized in the algorithm 4.

Algorithm 2: Sub-ontology extraction

Data : The artifact instance in natural text

Result: An OWL sub-ontology

```
Read artifact instance
split artifactData in lines dataLines
for all dataLines do
    T ← get dataLine dependency and POS tree.
    axioms ← identify patterns in T
    for all axioms do
        if axiom is a concept then
            ax ← axiom concept data
        if axiom is a relation then
            ax ← axiom relation data
        add( ax ) to subOntology
    update subOntology
return subOntology
```

5.3.3 Match and merge onologies

In order to identify a sub-ontology in a product ontology, it is necessary to parse each concept of the sub-ontology and find a similar term in the product ontology. When two concepts are identified as similar, then a match is created between both concepts. Once a match is identified the new relations that involve the matched concept should be added.

To resolve if two concepts are similar or not, there is a first comparison of both concepts using the *CommonCrowl* and *OntoNotes* trained model as common context, If two concepts are considered similar based on certain parameter, a second similarity comparison is executed taking the dot product of the common context model and the domain context trained data model to resolve the similarity based on a predefined parameter. The similarity is automatically calculated, taking the cosine of the euclidean product of both context word vectors.

If a concept is not identified in the product ontology, then it is added to it as a new concept. The process is described in the algorithm 3.

Algorithm 3: Ontology merge

Data : The artifact instance in natural text and Product Ontology in OWL language

Result: A merged ontology

```
1  Read artifact instance
2  Set common similarity parameter
3  Set domain similarity parameter
4  Read productOntology instance
5  Assign all elements of productOntology to a productOntologyMerged
6  Extract subOntology from artifact instance
7  for all subOntologyConcepts do
8      for all productOntologyConcepts do
9          if subOntologyConcept is equal to productOntologyConcept then
10             change subOntologyConcept attribute as equivalent
11             add(subOntologyConcept) to productOntologyMerged
12             add(subOntologyConceptRelations) to productOntologyMerged
13         else
14             s ← commonSimilarity(subOntologyConcept,
                 productOntologyConcept)
15             if s above parameter then
16                 s ← commonAndDomainSimilarity(subOntologyConcept,
                     productOntologyConcept)
17                 if s above parameter then
18                     change subOntologyConcept attribute as equivalent
19                     add(subOntologyConcept) to productOntologyMerged
20                     add(subOntologyConceptRelations) to
                         productOntologyMerged
21     if no equivalent concept found then
22         change subOntologyConcept attribute as new
23         add(subOntologyConcept) to productOntologyMerged
24         add(subOntologyConceptRelations) to productOntologyMerged
25 return productOntologyMerged
```

Algorithm 4: Common and domain similarity of two concepts

Data : Terms a and b

Result: Similarity probability p_c

1 $y_a = P(a|commonContext) * (a|domainContext)$

2 $y_b = P(b|commonContext) * (b|domainContext)$

3 $p_c = (y_a * y_b)/(\parallel y_a \parallel * \parallel y_b \parallel)$

4 return p_c

Chapter 6

Proof-of-concept Implementation

This chapter describes a proof-of-concept created to generate ontologies and sub-ontologies and identify trace matches between artefacts in a given product ontology.

6.1 Scope

The scope of this PoC is restricted to the extraction and visualisation of ontologies and sub-ontologies from user stories that follow a proper user story template. The generation of a sub-ontology is automated using NLP techniques. However, the product ontology can be either generated from a set of user stories or created manually using Web OWL language.

Based on ontology generation and formalization, a proof-of-concept is developed to generate and identify ontologies and sub-ontologies supported by a graphical interface under the following requirements. These basic requirements are:

1. Easy to integrate with other technologies.
2. Graphical output in a web environment.
3. Generation of ontologies from the text written in natural language.
4. Comparison of ontologies and identification of coincidences.

6.2 Technology

To fulfil the basic requirements, a combination of different technologies are selected considering the time constraints. The extraction and generation of ontologies are developed using Python.

Web OWL is the language specification used to describe an ontology and WebVOWL 1.1.4 for the visualization of an ontology using web owl language.

WebOWL is a web application developed in JavaScript. The use of JavaScript facilitates its integration in web environments such as GitLab, GirHub, or any other web platform. WebOWL was created to visualize ontology elements such as concepts, relations, data types in the shape of a directed graph.

The mean to communicate between Python and WebVOWL is though JSON files defined by a web owl language to facilitate the transfer and interaction of the ontologies between systems.

In Web OWL language a $concept$ is named as a $class$, a relation between two concepts as a $propertyattribute$. Therefore our definition of the relation between two concepts can be redefined as:

Definition 11. *Property attribute*

Let be c *a class* $\exists C$ *: Set(Class).(* $\exists$*.PropertyAttibute*$(d, r) \land d \in C \land r \in C$*) where* d *is a domain and* r *is the range.*

6.3 Architecture

The system can be deployed in a Linux server. All components run in an Apache web server as an open-source cross-platform as can be seen in figure 6.1. Its execution using Eclipse is optional and convenient when running a stand-alone. The interface with the user is through a web browser.

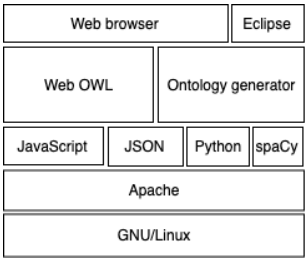

Figure 6.1: Ontology generator product development context

The system has four main modules, the ontology extractor, the NLP ontology generator, and OWL object factory, the ontology merger and the OWL object factory as represented in the functional architecture in figure 6.2.

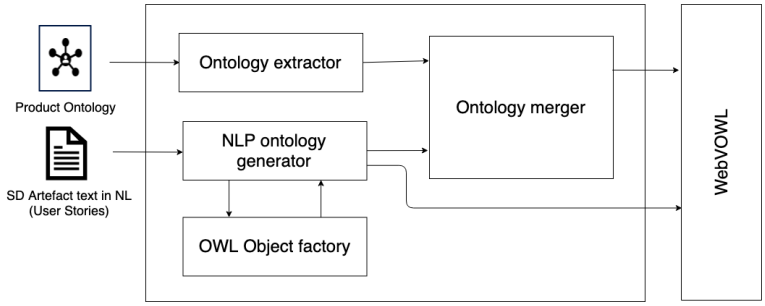

Figure 6.2: Ontology generator product development context

6.3.1 Ontology extractor

The ontology extractor takes as an input an already existing ontology in web owl language. This ontology can be a product ontology that contains all the concepts that describe the current system. A product ontology can be created manually in WebVOWL, generated from a software development artefact as user stories using the NLP ontology generator or a combination of both.

It is very rare that already exist an ontology of a product or make it available. Therefore, the NLP ontology generator automates this task that automatically an ontology can be generated. However, the generated ontology may or may fully represent the current system features. Therefore, there is a flexibility that the ontology can be manually manipulated and changed according to the architect or any other role that has a complete picture of the product.

If changes are performed in the product ontology using the graphic interface that provides WebVOWL, then the ontology can be exported again in a JSON file format and stored locally. This brings high flexibility, especially when concepts are not linguistically directly related, and further descriptions need to be changed or added.

The ontology extractor reads this web owl statements and transforms it into an internal structure of lists. The module does not perform any transformation of the data or tags, only the data structure. It is assumed that the syntax of the ontology is correct as no consistency checks are additionally performed.

Finally, the module makes available the ontology in an internal structure to the ontology merger module for further processing.

6.3.2 NLP ontology generator

This module is responsible for reading the software development artefact. Using natural language techniques, it can generate a sub-ontology, or a product ontology, in case that

the selected artifact describes the whole product.

To generate an ontology, it is necessary to read the artefact and keep it in a structure that can be available for its processing. Each line represents a statement containing only the basic elements of a user story. This means, to have only one role, one action verb and a predicate but does not require to use a user story template. However, it needs to be written in an active voice, e.g. ”the user should be able to change his password...” instead of ”...the password is changed by the user”.

The NLP ontology generator parses line by line of the file and divides into tokens, POS tagging and dependencies taking into consideration only noun chunks.

Using the tagged information, a pattern recognition is performed using the corollaries previously described and extract the relevant ontological information of each sentence. This information is classified according to their properties.

When a group of axioms are identified, then the module requests the OWL factory the create requested objects. This factory is in charge to generate and combine the axioms according to its syntax as axiom objects. These objects are represented as classes, class attributes, properties and property attributes that keep the ontological information in a simplified version of the OWL language. For instance, when two concepts and a relation between them are identified, the ontology generator requests the object factory of such sub-ontology objects.

There are other characteristics or attributes of a concept that are identified by the pattern recognition. Such as additional information is included in the ontology as part of the description of the concept and helps to describe the concept.

Once the parse of the entire file is parsed, the ontology generator requests to object factory the generated data of the sub-ontology. Then it makes it ready and available for the use of the ontology merger module in an internal format.

6.3.3 OWL Object factory

The OWL object factory is the most object-oriented programming (OOP) module. The main function is to generate axiom objects based on the ontological information provided by the ontology generator.

As described in previous chapters, the basic unit in the web owl language is the axiom. An axiom is represented as a class in OOP. An axiom has specific attributes like id, axiom name and type and respective methods to return these properties. Each object knows the syntax of their attributes and is capable of returning its attributes in the owl language syntax.

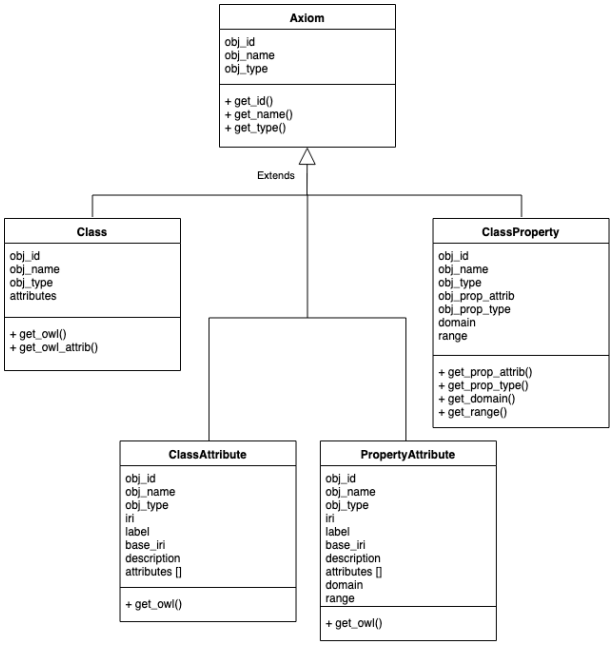

Figure 6.3: Class diagram of the OWL object factory

Different classes inherit from the axiom class such as Class, ClassProperty, ClassAttribute and property attribute as described in the class diagram of figure 6.3. This class design allows the factory to generate the objects. It does not need to know the syntax of the web OWL language, as this is encapsulated in each axiom of each class. Therefore, this is transparent for the factory.

6.3.4 Ontology merger

The product ontology and the artefact sub-ontology are the main inputs of the ontology merger. These ontologies are combined by searching each axiom of the sub-ontology in the product ontology.

The ontology merger takes a class axiom of the sub-ontology and looks for similarities in the axiom classes of the product ontology. Once it is determined that two classes are similar, then we can say that we found a match of concepts or trace match. The sub-ontology class is created as an equivalent type class. Graphically this can be seen represented as a circle with a double border.

The comparison of concepts is performed using word vectors and semantic similarity. These vectors are grouped in a model. This model can be extended by including vectors of new vocabulary or removing them. Word vectors can be generated using different models like the bag-of-word model (CBOW) introduced in Mikolov et al. (2013). This study takes the model “en_core_web_md” provided by spaCy. This model is trained

with vectors of words in English and other languages. Based on such vectors, it is possible to determine the proportion of similarity between two words from 0% to 100% similar.

Depending on the model, the proportion then comparing words is higher if the model is trained with words in the domain of the ontology. If we talk about a health product systems, vectors with words of medicine vocabulary can increase the precision of finding the right trace matches.

This study takes the basic model with a proportion of similarity between two sub-ontology name words above 90% to identify a concept match. If the sub-ontology name does not match, then a second search is performed in the description of the product ontology in order to find a similarity with the same criteria.

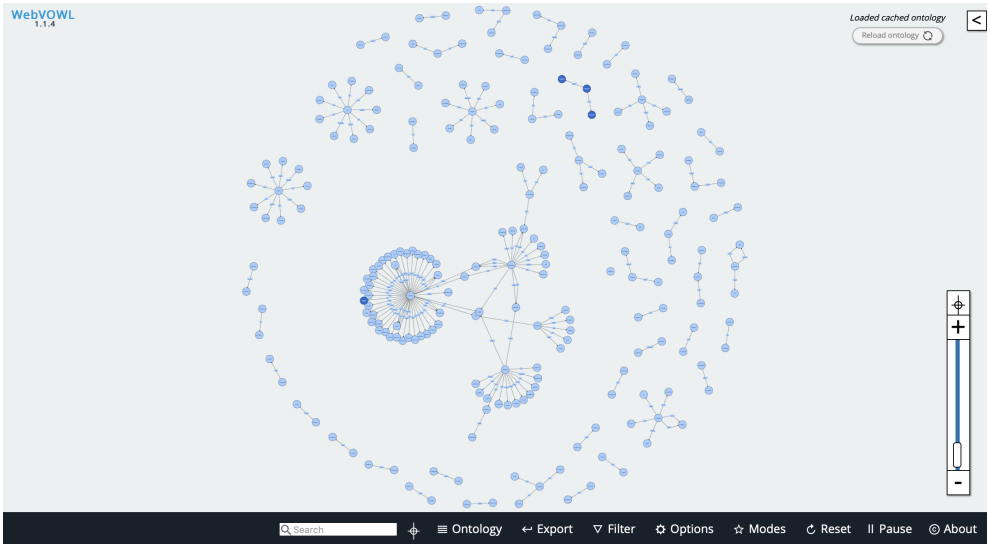

Figure 6.4: Graph visualization of WebVWOL

If an axiom name is not found in the product ontology, it can mean that it refers to a new feature that it is not implemented yet. In this case, a new axiom is created in the product ontology and identified as “external” to differentiate them from the rest of the ontology. In WebVOWL this is represented in dark blue circles and can be easily recognised from the rest of the ontology.

Once the sub-ontology is merged and identified in the product ontology, a new product ontology is generated and can be graphically displayed using VewVOWL. Figure 6.4 is an example of the result of this process. The sub-ontology indicates a new functionality in the form of new concepts represented in dark blue circles and the rest of the product ontology in light blue circles.

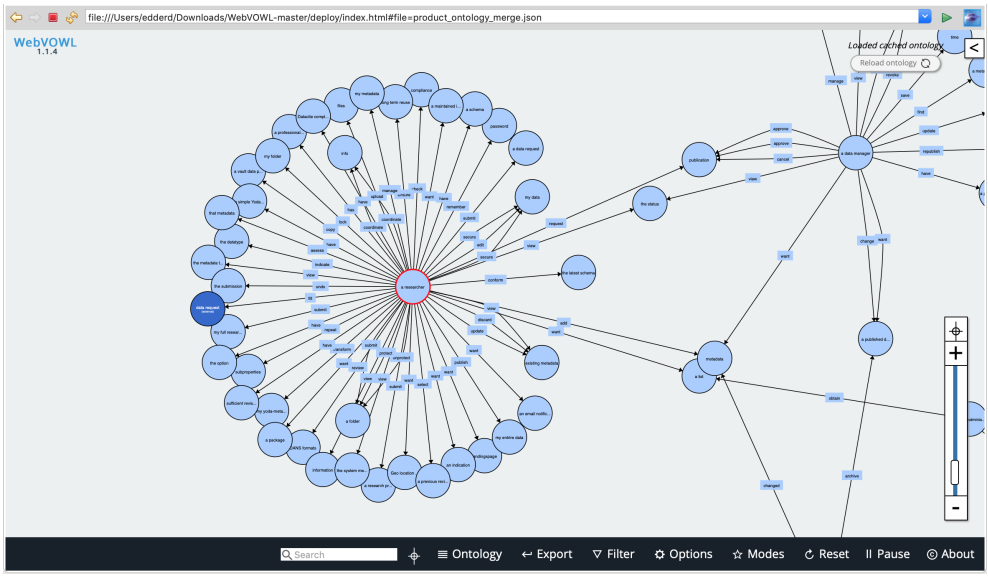

Figure 6.5: Graph visualization of WebVWOL

A better perspective of part of the graph can be seen in figure 6.5 where actions are concentrated in a few roles, e.g. *"a researcher"*. A *"data request"* concept is added with a further description *"a research proposal"* from the sub-ontology. The user story that generated this concept is written as *"As a researcher, I want to fill data request including a research proposal so that I can submit it as a whole"*

Chapter 7

Experimental Results

This chapter describes the proof-of-concept as a treatment, goals, variables, hypothesis and data collection procedure of the experiment that answers the research questions.

7.1 Experimental setup

7.1.1 The treatments

Ontology extraction and ontology merge are the main solution proposed to the practical problem of between the requirements engineering and the software architecture through traceability among artifacts.

From one side, the ontology extraction intents to conceptualize the requirements that re written in natural text. From the other side, the Ontology merge in the PoC automates the match of ontologies and identification of sub-ontology concepts.

To determine the relative accuracy of the theory raised by this research, the proof-of-concept is used as a treatment artefact. This PoC implements the main concepts described previously in this study.

7.1.2 Experimental goal

The experimental set-up entails to study at what level the treatment in study facilitates the extraction and identification of sub-ontologies in a PO.

A set of independent variables are used as an input to be used by the treatment. The results are measured in different dependent variables relevant to measure the goal of this setup.

The main goal of the experiment is to determine the quality of the treatment with

different independent variables. We define quality as a general measure of completeness, efficiency, performance, usefulness and viability, as shown in figure 7.1.

Object of study	Purpose	Focus	Stakeholder	Context factors
Ontology merge PoC	Measure variables	Quality	Architect, Software developer	RE4SA, university

Table 7.1: Experiment definition.

The completeness is measured using precision and recall, having an ontology manually created manually from one of the artifacts to be measured and considered as our gold standard.

To measure the efficiency, the number of correct concepts identified or matched divided by the time that take the process is taken.

The performance is measured taking the time of the process with either the extraction or the match ontologies process.

Finally, to evaluate the usefulness and the viabilty,

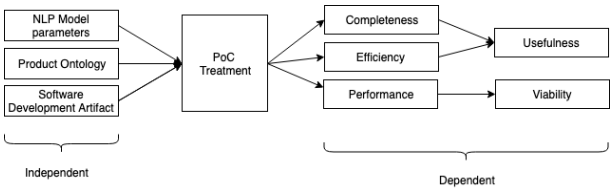

Figure 7.1: Independent and dependent variables. An arrow between two variables suggests there is some relation between the two, i.e., a change in the one variable can lead to change in the other one as well.

7.1.3 Hypothesis

We take part in the research questions and create a hypothesis on each to help reach this goal.

- RQ_3 : H_1. There exist at least two NLP techniques that are suitable to identify relevant ontological information from a software development artifact.
- RQ_4 : H_2. When using two NLP techniques and recognizing patterns, there is suitable ontological information to identify sub-ontologies related to a software development artefact.
- RQ_5 : H_3. When using NLP similarity techniques to match concepts, there is ontological information to identify a sub-ontology in a product ontology.

Variable	RQs	Metric
Completeness	RQ_3	Precision and recall
Efficiency	RQ_3 RQ_4	Calculated as effective match / time
Performance		Time taken to generate and merge ontologies.
Usefulness	RQ_5	Average survey grade.

Table 7.2: Variables and their metrics

7.1.4 Design

The set-up for the dependent values is divided into two phases. The first phase involves the execution of the proof-of-concept in two activities. The first activity is designed to evaluate the dependent variables when generating an ontology and the second activity to assess the variables when identifying a sub-ontology in its respective product ontology.

The second phase consists of the execution of different interviews to scrum and non-scrum practitioners, showing the core concepts of this study and the application of a survey. The results of the survey can give an idea about the viability and usefulness of this kind of tools with people from the industry.

Based on the independent variable selection, a group of data set consisting of user stories are collected. These independent variables are the input for the proof-of-concept under the same conditions in different sessions. Dependent variables are calculated from observed values produced by the proof-of-concept and the application of surveys after executing the generation and identification of sub-ontologies.

7.1.5 Subjects of study

A group of 6 people with an IT background, especially in software development, are selected to perform the demos of the system. It was required to have a working experience of at least five years. 66.7% of people interviewed had worked in Agile environments for some years, meanwhile the other 33.3% not at all. None of the subjects had an idea what the demo will be about before the execution of the interview.

7.1.6 Context

To execute the test in a real-life problem, a request for collaboration with development experts of Yoda system was requested and generously accepted. Yoda (acronym of your data) is a system that provides an integrated digital environment for researchers. Yoda was created to facilitate the daily work of researchers of storing, sharing and publish information.

The interviews and the demos were scheduled individually. In some cases, via remote, the person was not facially in the country at the moment of this study.

Business professionals are selected to take part in the interviews, including developers as agile practitioners, consultants and agile practitioners.

The same example demo was performed to all people interviewed. A definition of what an ontology was given to the interviewees. Then, an explanation of the concepts of what a product ontology and a sub-ontology are. After that, a demo of the system was executed with the option to ask some questions. Finally, the filling of a survey about the shown functionally concerned to this study was carried out.

7.1.7 Objects of study

The proof-of-concept is built as a back-end and added as an extension of the open-source system WebVOWL that is used as a graphic interface to the users. The main object of study is the added functionality added to the front-end and delivered as a proof-of-concept. This was explained to the people interviewed in order to void confusion about the concept and the functionality provided by the front-end.

7.1.8 Independent variables

The first independent variable is the linguistic model used to perform the NLP processing. For this study the *sayen_core_web_sm* v2.1.0 model is used for tagging and similarity recognition. The model has 20,000 unique vectors with 300 dimensions. The parameter to consider if two concepts are similar is set to ¿90%.

The second variable is the custom model trained with data from the product ontology.

The third variable is a product ontology generated beforehand with a set of user stories with status ”done” from the product backlog.

The fourth variable is software development artefact. This artefact is in the form of user stories written in natural language in a text file. Each line in the file is a different user story. For this study, a random set of 50 user stories that belong to different epics with status ”to do” are selected.

7.2 Execution

The first phase is carried out in a controlled environment where the tool is deployed and 100% available only for the researcher for its use.

The product ontology is uploaded into the server in a *son* file and then the artefact with the new requirements is also uploaded in a *txt* file.

Once both artefacts are uploaded, then the sub-ontology is generated, and the result is visualised as a graph using WebVWOL. Then, the number of right concepts and relations in the produced sub-ontology are counted, the same for the number of concepts and relations ignored.

With the product ontology already uploaded and the sub-ontology generated, the next step is to proceed with the identification of the sub-ontology into the product ontology. This action is performed by triggering the tool and observe the results.

The second phase of the study is performed to each person in a semi-structured interview, followed by a demonstration of the proof-of-concept and the filling of a survey about the concept. Then, the results are collected for further analysis.

7.3 Results

The results are divided into two result sets. The first set to evaluate the performance of the proof-of-concept when generating a sub-ontology, and the second set with the results produced by the identification of a sub-ontology in a product ontology match.

7.3.1 Sub-ontology generation

To perform this task, it was necessary to take out a few user stories that were not clear, incomplete or in a different format. The final data set contained 39 user stories. The time to generate the JSON within the OWL language lasted approx 18 seconds. The generation was the only process in execution at the moment of the test.

From the data set, it was expected to find 12 roles, 31 relations and 37. The expected concepts and relationships were analysed and translated into ontology terms manually, as seen in table 7.3.

From the results obtained, it was possible to observe that all roles involved in the sub-ontology are identified. However, the precision decreased when identifying concepts related to those roles and as a result, the related relations between them as well.

A relation between concepts gets lost as a result of the missing target concept. The main reason for this problem is that few user stories are written in passive voice or there is a possibility that two roles are involved. This can be reflected in the recall result in table 7.5.

Role concept	Relation	Target concept
Geo researcher	store	my HPTlab
	annotate	my TEClab data
	have	EPOS metadata schema
Researcher	Fill	data request
	split	firstname/lastname
	receive	email notifications
	download	system metadata
	include	my metadata
	enter	metadata bounding box coordinates
	depublish	an archived data package
	specify	specify my data package type
	locate	a data package based on its EPIC PID
	want	help
data manager		
	want	my metadata schema
	want	all persistent identifiers of a package
	archive	a published package at DANS
	want/know	group within my category
	know	which groups a user belongs
	have	a search group feature
	detect	inactive groups
	approve	a data request
board member	view	research proposal evaluations
DMC member	review	a research proposal
Admin	display	the actual checksums
	have	an overview of all user autorisation changes
User	see	bounding box
	access	Yoda
	reference	a data object
iRODS admin	keep	indefinately keep info
Yoda admin	refactor	Intake module GRP groups
data receiver	want	login and have a list of jobs shared with me
Ron researcher	download	a folder and its content
Data sender	share	my distribution job
	add	data to my distribution job
	want/view	my distribution jobs and their details
	create	data distribution job
	want	login the data distribution service

Table 7.3: Variables and their metrics

		Actual values		
		Positive	Negative	Total
Predicted values	Positive	73	6	80
	Negative	13	2	15
	Total	86	8	N

Table 7.4: Precision and recall matrix for sub-ontology generation

Measure	Result value
$recall$	0.85
$precision$	0.92
$F_1 score$	0.88
$Efficiency$	$4\ /s$
$Performance$	$18s$

Table 7.5: Sub-ontology generation in OWL language

7.3.2 Sub-ontology match

Once the product ontology and the sub-ontology are available, it is possible to start the process to identify the sub-ontology in the product ontology.

As described before, it is necessary to identify similar concepts shared between the sub-ontology and the PO. This matching of concepts is carried out comparing every concept in the sub-ontology and find a similarity of terms in the PO. Once it is determined that tho concepts are similar then a match is created and the new concept is declared as an *equivalent class* in terms of OWL language.

The concepts identified correctly in the sub-ontology generated was taking in to account for the matching analysis.

The total time of the matching was 35.45 seconds. The poof-of-concept was the only process in execution in the server, and the graphical interface provided feedback once the process finished.

To facilitate the task of the analysis, the *json* file was downloaded from the server. The file containing all classes and attributes was analysed manually to evaluate the correct matches in the PO and the identification of new concepts as new classes added to the ontology. The new classes are identified as external in OWL language as they are still not implemented yet.

The program identified correctly 40 matches out of 41 expected. All roles were , but few concepts without a match detected.

Concepts that were identified similar because of the context used were also identified, *system* and *metadata*, *Yoda* and *Yoda system*, *a folder* and *research folder*, *EPOS metadata schema* and *metadata*, *Researcher* and *Geo researcher*, *datamanager* and *data manager* are some examples.

		Actual values		
		Positive	Negative	Total
Predicted values	Positive	40	2	42
	Negative	1	0	1
	Total	41	2	N

Table 7.6: Precision and recall matrix for sub-ontology match

The precision and the $F_1 score$ range above 0.95 with a calculated recall of 0.98. However, the efficiency decreased significantly. One factor that heavily impacts the performance is the fact that the look for similarity between terms. The similarity triggers a task that calculates the translation from word to vectors. Multiple sums or dot products are automatically triggered, first, the word-vector against the vocabulary-matrix and second, with the context vector-matrix to identify the context of the two words. Another factor is the complexity of the code due to the need for nested iterations to process graphs or lists as internal data structures.

Measure	Result value
$recall$	0.98
$precision$	0.95
$F_1 score$	0.96
$Efficiency$	0.19 c/s
$Performance$	425s

Table 7.7: Sub-ontology generation in OWL language

Measure	Ontology generation	Sub-ontology match	Combined
$recall$	0.85	0.98	0.91
$precision$	0.92	0.95	0.94
$F_1 score$	0.88	0.96	0.92
$Efficiency$	4 $/s$	0.19 c/s	2.1 /s
$Performance$	18 s	425s	221.5

Table 7.8: Combined results of ontology generation and match

7.4 Experts evaluation

A critical aspect of this study is to analyse if the concept of generating ontologies from text and identifying them in a product ontology. Then, contributing to the identification of trace links and, as a result, the possibility reaches the desired goal.

It is essential to analyse and evaluate that the theory presented by this thesis can have a real benefit in the area of study. The implementation of this concept can provide real beneficial value for software development practitioners to facilitate their work.

The opinion of experts in the area is a good indicator to assess if the current path is appropriate or not.

The second phase of the experimental setup involved an interview with experts with long experience in the IT industry with and without experience in Agile methods.

All experts interviewed had small or no notion about what an ontology is. Once a definition was given, then the relation with a conceptual map, conceptual model, knowledge map was related in some cases. At the moment of the interview, 83% had more than ten years in the IT industry.

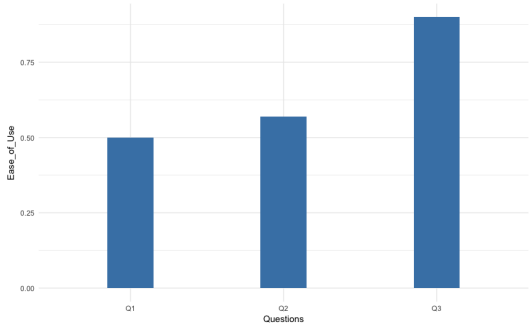

Figure 7.2: Ease of use

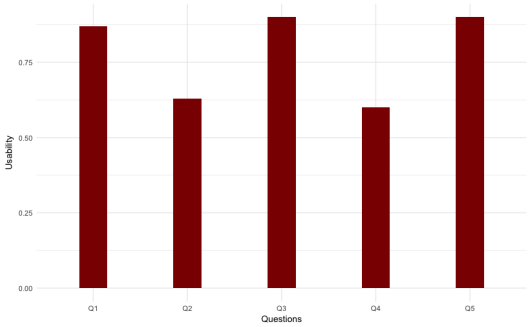

Figure 7.3: Usability

One of the hypothesis is that it is possible to extract relevant ontological information from natural text. The facilitation to transform from text was pointed out by one of the interviewees:

Passing from text to process and probably solution design.

Another hypothesis is that there is enough ontological information to identify a sub-ontology in a PO. This hypothesis can be supported by the idea to have an overview of the whole system in one single artefact was highlighted for one of the interviewees:

Gives a clear one-view overview, and it seems very flexible.

To further confirm this impression of the concept, a survey is performed. The survey evaluates the interviewee's impression about the ease to use of the concept itself, the usefulness and the intention to use a tool with the characteristics shown.

The survey is filled by each interviewee based on the perception of the whole concept of the functionality, no in the interface itself.

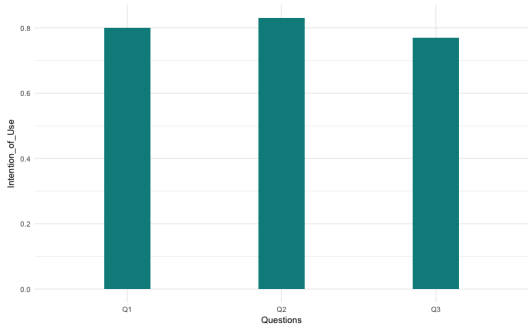

Figure 7.4: Intention of use

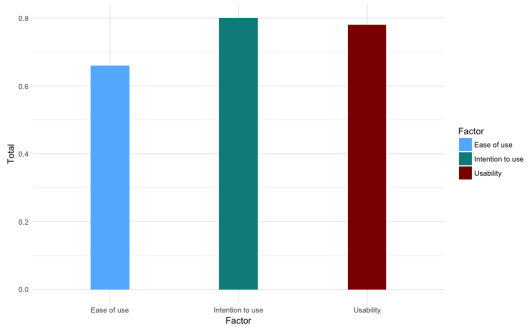

Figure 7.5: Totals

Three factors are indented to be measured with this survey. The first factor measures their perception of the ease of use. Despite that the interviewees can not have a complete manipulation of the proof-of-concept, it is asked about their perception from the demo shown.

The second factor is about their perception about the usability of the tool to have an idea if they believe that the concept can be useful if it is integrated into the development process.

The third factor is about their intention to use a tool that applies such ontology concepts.

Finally, all results are tantalised, as shown in figure 7.5. These results give an estimation of the overall factors and have an indication about the the purpose of this study from experts opinion.

Chapter 8

Discussion

The importance of the relation between Requirements Engineering and Software architecture is very well known. The adoption of new methodologies has increased the attention to straightening this relation. Such is the case of Agile and Scrum methods where small increments. Each increment has to deliver specific value to the organisation implementing it.

This thesis explored two main problems. First, the fact that requirements are collected, gathered and stored in the best case in a semi-formal format written in natural text (e.g. User Stories, Epic stories, etc.). And second, the lack of support to find a trace between the requirement and the architecture. These two problems prevent an effective delivering of value on each increment due to inconsistencies or delays in the process.

The solution explored to solve these problems is the use of an intermediate layer between the requirements and the architecture defined as an ontology. Taking the requirements as the starting point, perform a linguistic analysis and finalise with the creation of a proof-of-concept produced from the finding of this research.

The results of the literature review and the available information from software development artifacts were used to gather the essential elements to develop this thesis. Research methods, linguistic techniques and tools found were part of such elements that contributed in sequence to solve each research sub-question.

The following sections will detail how each research sub-question was gradually answered and how the clarification of the main research question came to light. Subsequently, the conclusions are drawn in this research. Finally, the threats that can affect the validity and hints for further research are discussed.

8.1 Answering the sub research questions

The five sub research questions formulated in this thesis were gradually answered in the same order as they were formulated. The answers of each sub question served as an input for the next question formulated. To facilitate their identification, the sub questions in three sections. First, those concerned to the generation of sub-ontologies. Then, the ones related to the identification of sub-ontologies in a Po ontology to finally find the links among the artefacts.

8.1.1 Generating ontologies from requirements

Requirements drive every software product. From there the importance of their correct generation and use.

Up to now, most of the requirements are created in natural language and does not follow a clear formal structure for its creation. Therefore, the first research was created to find the necessary information mostly from the literature.

In the literature review it was possible to find state-of-the-art concepts, models, techniques and tools that can be used in the the processing of natural language.

The first finding was that NLP is a broad field with different techniques that concerns with the interaction between humans and computers.

Tokenization, Part-of-speech, dependency parsing, entity recognition between others were identified as techniques to detect patterns from software artefacts in natural language. Both techniques provide a representation that can be formalized in an algorithm or abstracted in a model.

The second finding is the availability of different tools that automate the process of annotation of text as a corpus. Between those tools it was possible to identify the four most reliable and open source, Stanford CoreNLP from Stanford University, SyntaxNet and spaCy under MIT license and NLTK from NLTK project.

Each tool was compared and evaluated to determine what would be the most suitable for this study. spaCy was at the end selected due to its facility to use it in software production environments, the available models and the facility to integrate with different systems though python development.

The third finding was the use of convolutional neural networks with word prediction models like Word2vec to help in the problem of ambiguity and the similarity of concepts. In the following subsection, we will discuss further these models.

These three findings contributed to solve the first sub research question stated as:

> RQ_1**:** "How can relevant ontological information from system development artifacts written in natural language be extracted?."

To help to solve RQ_2 and RQ_3, a set of artifacts were created and used to perform a linguistic analysis in several iterations. Each iteration was consisting of the selection of one NLP technique, the creation of an artifact to automate the tagging process, perform the annotation and analyse the results. The process of identifying patterns started bottom-up from the syntactic level towards the pragmatic level.

The first two iterations provided significant pattern information. However, each pattern individually did not give enough knowledge to create an ontology that could reflect the purpose of the requirement. Therefore, in the third iteration, it was decided to combine both techniques. As a result, it was possible to identify a more precise pattern and enhance the conceptual model of the structure of a user story with NLP ontological information.

> RQ_2 "What patterns from an annotated text in system development artifacts written in natural language can be identified?"

> RQ_3**:** "What NLP techniques are suitable to identify sub-ontologies in an annotated text in software artifacts written in natural language text using NLP?"

From the experimental results of the proof-of-concept and the surveys performed to expert, it is possible to identify that the generation of ontologies is very useful in terms of precision and ease of use. The precision was measured with 0.92%, which is a good indicator of the usability of the concept. However, the recall is not as good with a score of 0.85%, indicating that there is still some space of improvement in the generation of sub-ontologies, possibly by adopting another NLP technique that helps to increase the precision and recall.

8.1.2 Indentifying sub-ontologies in a product ontology

One significant characteristic when using ontologies is that it is possible to identify dependencies in artefacts and with other concepts. Therefore, the importance to define a sub-ontology of new requirements in the product ontology.

The simplest way to do this is by selecting each concept of the sub-ontology and compare it with each concept of the PO and iterate until all concepts are identified.

However, a simple comparison may not be precise as some concepts can be similar and be the same or be similar but not the same concept.

The use of Word2vec models is an excellent approach to tackle such kind of problems derived from the introduction of ambiguous terms. Such model enhanced with a third dimension representing the domain, it is possible to increase the accuracy in the resolution of ambiguity.

From the concepts marked as similar and considered the same concept we can find, for instance, the similarity between *dataset* and *data Geo researcher*. Using the Glove model, the similarity between both terms scored higher than 90% whereas with the domain model scored less than 69%. Another example observed is the similarity between the concepts *my data* and *my HPTlab data*. The first concept scored high with the GloVe model but less than 80% with the domain model.

One subjective aspect of this model is up to what extent you can say that the two concepts are similar or not. What would be a proper criteria to say that two concepts are similar having the probabilities of a word given a context, We could say that below 90% of probability is not considered the same concept The other aspect is if there are enough train data to do this determination, if not the model needs to be trained with more domain information with the context in order to perform better.

Probably this may not solve the complete problem but is a right approach when resolving the ambiguity. Therefore, there is enough information available to answer the sub-question RQ_4**:**

> RQ_4**:** "How can the identified sub-ontologies be related to a software development artifact ontology?"

8.1.3 Finding links between artifacts

The creation of trace links will not be possible without the use of a common language. A common language that is formalized, standardized and accepted in the software industry to construct ontologies. This language helps to encode knowledge about a specific domain.

There exist several ontology languages. The selection of the language depends on the needs of the stakeholders in question. There is no right ontology language, but the adoption of one is a must when working with ontologies if you expect to share that knowledge in an automated way.

The WebOWL markup ontology language is used based on the needs of this study and the need to have a formal language that can interact with the different modules of the proof-of-concept.

For the proof-of-concept, a simplification of the WebOWL is created but can be extended to reflect more needs of the stakeholders like inheritance or detailed data types.

In the proof-of-concept, the NLP extractor collects the ontological information. Based on the patterns found, classifies the text from the artifact in a sub-ontology. Then, it is translated to the OWL language by the OWL factory.

From the product ontology, you can have each module or feature in the system represented as an OWL class. An OWL class can be equivalent to another class. If a feature or a group of features have an equivalent class in the product ontology, then it is clear that a trace link can be identified between the ontology and the feature. The creation of feature or module classes, however, is not part of this study.

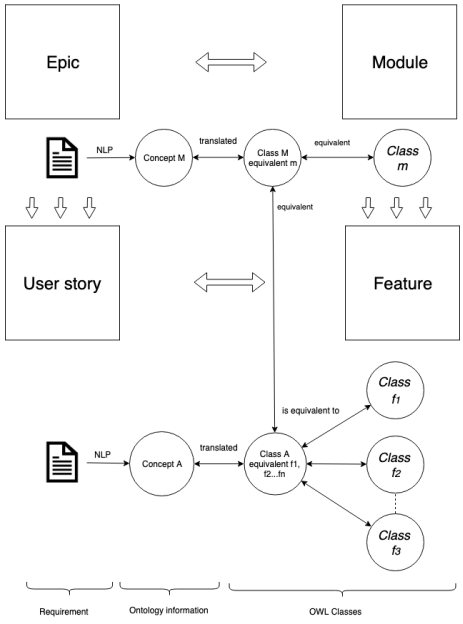

Figure 8.1: Equivalent classes and trace links when matching concepts in OWL language

When we proceed to identify a sub-ontology in a PO, we identify a match when two concepts are linguistically similar in the common and product domain. This match is recognized as an equivalent value in the OWL ontology language.

When this match is established it is possible to track all the equivalent values of the class and, as a result, the links between the artefacts either modules or features as can be seen in figure 8.1. This answers the sub-question RQ_5.

> RQ_5**:** "How can a direct trace link be derived from a sub-ontology to specific system development artifacts?"

8.2 Main research question

In order to answer the main research question, five sub-questions are created to divide the problem into different parts. Each sub-question provided certain value to help in the answer of the *MRQ*.

> **MRQ:** "What kind of relevant ontological information can be extracted from annotated patterns of software artifacts using natural language processing that facilitates to trace links between them in the Software development process?""

Taking each answer to each sub research question, we can summarise that:

- The extraction of ontological information can be done with available NLP techniques and tools.
- Part-of-speech and dependencies are suitable to discover patterns in text artifacts.
- It is possible to create an ontology-based on such patterns and using a defined ontological language.
- A sub ontology of an artefact can be identified in a product ontology.
- Once a concept in a product ontology matches with a concept in the sub-ontology an equivalent relation is created and, therefore, it is possible to find trace links between artefacts.

As a result, the relevant ontological information extracted from a software artifact, in this case, user stories, are: concepts relations between concepts, external concepts and equivalent concepts and the trace links are automatically derived from each equivalent concept as described in figure 8.1.

Taking the experts' survey results is possible to have an indicator that the integration of anthologies in the software development process may not be too difficult to adopt and possibly will reduce the development time and increase the value delivered to the stakeholders.

8.3 Conclusions

As part of this study, several sub research questions are formulated to answer the main research question. With the main research, question answered, we can draw a conclusion about this study and its process.

The creation of small artifacts or sub-artifacts helps to reduce the time the analysis of NLP annotation, especially when a big corpus need to be analysed with hundreds of line statements.

The combination of at least two NLP techniques is necessary to extract relevant knowledge from text. If they are combined with a convolutional neural network, the effectiveness of finding this knowledge can be increased.

The use of ontological tools as a layer between the requirement engineering and software architecture can provide a good solution to reduce the software development time and reduce the inconsistencies between them. Ontology match and artifact links are examples of how this time can be reduced by automating the task to find the traces between the requirements and the software architecture.

Dependencies between concepts are more clearly seen in an ontology. This reduces the risk to have inconsistencies in the architecture.

Finally, this can be part of the solution of the traceability problem that currently exists between the Requirement Engineering and the Software Architecture together with current methods available and implementing models like RE4SA that contribute in the formalization of the process of software development.

8.4 Threats to validity

In retrospective to the validity treats, some factors are identified from the process of this research.

8.4.1 External validity

Several external threats can be involved, especially when working with text written in natural language. There was a special effort pot on to mitigate those threats. Nevertheless, there is always a possibility to have them when coming from external sources. For instance, the training data to resolve similarity may not be enough to determine the complete domain of the ontology. New ambiguous terms could have been accepted or rejected because of lack of domain context.

For this study, only one case was selected to test the proof-of-concept. A further test is needed using different data sets and the context domain to validate the accuracy.

From the people interviewed, there is a possibility that their opinion may not be the same when manipulating the tool. However, as per their experience and complete independence from this study, there is no reason to think that their opinion could change

easily.

8.4.2 Internal validity

The adoption of a research method helped to mitigate most of the internal threats.

The tool support was important to reduce errors from manual activities by generating small artifacts to facilitate the analysis.

This study was conducted by one researcher with the same role as a developer. There was strict control to validate the results through software tools to mitigate actions that could harm the validity of the study. Despite such effort, more experiments need to be conducted in a controlled environment to correct any bug that could have been introduced to validate the concept completely.

8.5 Future research

This research explored only one part of the study of ontologies using natural language processing. NLP techniques and the use of ontologies should be explored to provide a better understanding of the real world to a machine through computational programs.

Further use of convolutional neural networks is one of them that can be highly helpfully to reach this goal. It can be a good area of opportunity to identify what is the context of a sentence in a context domain to classify it better. Another area is to classify each noun in a sentence taking the whole domain and identify the right context. However, this may require some computational resources or parallel techniques.

References

Al Omran, F. N. A., & Treude, C. (2017). Choosing an nlp library for analyzing software documentation: a systematic literature review and a series of experiments. In *Proceedings of the 14th international conference on mining software repositories* (pp. 187–197).

Bass, L., Clements, P., & Kazman, R. (2013). Architectural tactics and patterns. *Software Architecture in Practice*, 214.

Berland, M., & Charniak, E. (1999). Finding parts in very large corpora. In *Proceedings of the 37th annual meeting of the association for computational linguistics on computational linguistics* (pp. 57–64).

Bernstein, A., & Kaufmann, E. (2006). Gino–a guided input natural language ontology editor. In *International semantic web conference* (pp. 144–157).

Blessinga, R. (2018). *Designing the automated greenhouse-matching requirements and architecture for startup product specification using epic stories* (Unpublished master's thesis).

Buitelaar, P., Cimiano, P., Haase, P., & Sintek, M. (2009). Towards linguistically grounded ontologies. In *European semantic web conference* (pp. 111–125).

Cambria, E., & White, B. (2014). Jumping nlp curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, *9*(2), 48–57.

Chowdhury, G. G. (2003). Natural language processing. *Annual review of information science and technology*, *37*(1), 51–89.

Churchland, P. S., & Sejnowski, T. J. (1990). Neural representation and neural computation. *Philosophical Perspectives*, *4*, 343–382.

Cleland-Huang, J., Gotel, O. C., Huffman Hayes, J., Mäder, P., & Zisman, A. (2014). In *Software traceability: trends and future directions* (pp. 55–69).

Crystal, D. (2011). *A dictionary of linguistics and phonetics* (Vol. 30). John Wiley & Sons.

Da Silva, A. R. (2015). Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, *43*, 139–155.

Erdmann, M., Maedche, A., Schnurr, H.-P., & Staab, S. (2000). From manual to semi-automatic semantic annotation: About ontology-based text annotation tools. In *Proceedings of the coling-2000 workshop on semantic annotation and intelligent content* (pp. 79–85).

Gabrilovich, . M. S., E. (2009). Wikipedia-based semantic interpretation for natural language processing. *Artificial Intelligence Research*, *34*, 443–498.

Gotel, O. C., & Finkelstein, C. (1994). An analysis of the requirements traceability problem. In *Proceedings of ieee international conference on requirements engineering* (pp. 94–101).

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, *43*(5-6), 907–928.

Haas, S. W. (1996). Natural language processing: toward large-scale, robust systems. *Annual review of information science and technology (ARIST)*, *31*, 83–119.

Happel, H.-J., & Seedorf, S. (2006). Applications of ontologies in software engineering. In *Proc. of workshop on sematic web enabled software engineering"(swese) on the iswc* (pp. 5–9).

Hirschberg, J., & Manning, C. D. (2015). Advances in natural language processing. *Science*, *349*(6245), 261–266.

Kaiya, H., & Saeki, M. (2006). Using domain ontology as domain knowledge for requirements elicitation. in requirements engineering. *14th IEEE International Conference*, 189–198.

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kuhn, R., & De Mori, R. (1995). The application of semantic classification trees to natural language understanding. *IEEE transactions on pattern analysis and machine intelligence*, *17*(5), 449–460.

Lapedes, A., & Farber, R. (1987). *Nonlinear signal processing using neural networks: Prediction and system modelling* (Tech. Rep.).

Lapedes, A. S., & Farber, R. M. (1988). How neural nets work. In *Neural information processing systems* (pp. 442–456).

Lawrence, S., Giles, C. L., Tsoi, A. C., & Back, A. D. (1997). Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, *8*(1), 98–113.

Leech, G. (2014). The state of the art in corpus linguistics. In *English corpus linguistics*

(pp. 20–41). Routledge.

Liu, K., Hogan, W. R., & Crowley, R. S. (2011). Natural language processing methods and systems for biomedical ontology learning. *Journal of biomedical informatics*, *44*(1), 163–179.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., & Brinkkemper, S. (2015). Forging high-quality user stories: towards a discipline for agile requirements. *Requirements Engineering Conference*, 126–135.

Maedche, A., & Staab, S. (2001). Ontology learning for the semantic web. *IEEE Intelligent systems*, *16*(2), 72–79.

Mani, I. (1999). *Advances in automatic text summarization*. MIT press.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., & McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations* (pp. 55–60).

Martens, A. (2018). *Ontological traceability for software* (Unpublished master's thesis). Utrecht University.

Maynard, D., Li, Y., & Peters., W. (2008). Ontology learning and population: Bridging the gap between text and knowledge. In (chap. 4). IOS Press.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Muter, L., Deoskar, T., Mathijssen1, M., Brinkkemper, S., & Dalpiaz, F. (2019). *Refinement of user stories into backlog items: Linguistic structure and action verbs.* (Accepted for publcation)

Novichkova, S., Egorov, S., & Daraselia, N. (2003, Sep). Medscan, a natural language processing engine for medline abstracts. *Bioinformatics (Oxford, England)*, *19*(13), 1699–706.

Pustejovsky, J., & Stubbs, A. (2012). *Natural language annotation for machine learning*. " O'Reilly Media, Inc.".

Quirchmayr, T., Paech, B., Kohl, R., Karey, H., & Kasdepke, G. (2018). Semi-automatic rule-based domain terminology and software feature-relevant information extraction from natural language user manuals. *Empirical Software Engineering*, 1–54.

Rindflesch, T. C., & Fiszman., M. (2003). The interaction of domain knowledge and linguistic structure in natural language processing: interpreting hypernymic propositions in biomedical text. *Biomedical informatics*, *36*(6), 462–477.

Robeer, M., Lucassen, G., van der Werf, E., J. M., Dalpiaz, F., & Brinkkemper, S. (2016). Automated extraction of conceptual models from user stories via nlp.

2016 IEEE 24th International Requirements Engineering Conference (RE), 196–205.

Rolland, C., & Proix, C. (1992). A natural language approach for requirements engineering. In *International conference on advanced information systems engineering* (pp. 257–277).

Rong, X. (2014). word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

Simard, P. Y., Steinkraus, D., Platt, J. C., et al. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Icdar* (Vol. 3).

Smeaton, A. F. (1999). Using nlp or nlp resources for information retrieval tasks. In *Natural language information retrieval* (pp. 99–111). Springer.

Soricut, R., & Marcu, D. (2003). Sentence level discourse parsing using syntactic and lexical information. In *Proceedings of the 2003 conference of the north american chapter of the association for computational linguistics on human language technology-volume 1* (pp. 149–156).

Uschold, M., & Gruninger, M. (1996). Ontologies: Principles, methods and applications. *The knowledge engineering review*, *11*(2), 93–136.

van de Weerd, I., & Brinkkemper, S. (2009). Meta-modeling for situational analysis and design methods. In (pp. 35–54). IGI Global.

Warner, A. J. (1987). Natural language processing. In *Annual review of information science and technology, vol. 22* (pp. 79–108).

Wieringa, R. J. (2014). *Design science methodology for information systems and software engineering*. Springer.

Williams, S. (2013). *An analysis of pos tag patterns in ontology identifiers and labels* (Tech. Rep.). Technical report, Technical Report TR2013/02, Department of Computing, The ....

Winkler, S., & von Pilgrim, J. (2010). A survey of traceability in requirements engineering and model-driven development. *Software and Systems Modeling*, *9*(4), 529–565.

Zhang, Y., Witte, R., Rilling, J., & Haarslev, V. (2008). Ontological approach for the semantic recovery of traceability links between software artefacts. *IET Software*, *2*(8), 185–203.

Appendix A

User Stories

Table A.1: Sub-ontology User Stories.

US ID	User Story
Y-0001	As a geo researcher I want to store my HPTlab data in YoDa So I can annotate my HPT lab data according none analog modelling subdomain
Y-0002	As a geo researcher I want to annotate my TEClab data in YoDa according to the analog modelling subdomain definition
Y-0003	As a researcher I want to fill data request including a research proposal so that I can submit it as a whole
Y-0004	As a researcher I want firstname/lastname split in metadata schema
Y-0005	As a GEO user I want my data harvested via OAI-PMH with EPOS GFZ-ISO schema
Y-0006	As a data manager I can approve a data request so that the distribution process can start
Y-0007	As a board member I can view research proposal evaluations submitted by DMC members
Y-0008	as a researcher i can opt to receive email notifications on submit
Y-0009	As a DMC member I can create and submit a review of a research proposal and data request
Y-0010	as a Ron Researcher i can download a folder and its content so that I have an easy way of incidentally work with files
Y-0011	As a researcher I want help to lookup my personal PID so that I can add it to metedata
Y-0012	As an admin I want ichk to display the actual checksums upon mismatch so that I can analyze the cause
Y-0013	As a user I want to see a bounding box on the landingpage so that I can easily see the location of the data
Y-0014	As a researcher i want to search in a better way
Y-0015	As a datamanager I want my metadata schema compliant with DANS
Y-0016	As a researcher I want to download system metadata with the published data package

ID	User Story
Y-0017	As a researcher I want my metadata to be included in DANS export
Y-0018	As a researcher I want to enter metadata bounding box coordinates using maps so I can useEPOS schema
Y-0019	As as user I want to access Yoda via my own domain name so that I have a branded application
Y-0020	As a researcher I want to depublish an archived data package
Y-0021	As a datamanager I want all persistent identifiers of a package updated after archiving
Y-0022	As a datamanager I want to archive a published package at DANS
Y-0023	As an iRODS admin I want to indefinately keep info on crucial events so that I have a provenance log
Y-0024	As a datamanager I want to know that a group within my category has not been active for 3 months so that I can detect inactive groups
Y-0025	As a Yoda admin I want to have an overview of all user autorisation changes so that I can analyse incidents
Y-0026	As a datamanager I want to know to which groups a user belongs so that I can manage my community
Y-0027	As a researcher I want to specify my data package type so that I can have types other than dataset
Y-0028	As a user I want to reference a data object in my package as the value of a metadata field so that i can describe the function of an object in my data package
Y-0029	As a Yoda admin I want: Refactor Intake module GRP groups to INTAKE groups
Y-0030	As a GEO reseacher I want to have EPOS metadata schema so that I can comply with European metadata standard for EPOS
Y-0031	As a datamanager I want to have a search group feature in the groupmanager
Y-0032	As a data receiver I want to be notified of data being shared with me so that I know I can download files
Y-0033	As a data sender I want to share my distribution job
Y-0034	As a data receiver I want login and have a list of jobs shared with me
Y-0035	As a data sender I want to add data to my distribution job
Y-0036	As a data sender I want to view my distribution jobs and their details so that I am informed about my distribution job
Y-0037	As a data sender I want to create a data distribution job
Y-0038	As a data sender I want to login the data distribution service
Y-0039	As a researcher I want to locate a data package based on its EPIC PID so that I can find vaulted data packages

Remaining user stories available under request of disclosure and with system owner agreement.

Appendix B

Survey

Table B.1: Sub-ontology User Stories.

Factor	Question	
		1 2 3 4 5
Ease of use	I find the idea very complex and difficult to understand. Overall, I found the use of ontologies very difficult to use I completely understood the purpose of the tool.	
ine Usability	I believe that tools like that reduces the time when developing software products. I believe that it will be very difficult to apply a similar tool in a software development environment Overall, I found concept of the tool very useful. Overall, I found the concept of the tool does not provide an effective solution to traceability in software development. I believe that using tools like that will help to reduce effort to identify development objects when a business process is changed by any reason.	
Intention to use	I will definitely would try to integrate a tool like that in software development. I will definitely use a tool to generate ontologies and sub-ontologies of the requirements. I will not use any tool like that at all in the future.	