



HAL
open science

Bridging the gap between natural language system requirements and architecture design models

Takwa Kochbati

► **To cite this version:**

Takwa Kochbati. Bridging the gap between natural language system requirements and architecture design models. Software Engineering [cs.SE]. Université Paris-Saclay, 2021. English. NNT : 2021UP-ASG076 . tel-03411393

HAL Id: tel-03411393

<https://theses.hal.science/tel-03411393>

Submitted on 2 Nov 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Bridging the gap between natural
language system requirements and
architecture design models

*Comblant le fossé entre les exigences du système exprimées en
langage naturel et les modèles de conception d'architecture*

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°580, sciences et technologies de l'information et de la
communication (STIC)

Spécialité de doctorat : Informatique

Unité de recherche : Université Paris-Saclay, CEA, Institut LIST, 91191, Gif-sur-Yvette, France

Référent : Faculté des sciences d'Orsay

**Thèse présentée et soutenue à Paris-Saclay,
le 19/10/2021, par**

Takwa KOCHBATI

Composition du Jury

Aurélié NEVEOL

Directrice de Recherche, Université Paris-Saclay, CNRS, LISN

Présidente

Ileana OBER

Professeur, Université Paul Sabatier

Rapportrice & Examinatrice

Reda BENDRAOU

Professeur, Université Paris Ouest Nanterre la Défense

Rapporteur & Examineur

Xavier LE PALLEC

Maître de conférences HDR, Université de Lille

Examineur

Direction de la thèse

Sébastien GERARD

Directeur de recherche HDR, CEA LIST

Directeur de thèse

Shuai LI

Ingénieur chercheur, CEA LIST

Co-Encadrant

*To my family, the reason of what I become today.
Thanks for your great support and continuous love and care.*

Acknowledgements

First of all, I would like to express my sincere gratitude to my thesis advisor Sébastien Gérard and my scientific supervisor Shuai Li for their availability, their attentive guidance and their involvement in this thesis.

I would also like to thank Chokri Mraidha for his valuable and constructive suggestions during the planning and development of my research.

I want to extend my thanks to Xavier Le Pallec and Pierre Zweigenbaum for their valuable advice during the mid-term defense.

Last but not least, the most important thanks go to my family, who have always been there for me: my parents Samia and Habib for their selfless support and love, my brother Aymen and my sisters Abir, Narjess and Sameh, with whom sharing life has always been the best thing in the world.

Contents

1 Introduction	8
1.1 AI for automated SE	8
1.2 Automation of MBSE with AI - potential issues	9
1.3 Goal of the thesis	10
1.4 Thesis structure	12
2 Background	14
2.1 Requirements engineering	14
2.1.1 System requirements	14
2.1.2 Requirements engineering process	15
2.2 AI techniques for SE	17
2.2.1 Natural Language Processing	17
2.2.2 Text clustering	24
2.3 Conclusion	27
3 State of the art	28
3.1 Current research directions in deriving architecture models from natural language software requirements	28
3.2 The machine learning techniques used to group natural language software requirements	30
3.2.1 Related works of requirements clustering	31
3.2.2 Related works of requirements classification	32
3.2.3 Summary	35
3.3 From natural language requirements to visual models	36
3.3.1 Related works	37
3.4 Summary	41
3.5 Research questions	44

3.6	Scope of contributions	46
3.7	Contribution Overview	47
3.8	Conclusion	49
4	Semantic Clustering of System Requirements	50
4.1	The semantic clustering solution overview	50
4.2	Preprocessing	51
4.2.1	Cleaning	52
4.2.2	Tokenization	53
4.2.3	Annotation	53
4.2.4	Normalization	54
4.3	Semantic similarity computation module	55
4.3.1	Word-level similarity computation	55
4.3.2	Requirement-level similarity computation	57
4.4	Requirements clustering	59
4.5	Labelling	60
4.6	Conclusion	61
5	Automatic generation of the preliminary architecture design models	62
5.1	Linking requirements to models	62
5.1.1	Requirements engineering challenges	63
5.1.2	Use case modeling	64
5.2	Overview of the model extractor approach	66
5.2.1	Extracting relevant use case model elements	67
5.2.2	Mapping into preliminary UML design models	73
5.3	Conclusion	74
6	Evaluation of the requirements semantic clustering solution	76
6.1	Key Performance Indicators (KPIs)	76
6.2	Case studies description	78
6.2.1	User stories	78
6.2.2	Functional requirements written in plain text	80
6.3	Results analysis and evaluation	81
6.3.1	Experimental settings	81
6.3.2	Evaluation of the semantic clustering of the user story case studies	82

6.3.3 Evaluation of the semantic clustering of the case studies of functional requirements written in plain text	86
6.4 Assessing KPIs	91
6.5 Threats to validity	94
6.6 Conclusion	95
7 Evaluation of the automatic generation of the preliminary architecture design models	96
7.1 Key Performance Indicators (KPIs)	96
7.2 Results analysis and evaluation	98
7.2.1 Relevant model elements extraction	98
7.3 Assessing KPIs	106
7.4 Threats to validity	107
7.5 Conclusion	108
8 Conclusion and future work	109
A Résumé	113
A.1 L'intelligence artificielle pour automatiser l'ingénierie logicielle	113
A.2 Automatisation du MBSE en utilisant l'IA - les problèmes potentiels	115
A.3 Objectif de la thèse	116
A.4 Contributions	118
A.5 Validation	121
B Technical demonstration of the developed tool	123
B.1 The semantic clustering approach	123
B.1.1 Software and libraries	123
B.1.2 Execution	124
B.2 Automatic generation of the UML use case model	126
B.2.1 Software and libraries	126
B.2.2 Execution	126

List of Figures

2.1	Vector space for a small sample corpus	19
2.2	Two model architectures of word2vec. The CBOW model is to predict the current word based on the words around it, and the Skip-gram model can find the most likely surrounding words based on the current word.	23
2.3	An example of graphical representation of a dendrogram tree	27
3.1	General process for applying AI techniques to bridge the gap between natural language requirements and design models	29
3.2	Overview of the proposed approach	48
4.1	The requirement semantic clustering approach	52
4.2	An example of a tokenized sentence.	53
4.3	An example of a parsed sentence with POS tags.	54
4.4	An example of tokens after conducting stemming.	54
4.5	Simplified process of obtaining word vectors using word2vec.	56
4.6	An example for computing $\max Sim(w, R)$	58
4.7	A simplified example of clusters identification.	60
5.1	Use case model and design model.	65
5.2	Overview of the model extractor approach.	66
5.3	Extracting relevant use case model elements.	68
5.4	Example of the result of the dependency parsing using <i>Spacy</i> .	69
5.5	Example of the structure of the resulted CSV file grouping the use case model elements.	72
5.6	Pseudo-code of the mapping algorithm.	74
6.1	Bar graph of the clustering accuracy across user story case studies.	83
6.2	Dunn index bar graph of the "CMS Company" case study.	85

6.3 Dendrogram of the identified clusters of the "CMS Company" case study.	86
6.4 Bar graph of the precision values of the identified clusters.	89
6.5 Bar graph of the recall values of the identified clusters.	90
6.6 Bar graph of the F-measure values of the identified clusters.	90
6.7 The number of the identified clusters against the number of the requirements for each case study of functional requirements.	91
6.8 Execution time by number of user stories.	93
6.9 Execution time by number of functional requirements written in plain text.	94
7.1 Example of extracted elements for the "CMS Company" case study.	99
7.2 Example of extracted elements for the "Web Company" case study.	99
7.3 Example of extracted elements for the "Archive Space" case study.	100
7.4 Example of extracted elements for the "E-store System" case study.	100
7.5 Example of extracted elements for the "WASP System" case study.	101
7.6 Example of extracted elements for the "UIIS System" case study.	101
7.7 Example of extracted elements for the "MHC-PM System" case study.	101
7.8 The generated UML use case model for the CMS Company.	102
7.9 Example of a mapping of a cluster into a use case package for the CMS Company case study.	103
7.10 Bar graph of F-measure values of the extracted model elements.	104
A.1 Un aperçu de l'approche proposée	119
B.1 The generation of semantic clusters of requirements.	124
B.2 The use case model elements extraction.	125
B.3 The UML use case model generation.	127

List of Tables

2.1 term-by-document matrix M for a small sample corpus	19
3.1 Overview of the existing approaches for clustering of natural language software requirements	33
3.2 Overview of the existing approaches for classifying natural language software requirements	35
3.3 Overview of the existing approaches for extracting design models from natural language software requirements	40
3.4 Summary table of recent work in automating the transition from natural language requirements and architecture design models	42
5.1 Model elements mapping	73
6.1 Characteristics of the case studies of user stories	79
6.2 Characteristics of the case studies of functional requirements written in plain text	80
6.3 An example of a semantic cluster identified from the "CMS Company" case study.	83
6.4 An example of a semantic cluster identified from the "Web Company" case study.	84
6.5 An example of a semantic cluster identified from the "Archive Space" case study.	85
6.6 Accuracy of the generated semantic clusters for the user story case studies.	85
6.7 Identifying the C_Gap for the user story case studies.	86
6.8 An example of a semantic cluster identified from the "E-store system" case study.	87
6.9 An example of a semantic cluster identified from the "WASP system" case study.	87
6.10 An example of a semantic cluster identified from the "UUIS system" case study.	88
6.11 An example of a semantic cluster identified from the "MHC-PM system" case study.	88
6.12 Accuracy of the generated semantic clusters for the case studies of functional requirements written in plain text.	89
6.13 The execution time of the clustering approach for the user story case studies	92

6.14 The execution time of the clustering approach for the case studies of functional requirements written in plain text	93
7.1 Accuracy of the generated UML use case model for the "Web Company" case study. .	104
7.2 Accuracy of the generated UML use case model for the "CMS Company" case study. .	104
7.3 Accuracy of the generated UML use case model for the "Archive Space" case study. .	105
7.4 Accuracy of the generated UML use case model for the "E-store System" case study. .	105
7.5 Accuracy of the generated UML use case model for the "WASP System" case study. .	105
7.6 Accuracy of the generated UML use case model for the "UUIS System" case study. . .	105
7.7 Accuracy of the generated UML use case model for the "MHC-PM System" case study. .	105
7.8 The execution time of the automatic models generation approach for the user story case studies.	107
7.9 The execution time of the automatic models generation approach for the case studies of functional requirements.	107

Chapter 1

Introduction

This Chapter presents an introduction to this thesis, in which we give an overview of the topics it deals with. First, we present the general context of this thesis that consists in using Artificial Intelligence (AI) for Software Engineering (SE). Then, we give the potential issues that we have identified for the automation of Model Based Systems Engineering (MBSE) with AI. Afterwards, we present the goal of the thesis and finally, we present the structure of this thesis document.

1.1 AI for automated SE

The software engineering (SE) industry is always looking for better and efficient ways of building higher quality software systems.

Model-Based Systems Engineering (MBSE), as defined by the International Council on Systems Engineering (INCOSE) [112], is *“the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases”*. MBSE comprises multiple modeling concepts: processes, languages, methods, and tools to produce one system model or more. Hence, it promises to support SE companies by enabling the realization of successful systems by fostering a holistic view of design and empowering high quality and maintainable software architecture.

Despite its theoretical advantages, several studies demonstrated that MBSE methodology remains difficult to apply [87, 10]. Consequently, MBSE is not yet widely adopted in real-world applications since it still struggles with huge challenges [5, 27, 28] that neither the MB nor the SE part is able to handle. We can summarize all the challenges related to the adoption of MBSE in one: its benefits do not outweigh its costs [24].

Meanwhile, recent technological advancements regarding big data management, development of more complex systems, algorithms, and tools have enabled a lot of opportunities for industries to make use of Artificial Intelligence (AI). The discipline of AI has been generally recognized for more than seven decades. It can be described as the the science of mimicking human mental faculties on a computer [54]. AI systems include modules that enables the generation of types of learning. For instance, Machine Learning (ML) is a type of artificial intelligence technique that makes decisions or predictions based on data. Natural Language Processing (NLP) is the branch of AI which enables computers to understand, interpret, and manipulate human languages. In fact, AI techniques together with suitable technology have enabled systems to perceive, predict, and act in assisting humans in a wide range of applications.

In 2016, AI had a major breakthrough when a computing system developed by Google DeepMind's researchers, AlphaGo, beat the world's best human player in Go, a game far more complicated than chess. During the same year, Microsoft corporation launched its AI chatbot Tay to better understand the way teenagers talk via Twitter. Tay aimed at learning how to speak better over time through conversations, however, it was shut down only 16 hours after its launch when it began to post inflammatory and offensive tweets. Though these two examples cannot be compared due to their different domains of application, it is crucial to acknowledge both advantages and drawbacks of the application of AI and to keep it under control.

Although the disciplines of AI and SE have developed separately, they have many commonalities. Actually, both disciplines deals with modeling real world objects from the real world like business process, expert knowledge, or process models [94].

Nowadays several research directions of both AI and SE disciplines come closer together and are beginning to build new research areas, automated software engineering being one of them. Automated software engineering is a research area which is constantly developing new methodologies and technologies in order to create software systems that exhibit some form of human intelligence. It aims to assist or automate the activities in software engineering in order to improve the efficiency, reduce time and costs of the system development process.

1.2 Automation of MBSE with AI - potential issues

In recent years, system design constraints evolves more and more requiring to embed more stakeholders in the projects to handle various new concerns - such as security, safety, cost, and sustainability – earlier in the process, at specification time. Consequently, modern software projects are becoming many times larger and hence more complex than in the past. Especially, the exponential

growth of the number of system requirements raises difficulties in managing manually the requirements and having a clear crystal view of the expectation and scope of the system to be designed [22].

Model-based development methods ensure the formalized application of modeling for system specification, instead of doing it just using informal text or drawings description. However, architecture design models are always extracted manually by engineers which became a tedious, time-consuming and error prone task especially with the exponential growth of the system requirements and the needs to trace everything all along the life-cycle of the product to be designed. Thus, this task is critical since errors introduced at the beginning stages of development are the hardest and most expensive to correct [23].

Moreover, the growing demand on agility and the development of more complex systems have led practitioners to focus on programming rather than requirements management, planning, specification, architecture, design and documentation [80]. Consequently, adopting model-based methods to develop complex systems becomes challenging as it requires much time, cost, and resources investment [72]. In fact, the ROI (Return On Investments) deploying MBSE is indeed more perceived on the long term than on the short term [13]. The lack of human expertise as well as powerful automation tools are often cited as the main key barriers that still slow down the spread of the MBSE approach and present significant hurdles to demonstrate its ROI. Hence, it stands to reason that advances in AI can bring great practical value to mitigate some of the challenges raised by the adoption of MBSE.

In the 1990s, Ian [106] asked the question: “*Why was AI never mentioned in a seminar discussion about the future of software engineering?*”. After almost 30 years, and looking in the SE Handbook [112] and the recent inspiring SE Vision 2025 [57], it is also surprising that the application of AI for SE is not clearly mentioned. We therefore ask ourselves, ***instead of ignoring many well-known MBSE methods, why not automate the design of systems starting from early requirements with the help of some intelligent solutions?***

Our research encompasses the integration of the fields of MBSE and AI -especially ML and NLP techniques- to automate the generation of architecture design of complex systems and thus, improve the efficiency, reduce time and costs of the development process.

1.3 Goal of the thesis

In the first steps of the software development process, engineers discover and collect requirements from customers and then, they manually record them in a requirements specification document.

The gathered requirements describe different aspects of the target software in an informal natural language. Requirements elicitation and management has a significant impact on information systems quality and success, as the errors introduced at the beginning stages of development are the hardest and most expensive to correct [23]. Industry figures state that insufficient requirement engineering is the root cause of the failure of more than 50% of software projects [70]. Hence, it is crucial that the requirements set be well understood and well managed by engineers [118].

In recent years, system design constraints evolve more and more requiring to embed more stakeholders in the projects to handle various new concerns - such as security, safety, cost, and sustainability – earlier in the process, at specification time. Consequently, the sets of requirements used in the analysis and design of such systems are often so large that traditional requirements management and organisation techniques become unwieldy. Moreover, modern software projects are becoming many times larger and more complex than in the past.

Indeed, many of the classic challenges of developing software products derive from this essential complexity and its nonlinear increases with size [22]. Especially, the exponential growth of the number of requirements raises difficulties in manually managing requirements and having a clear crystal view of the expectation and scope of the system to be designed. Although MBSE methods remain to be the focal point that ensure the transition from natural language requirements to architecture design, its adoption still faces significant hurdles to demonstrate its ROI, especially with the increasing complexity of the developed systems.

In this thesis, we aim to contribute a first step towards applying AI for MBSE to optimize the adoption of MBSE and resolve a set of its challenges. Specifically, the main objective of this thesis is the following:

Define a new flow of AI components including their specific parametrization, enabling the automation of the transition from natural language requirements to the architecture design of complex systems.

This main goal hides two underlying research directions that we investigated in our proposal:

The first research direction consists in solving the challenges raised by the exponential growth of requirements and the increasing complexity of systems. One of the most used and efficient design paradigms to deal with complexity is the well-known “divide-to-conquer” strategy i.e., building smallest pieces to reduce the complexity.

In order to meet this goal, we propose a solution based on ML and NLP techniques that decomposes the system into a set of sub-systems based on the semantic similarity of early requirements.

In fact, such early decomposition helps developers to better understand and realize the target software project. In addition, it helps to design an initial architecture described by a package break-down model of the software product as the identified groups of requirements represent components or sub-systems that should be implemented and reused [107, 6].

The second research direction consists in proposing an automatic model generator that provides a semi-formal representation of the initial architecture design of the system based on the obtained groups of requirements.

In order to achieve this goal, further analyses of the requirements within each identified group of requirements by means of NLP techniques are required in order to extract the relevant elements that are needed to build the preliminary design model. As a result, we obtain an initial architecture design model that consists of a preliminary package break-down model of the target system. Indeed, adopting an initial package break-down model helps to represent and communicate what is important among stakeholders and developers, gives insights on the expected features of the target system and helps to keep track of the gathered requirement throughout the project.

1.4 Thesis structure

This thesis is structured in four parts and several appendices. The contents of the rest of this thesis is organized as follows:

Part 1, made of chapters 2 and 3 presents the background encompassing basic concepts needed to be known in advance and gives an overview of the related work and studies that make use of AI techniques to bridge the gap between natural language system requirements and the preliminary architectural design.

Part 2 is made of chapter 4 and 5 and presents the contributions of the thesis. Chapter 4 describes the approach that we have proposed to support the decomposition of complex systems based on early system requirements, whereas Chapter 5 presents the approach that we have proposed to automatically generate preliminary architecture design models describing the decomposition of the system.

Part 3 is made of chapter 6 and 7 and presents the prototyping and the validation of the contributions of this thesis. Chapter 6 presents the prototyping and the validation of the approach that has been proposed to decompose the complex systems based on early requirements. Chapter 7 presents the prototyping and the validation of the approach that has proposed to automatically generate preliminary architecture design models describing the decomposition of the system. We

evaluated our contributions with several case studies and we compared our results with baselines from related works.

Chapter 8 concludes our thesis work by summarizing our contributions and analyzing the attainment of objectives and discussing the potential directions for future research.

Finally, the Appendices extend and clarify information to give a better understanding of some of the issues presented in previous chapters.

Chapter 2

Background

In this chapter, we discuss and clarify some key concepts that are needed to be known of in advance. Firstly, we present the basic information related to requirements engineering. Afterwards, we present the fundamental concepts regarding AI techniques used in SE and particularly AI techniques used to support the transition from system requirements to design models.

2.1 Requirements engineering

Our research scopes the early stages of software development particularly the transition from system requirements to architecture design models. In what follows, we detail the basic concepts related to requirements engineering.

2.1.1 System requirements

Requirements engineering is the formalised process of defining the features and constraints of a system to be developed. These features and constraints are known as the "requirements" of the system. More formally, a requirement is a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard or specification [1]. System requirements are mainly divided into functional and non-functional requirements.

Functional requirements describe what the system does or must not do. Essentially, they help to capture the intended behavior of the system. This behavior may be expressed as functions, services or tasks or which system is required to perform. Functional requirements are supported by non-functional requirements (also known as "quality requirements"), which impose constraints on the design or implementation (such as performance requirements, security, or reliability). Mainly, func-

tional requirements are expressed in the form "system must do «requirement»," while non-functional requirements take the form "system shall be «requirement»." [31]

Non-functional requirements specify how a system is supposed to be. They describe aspects of the system that are not directly related to the function behavior of the system [23]. Non-functional requirements are concerned with emergent properties, for instance: reliability, performances or reparability etc [111]. These are constraints and boundaries which are essential to be acknowledged in software development. Generally, non-functional requirements are in the form of "system shall be «requirement»".

2.1.2 Requirements engineering process

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system.

The activities involved in requirements engineering vary widely, depending on the type of system being developed and the organization's specific practice(s) involved [107]. These may include:

- **Requirement elicitation:**

It is also known as the gathering of requirements. In this step, developers and stakeholders meet; the latter are inquired concerning their needs and wants regarding the software product. Requirements elicitation is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of same type, standards and other stakeholders of the project.

The techniques used for requirements elicitation include interviews, brainstorming, task analysis, prototyping, etc. Elicitation does not produce formal models of the requirements understood. Instead, it widens the domain knowledge of the analyst and thus helps in providing input to the next stage.

- **Requirement analysis:**

In this step, the requirements are analyzed to identify inconsistencies, defects, omission, etc. Requirements are identified (including new ones if the development is iterative), and conflicts with stakeholders are solved. Both written (e.g., use cases and user stories) and graphical tools (e.g., UML) are successfully used as aids. Although graphical tools are commonly used in the

design phase, some find them helpful at this stage too.

- **Requirement specification:**

At this stage, requirements are documented in a formal artifact called a Software Requirements Specification (SRS) document, which will become official only after validation. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team. A requirement specification (RS) can contain both written and graphical (models) information if necessary.

- **Requirement validation:**

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be checked against the following conditions. Hence, requirement validation consists in checking that the documented requirements and models are consistent and meet the stakeholder's needs. Only if the final draft passes the validation process, the RS becomes official.

- **Requirement management:**

Requirement management is the process of managing changing requirements during the requirements engineering process and system development. New requirements emerge during the process as business needs a change, and a better understanding of the system is developed. The priority of requirements from different viewpoints may change or be extended during development process. Moreover, the business and technical environment of the system changes during the development.

MBSE methods promise to support both the requirements and the design associated with the development of complex systems. However, the transition from system requirements to design model is usually done manually i.e., there is a huge lack of automation and assistance in transforming the information and knowledge from documents into models. Hence, we conclude that MBSE still requires a high investment effort compared to its achieved benefits, particularly in the scope of the transition from natural language system requirements to design models. This is mainly due to several human and technological factors [49].

As further shown by this thesis, AI techniques can help mitigate such MBSE challenges, because they are designed to deal with one of the most demanding challenges of all; the replication of intelligent behaviour. In the next section, we present some AI techniques that have been used to support

the automation and the assistance of the transition from natural language system requirements to design models.

2.2 AI techniques for SE

The software engineering community has exploited many of the practical algorithms, methods and techniques that have emerged from the AI community. These AI algorithms and techniques find important and effective applications that impact on almost every area of software engineering activity. Automated software engineering is a research area in which both AI and SE disciplines intersect.

In particular, the automation of the transition from natural language requirements to architecture design has gained a lot of attention in the last few years. In what follows, we give insights on the ML and NLP techniques that have been used to support such transition.

2.2.1 Natural Language Processing

Natural Language Processing is defined as follows by Chowdhury [29]: *"Natural Language Processing (NLP) is an area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things. NLP researchers aim to gather knowledge on how human beings understand and use language so that appropriate tools and techniques can be developed to make computer systems understand and manipulate natural languages to perform the desired tasks."*

Natural Language Processing (NLP) is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language. By using NLP techniques, computers can analyze, understand, and derive meaning from human language. There are many applications for NLP that can help engineers in the software development process such as automatic summarization, translation, or knowledge extraction.

As a technology, natural language processing has come of age over the past ten years, with products such as Siri, Alexa and Google's voice search employing NLP to understand and respond to user requests. Hence, NLP is crucial for extracting architecture design models from requirements written in natural language. The basic ideas and techniques of NLP tasks used in this thesis are introduced in the following.

Semantic similarity matching methods

Computing sentence similarity is not a trivial task, due to the variability of natural language expressions. Computing semantic similarity between two sentences or texts aims to check if two pieces of text mean the same thing or how semantically similar they are [65]. Many approaches have been proposed for semantic similarity computation by the employment of lexical matching, linguistic analysis and semantic features. Methods for lexical matching are based on the intersection of the word sets of the input texts and they aim to determine whether the words in two texts have similar spellings. Although they are successful in trivial cases, these lexical methods cannot always identify the semantic similarity of texts [83, 61, 58]. For example, the "US" would be closer to the "UK" this way, than it would be to the "States". Features based on linguistic analysis, like dependency parses or syntactic trees, are mainly used for text similarity [50]. Although most languages have linguistic tools such as parsers, their quality varies significantly across languages. Moreover, these tools might require manual intervention even though external tools such as Stanford Parser [79] are integrated. They also might be expensive to compute at run-time when it comes to high-quality parses. For semantic features, external knowledge sources such as Wikipedia [14] or WordNet [14, 43] have been used. The shortcoming of applying WordNet or other external lexical database of structured semantic knowledge is that high quality resources like these are not available for all languages, and proper names, domain-specific technical terms and slang tend to be underrepresented [4].

Word embedding

Word vectors—also referred to as Word embeddings—has recently seen an increasing interest as new ways of computing them efficiently have become available. Word embeddings models help to capture the context of a word in a document, semantic and syntactic similarity, as well as its relations with other words [71]. The vector representations of words are commonly achieved in two different ways: traditional distributional semantic models (DSMs) and neural word embedding.

- **Traditional DSMs**

Distributional semantic methods such as *Vector Space Model* (VSM) and *Latent Semantic Analysis* (LSA) [34] are based on the intuition that words appearing in similar contexts tend to have similar meanings.

-VSM: The VSM is a model for representing text in a vector space based on the bag of words approach. It was first presented as a model for Information Retrieval (IR) in [100] and was used in the System for the Mechanical Analysis and Retrieval of Text (SMART) information retrieval system [98, 36].

In VSM, text units of a corpus are represented by vectors. Traditionally a whole document is used as a text unit, but any other text unit like paragraphs or sentences can be used just as well. Each dimension of a vector corresponds to a term that is present in the corpus. A term might be, e.g., a single word, n-gram or a phrase. If a term occurs in a document the value of that dimension is non-zero. Values can be binary (1 if the term is present in the document and 0 if the term is not present in the document), frequencies of terms in the document, or term weights. A whole text corpus can then be represented by a term-by-document matrix M . Consider the following example of a sample text corpus containing the following three sentences:

*S1. "Small **dogs** are better than **cats**."*

*S2. "**Big cats** are nice and funny."*

*S3. "**Big dogs** are nice."*

The terms "cat", "dog" and "big" are used for indexing. The corpus can then be represented by the term-by-document matrix M as shown in Table 2.1.

Table 2.1: term-by-document matrix M for a small sample corpus

	S1	S2	S3
cat	1	1	0
dog	1	0	1
big	0	1	1

The sentences can be then represented as vectors in space as shown in Figure 2.1. Hence, similarities between documents or a query and a document can be calculated.

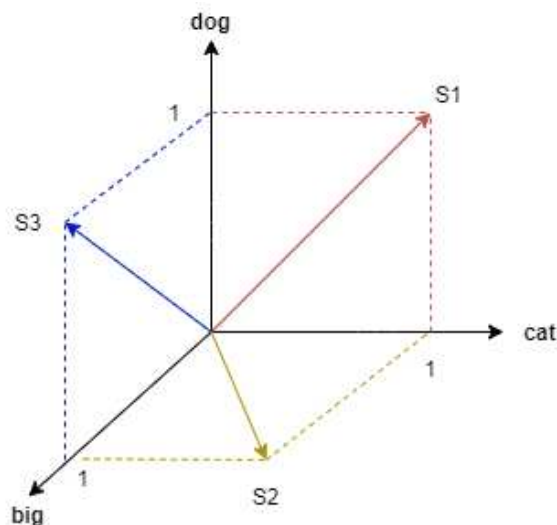


Figure 2.1: Vector space for a small sample corpus

Regarding terms weighting techniques, the most common one is the *Term Frequency-Inverse Document Frequency* (TF-IDF) weighting scheme [99]. TF-IDF is a weighting scheme that is often used in VSM together with cosine similarity to determine the similarity between two documents. TF-IDF considers the different frequency of words in all documents and is able to distinguish documents. In VSM, each vector is composed by terms and weights that represent documents. The similarity of documents can be expressed by the distance between vectors, the smaller the distance means the more similar the two documents. The formula is as follows (Equation 2.1):

$$TF - IDF = tf * idf \quad (2.1)$$

tf is the frequency of occurrence of term in the document and idf is the inverse document frequency that represents the specificity of a word. Given a word w in a corpus, $idf(w)$ has been defined as the log of the total number of documents in the corpus divided by the total number of documents including that word [62], where Equation 2.2:

$$idf(w) = \log(\text{Total number of documents} / \text{Number of documents with word } w \text{ in it}) \quad (2.2)$$

The most commonly used measure of text similarity is the cosine distance. This measure is based on the angle α between two vectors in the VSM. The closer the vectors are to each other the more similar are the documents. The calculation of the cosine of an angle between two vectors \vec{a} and \vec{b} can be derived from the Euclidean dot product as shown in Equation 2.3:

$$\text{cosine}(\alpha) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| * |\vec{b}|} \quad (2.3)$$

The values of $\text{cos}(\alpha)$ can range from -1 for opposing vectors to 1 for identical vectors.

Within the VSM only similarities between documents or between a query and documents can be calculated within one space. If terms were to be compared to each other another space would have to be considered. In a term space, where the terms represent the dimensions, the terms are considered to be linearly independent, which means their relations to each other are not taken into account. Moreover, in the traditional vector space the similarity calculation is based only on word matching. Each dimension of a vector corresponds to a term. Two documents with a similar topic but different vocabulary will not be placed next to each other.

Only documents that overlap in vocabulary will be considered similar.

-LSA: Latent Semantic Indexing (LSI) was developed as a special vector space approach to conceptual Information Retrieval (IR) [34]. It attempts to overcome two common problems of search engines – synonymy and polysemy. In the standard VSM [97], the terms are assumed to be independent and thus term associations are ignored. By contrast LSI re-expresses a co-occurrence matrix in a new coordinate system. The idea is to uncover the latent semantic structure of a document collection, i.e., to find hidden relations between terms, sentences, documents or other text units. This is achieved by using high-order co-occurrence [68]. This measure reflects the semantic similarity between words that are used in similar context, e.g., synonyms, antonyms, hyponyms or compounds. The technique is called LSI when it is applied to IR otherwise it is called LSA.

Similarly to the VSM, LSA can represent the text as document-term matrix using TF-IDF Vectorizer and calculate the similarity between documents based on their vectors. The difference is that the LSA algorithm assumes that words that will occur in similar pieces of text (the distributional hypothesis) are semantically close in meaning while VSM can not express the information of semantic level.

For example, there are documents, one contains a word “automobile”, and another contains a word “car”, these two words will be considered as two different words, have different TF-IDF weighting and they maybe influence the meaning of documents. The worst thing is the two documents is classified as two categorizations because of the two words, after all, the two words has the same meaning.

The purpose of LSA is to find true meaning of words in documents and solve the problem above-mentioned. A matrix containing word counts per document (rows represent unique words and columns represent each document) is constructed from a large piece of text and a mathematical technique called singular value decomposition (SVD) is used to reduce the number of rows while preserving the similarity structure among columns. For instance, there is 1000 documents and 8000 words, LSA will create 100-dimensional space and mapping all of words and documents to this space. The procedure of mapping documents to this space is SVD and reducing dimensions. Reducing dimensions is the most important procedure. The noise is removed and the semantic architecture becomes clear by this operation. In LSA, SVD of terms (words) by documents matrix can be formulated as follows:

$$C = USV^T, \quad (2.4)$$

where C is the term by documents matrix ($m \times n$). U is a $m \times m$ matrix and its columns is the orthogonal feature vectors of CC^T . V is a $n \times n$ matrix and its columns is the orthogonal feature vectors of C^TC . The feature values of CC^T and C^TC is same and it is $\lambda_1, \lambda_2, \dots, \lambda_n$. For S , S is a $n \times n$ matrix, $S_{ii} = \sqrt{\lambda_i}$, $\lambda_i > \lambda_{i+1}$ and zero otherwise.

To reduce the dimension of vector space to D , $S_{D+1,D+1}$ to S_{nn} are set to zero and S_{11} to S_{DD} are kept. After this, we can multiply U , S (having been reduced dimensions), V^T , and reconstruct the terms by documents matrix. In the new matrix, latent semantic of documents is presented, and we can calculate the similarity between documents or words much exactly. Each row of US is the term coordinate in latent semantic space and each row of VS is the document coordinate.

- **Neural word embedding**

Neural word embedding is a neural-network-based natural language processing architecture which can be seen as prediction models, since the vector representations of words or texts can be gained from a pre-trained language model trained on large text collections.

In [84, 85] an algorithm called word2vec is proposed. Word2vec is a two-layer neural network that is used to produce word embeddings (i.e., vectors), to obtain the word semantic similarity. The input of Word2vec is a text corpus. Given enough text data and contexts, word2vec can achieve highly accurate meanings of the words appearing in the input corpus and establish a word's association with other words. The output is a set of words representations (i.e., vectors), that is, vectors of similar words are grouped together in a semantic vector space. The word2vec model has two architectures.

The first architecture is Continuous Bag-of-Words Model (CBOW) that predicts the current word based on the words around it, and the second architecture is Continuous Skip-gram Model that predicts the surrounding words based on the current word (see Figure 2.2). The training objective of CBOW is to find distributed word representation that can predict the current word based on words around it. It is similar to the feed forward neural network language model (NNLM) that was proposed in [18]. But it removes the most time-consuming non-linear hidden layer and only has 3 layers. The projection layer is shared for all words, and every word gets projected into the same position on the second layer. CBOW does not only use words from past, but also uses words from future. Unlike standard continuous bag of words model, it uses continuous distributed representation of the context [84]. So the best performance of predicting the current word based on i preceding words and i following words is obtained. The purpose of Continuous Skip-gram model is to predicts multiple surrounding words from one input word

as shown in Figure 2.2.

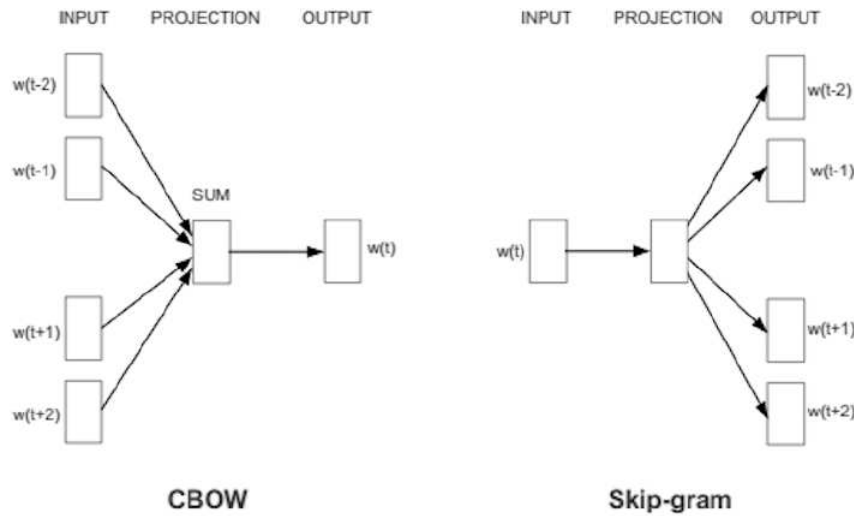


Figure 2.2: Two model architectures of word2vec. The CBOW model is to predict the current word based on the words around it, and the Skip-gram model can find the most likely surrounding words based on the current word.

By contrast with neural word embedding models, traditional DSMs can be considered as "count" models as they count co-occurrences among words by operating on co-occurrence matrices which decreases the similarity accuracy. For instance, VSM can only represent the effect of word frequencies in document content and meaning, and it can not express the phenomenon of polysemy and synonymy. This leads to the words that have the same meaning and different spelling is considered as totally different words, and the worst situation is it leads to the documents that contains these words are classified to different categories. LSA can solve this problem and it execute SVD and reducing dimensions on term and document relation matrix. The generated document vectors and the word vectors can well represent the latent semantic of documents or words, and it makes computing the similarity between words and words, words and documents, documents become more reliable. To some extent, it has solved the problem of synonymy that VSM can not resolve. But there still are the problem of polysemy.

In [16], Baroni et al compare word2vec word embeddings to traditional distributional semantics models. Their experiments show that neural-network-based word embedding models achieve better results than traditional DSMs such as VSM and LSA. Indeed, the word2vec model provides an efficient estimation of word representations in vector space. After training this model on a big data set, every word can be represented as a vector and this vector can fully express the sense of word. Based on this model, the problem of polysemy and synonymy has a good way to resolve.

2.2.2 Text clustering

Clustering algorithms aim at partitioning the amount of data by categorizing or grouping similar data items together into subsets or clusters [77]. The goal is to generate clusters with internal coherence, placing similar objects in the same group and assigning dissimilar objects to different groups. In this sense, requirements that belong to a certain cluster should be as similar as possible and dissimilar from requirements in other clusters. In order to group similar software requirements, there are mainly two essential factors to perform the clustering task: define a similarity measure among textual requirements and choose a suitable clustering algorithm. In what follows, we explain and clarify the existing methods for similarity computation and the clustering techniques.

Clustering algorithms

In terms of machine learning algorithms, clustering is an unsupervised task since it does not require training data and the result only depends on natural distribution of the data. The lack of supervision means that there is no human expert providing labeled training data, or assigning labels to data [95]. The clustering task depends on the distribution and makeup of the data that will determine cluster membership. It consists in dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. An input to the clustering algorithm is a component-attribute data matrix. Components are the entities that we want to group based on their similarities and attributes are the properties of the components.

Applications of clustering methods can be found in many disciplines [8, 114]. In the context of natural language requirements clustering, the goal is to provide an initial partition of the requirements based on a given similarity criteria. Such partition can give the designer some insight on the functional decomposition or possible conceptual components the desired architecture will have. Textual requirements clustering refers to the process of taking a set of requirements and grouping them so that, requirements in the same cluster are similar and requirements in different clusters are different. The input to the requirements clustering algorithm is a requirement-similarity matrix.

Clustering methods can be classified either as partitional or hierarchical [55]. Partitioning algorithms separate the data set into the specified number of clusters based on the similarity or distance among the data samples. Hierarchical algorithms compose the clusters in the hierarchical structure. In what follows, we present two examples of partitional and hierarchical clustering algorithms which are k-means and the Hierarchical Agglomerative Clustering algorithm (HAC).

- **K-means:**

K-means clustering [75] is one of the simplest and frequently used unsupervised learning algorithms, especially in data mining and statistics. Being a partitioning algorithm, its goal is to form groups of data points based on the number of clusters, represented by the variable k . k needs to be predefined before the execution. K-means uses an iterative refinement method to produce its final clustering based on the number of clusters defined by the user and the data set. Initially, k-means randomly chooses k as the mean values of k clusters, called centroids, and find the nearest data points of the chosen centroids to form k clusters. Then, it iteratively recalculates the new centroids for each cluster until the algorithm converges to one optimum value. k-means clustering would be suited with the numerical data with a low dimensionality because numerical data is used to compute the mean value. The type of data best suited for k-Means clustering would be numerical data with a relatively lower number of dimensions. The algorithm works as follows:

1. Place k points into the space represented by the objects that are being clustered. These points represent initial group centroids.
2. Assign each object to the group that has the closest centroid. The Euclidean distance can be used to calculate the distance between each data points and the initialized centroids.
3. When all objects have been assigned, recalculate the positions of the k centroids.
4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

K-means has only a few computations, only computing and comparing distance among data points and grouping clusters. Thus, it can be computationally faster than hierarchical clustering, having the time complexity of $O(n)$, where n is the number of data samples. Additionally, it can scale up to large data set and it can also adapt to new data samples. Despite these advantages, K-means has some disadvantages. The most important limitations of using k-means are:

- The number of clusters, k has to be specified manually.
- The clustering results can vary depending on initial values. K-means also randomly select the initial centroids for k clusters. Therefore, the results can be different from one execution and another, lacking inconsistency.
- K-means has difficulty with clustering data sets of varying sizes and density.

- **Hierarchical clustering:**

Hierarchical clustering algorithms seek to build a hierarchy of cluster. They work iteratively by connecting data points to form clusters based on their distance. It starts with some initial clusters and gradually converge to the solution. Hierarchical clustering has two categories: agglomerative and divisive. The agglomerative clustering algorithms initially takes each data point as an individual cluster and the iteratively merge the clusters until the final cluster contains all data points in it. This approach is also called a bottom-up approach. As an opposite technique of agglomerative clustering, divisive clustering techniques follow top-down flow which starts from a single cluster having all data points in it and iteratively split the cluster into smaller ones until each cluster contains one data point.

In what follows we present the steps followed by the standard algorithm for Hierarchical Agglomerative Clustering (HAC) which works in a bottom-up manner:

1. As an initial step, the algorithm takes each data point as a single cluster and we decide a specific proximity matrix to determine the distance between the clusters.
2. To find the closest pair of clusters, it computes the similarity (distance) between each of the clusters.
3. Then, the similar clusters are merged to form a cluster according to the distance function.
4. Iteration through step 2 and 3 continues until all data points are merged into one last cluster.

In general, hierarchical clustering is forming a single tree of clusters where each node is representing the clusters and each data point starts as a tree leaf. The root of the tree is the final cluster containing all of the data points. At different distances, different clusters will form, which can be represented using a dendrogram. The dendrogram represents the arrangement of the clusters in a hierarchical tree structure as shown in Figure 2.3. In a dendrogram, the y-axis marks the distance at which the clusters merge, while the objects are placed along the x-axis such that the clusters don't mix. When this hierarchy is cut off at a specific point based on predefined criteria such as text semantic similarity, a set of clusters is obtained. In the example below (Figure 2.3), the horizontal line crosses the hierarchy in four points (called cutting points) and we have four resulted clusters. One cluster containing 2,5, a second cluster containing 1, a third cluster containing 0, and a fourth cluster containing 3,4.

The advantage of using the HAC algorithm is that the number of clusters is not necessarily required to be specified in advance. Moreover, it can output the hierarchical structure of a

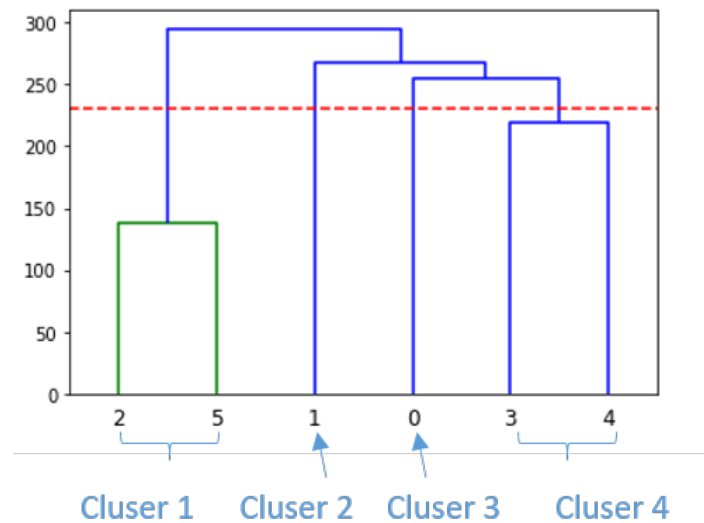


Figure 2.3: An example of graphical representation of a dendrogram tree

cluster tree (dendrogram), which can help in deciding the number of clusters. Nevertheless, the HAC algorithm has some disadvantages. The most important drawbacks is its time complexity. Comparing with other algorithms, it has a relatively higher complexity of $O(n^2 \log n)$, n being the number of data points.

2.3 Conclusion

In this chapter, we introduced the prerequisite knowledge for reading this thesis. Our research is interested in bridging the gap between natural language software requirements and architecture design models in the context of complex systems. In fact, providing a preliminary architecture design at early stage helps engineers to understand the system to be implemented. However, deriving architecture models from early requirements is usually performed manually by engineers and thus, it becomes time-consuming and error-prone task especially in the context of complex systems.

To address this, we proposed to employ advances in AI techniques to automate the transition from natural language system requirements to preliminary architecture design models. Basically, Natural Language Processing (NLP) and Machine Learning (ML) techniques are indispensable topics in this thesis, and we also introduce the main techniques that we used in the following chapters.

Chapter 3

State of the art

This chapter aims at assessing the state of the art of using machine learning and NLP techniques to automate/assist the extraction of architecture design models from early natural language requirements. We have explored the state of the art in three directions : (1) What are the main AI-based approaches used to extract architecture design models from natural language software requirements? (2) What are the main shortcomings and limitations of the used techniques? (3) To what extent do the current techniques succeed in automating the transition from natural language software requirements to visual architecture design models?

Afterwards, we give the research questions that we have identified and finally, we enumerate the contributions of the thesis.

3.1 Current research directions in deriving architecture models from natural language software requirements

The literature review shows that several NLP and machine learning techniques have been used in requirements engineering process in order to aid engineers in architectural software design based on early software requirements. We briefly describe the commonly used techniques and introduce how these techniques are generally used to bridge the gap between natural language software requirements and architecture models at early stages. The general process used by the current approaches is shown in Figure [3.1](#).

As a first step, natural language requirement documents are pre-processed in order to be easily analyzed by computers. In particular, natural language requirement documents are divided into words, phrases, or other meaningful elements (e.g., using tokenization). Additionally, the most com-

mon words that do not bring any linguistic information such as stop words (e.g., "the", "at", "and" etc) are removed. The other words can be tagged with their type (e.g., noun, verb, object, etc) using Part-of-Speech tagging (POS tagging).

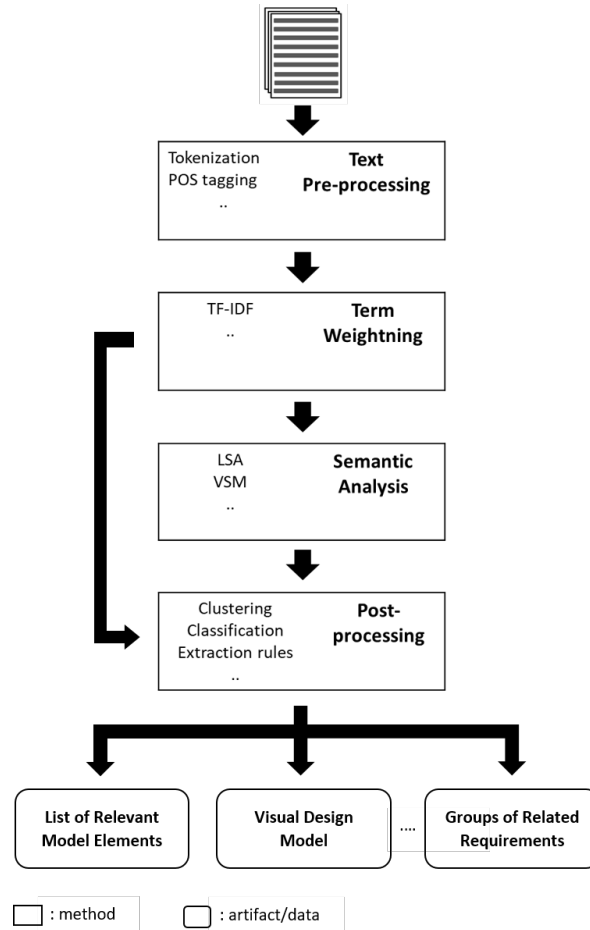


Figure 3.1: General process for applying AI techniques to bridge the gap between natural language requirements and design models

The second step is term weighting. This step is optional and it refers to the assignment of numerical values to terms that represent their importance in a document in order to improve retrieval effectiveness [99]. The commonly used technique for term weighting in current approaches is the Term Frequency-Inverse Document Frequency (TF-IDF) [92]. TF-IDF is intended to reflect the specificity of terms in different natural language documents by calculating the frequency of their occurrence in these documents. Hence, a term is considered important if it appears frequently in a document, but infrequently in other documents.

The next step is semantic analysis which is also an optional step. Semantic analysis is typically used to extract semantic information from text. In this step, Vector Space Model (VSM) and Latent Semantic Analysis (LSA) are widely used to conduct a semantic analysis. Using VSM, the pre-processed text is represented as vectors and then similarity is generally computed using a similarity

metric such as cosine similarity or the euclidean distance. LSA uses document-term matrix which describes the occurrences of terms in documents to analyze the similarity of documents.

After processing data by means of NLP techniques, the analyzed data are then post-processed in order to assist engineers in architectural software design. We observed that there exists two main research directions that were investigated in the post-processing step. The first direction focuses on extracting architecture design models from natural language requirements by generating the visual design model or extracting the list of relevant elements needed to build the model. Typically, specific extraction rules and algorithms have been mainly used to extract the set of relevant design model elements. The second direction focuses rather on grouping software requirements based on a certain criteria (e.g., similarity, shared functionalities etc) using machine learning techniques such as clustering and classification. Certainly, such grouping of software requirements can be easily mapped into design concerns of a possible architecture for the system. Particularly, with the increasing complexity of the developed systems, grouping similar requirements helps in assisting engineers by providing them with insights on the architectural design of the system decomposing its functionalities during early stages of the development process.

In what follows, we detail the related works interested in both directions we mentioned above. We provide detailed insights of techniques used, degree of automation achieved, shortcomings, and challenges ahead for extracting architecture design models from natural language requirements.

3.2 The machine learning techniques used to group natural language software requirements

Architectural software design, helps to analyze properties of complex systems, comprising a major issue in the building of new systems with solid foundations. In real life, requirement engineering and architectural modeling may not be as close as they seem to be in theory. Especially in the context of modern software systems, which are more complex than projects in the past, inter-dependencies and constraints between architectural elements and requirements are difficult to understand and trace during software development. One of the most used and efficient design paradigms to deal with complexity is the well-known “divide-to-conquer” strategy i.e., building smaller pieces to reduce the complexity. Herein lies the importance of an automatic solution to categorize software requirements into a set of groups in order to breakdown the target software system into a set of smaller sub-systems at early stages of the development process. Such requirements categorization methods provide an initial partition of the candidate architecture artifacts, that can give the software en-

engineers some insights on the system decomposition or possible conceptual components the desired architecture will have. Moreover, providing engineers with an automatic categorization of software requirements leads to a considerable gain in time and in development cost especially with the exponential growth of the number of the collected requirements in modern software systems. There are mainly two methods to categorize natural language software requirements: clustering and classification. Both methods aim to group software requirements based on some criteria. It is important to distinguish between these two main methods: the clustering is the process of grouping or organizing requirements into an undefined number of groups (clusters) while, the classification is the process of assigning requirements to a predefined number of classes. In the following, we explain both methods and present the related work interested in both categories.

3.2.1 Related works of requirements clustering

The usage of clustering techniques in the early phases of software engineering has gained a lot of attention in recent years.

In [7], the authors developed a tool based on hierarchical clustering of requirements in order to propose a packaging solution for software engineers. The tool's input is a software requirement document in the form of Message Sequence Chart (MSC). They defined a similarity measure that aims to cluster classes with high number communication in the same package. However, the optimal number of clusters is manually selected by software engineers based on the hierarchical tree generated by the clustering algorithm.

Casamayor et al. [26] propose an initial clustering of responsibilities from requirements, in order to identify architecture components. Responsibilities are verb phrases within sentences, which most likely will refer to tasks to be performed by the system and can be associated to some actor or component in the software model. The proposed approach firstly processes requirements documents by POS tagging technique to detect the actions and tasks that the system needs. Afterwards, the similarity function is computed based on the verb phrase each responsibility contains and the direct object it is related to. Then, a decomposition of responsibilities into functional components was performed and validated using four different clustering algorithms and several validity metrics.

Barbosa et al. [15] present an approach to cluster and sequence user stories in order to assist software engineers in the implementation phase. The sequencing of software requirements is based on a data dictionary that identifies the functional dependencies of the semantic steps in a given domain. To group the requirements, they start by vectorizing the sentences using the TF-IDF. Then, the partitional clustering algorithm K-medoids and the silhouette score were employed to identify the

best number of clusters.

In [60], the authors propose an approach that clusters similar requirements in order to reuse them in software product lines (SPLs). The work aims to extract features from requirements in order to identify similar requirement documents from online product reviews as a first step towards requirements reuse. First, they used tf-idf to represent features, then they employ the Latent Semantic Analysis (LSA) technique to compute the documents similarity. Then, they used the Agglomerative Clustering algorithm (HAC) to group similar product reviews as it provides better results compared to K-means.

In [96], the authors demonstrate the use of the Hierarchical Agglomerative Clustering algorithm (HAC) to break-down the project into a set of sub-projects based on related functional software requirements. They use traditional vector space models to vectorize text requirements. The clustering framework that they propose is dynamic and can be extended to include different clustering algorithms and distance measures.

In [44], the authors propose an approach based on clustering to identify relevant-architecture structures from both functional and non-functional requirements. Thus, the extracted structures help to derive strategies for the implementation of requirements in the architecture by providing information about when, where, and how to implement requirements in the architecture.

Table 3.1 summarizes the techniques used by these approaches, their inputs, their outputs as well as their degree of automation.

3.2.2 Related works of requirements classification

Requirements classification is the process of assigning requirements to a predefined number of classes. Automated classification is also known as supervised learning. Classification is the task of machine learning that learns a function that maps an input to an output based on labeled training data examples. In software requirements classification, training data are a set of pre-classified requirements that are similar to the requirements that are about to be classified. The algorithm "learns" where to classify test data (the requirements to be classified) based on the "supervision" of the training data. In requirements engineering, classifying software requirements by their kind into functional requirements and non-functional requirements [39, 108] is a widely accepted standard practice today. In what follows, we present research work interested in both categories.

- **Functional requirements classification**

Few works found in the literature of requirement engineering focus on the classification of functional requirements (FRs). In [107], Sommerville classifies FRs into two categories: user re-

Table 3.1: Overview of the existing approaches for clustering of natural language software requirements

Approach	Input	Output	Semantic analysis/ vectorization techniques	Clustering algorithm	Automation
Amannejad et al., 2014 [7]	Message sequence chart documents	Package model	The communication size between classes	HAC	Automatic
Casamayor et al., 2012 [26]	Set of responsibilities (verb phrases within a sentence)	Set of clusters grouping responsibilities	Not known	EM, cobWeb, Xmeans, DBScan	Semi-automatic
Barbosa et al., 2015 [15]	User stories (in semi-controlled natural language)	Clusters and sequences of user stories	TF-IDF	k-medoids	Semi-automatic
Jalab and Kasirun, 2014 [60]	natural language requirement documents	Clusters grouping similar requirements	TF-IDF, LSA	K-means, HAC	Semi-automatic
Salman et al., 2018 [96]	natural language requirement documents	Clusters of functional requirements	TF-IDF	HAC	automatic
Galster et al., 2013 [44]	natural language requirements	Clusters of relevant-architecture structures	Manually assigned values of architecture relevant attribute	A specific clustering algorithm based on euclidean distance	Semi-automatic

requirements and system requirements. Ghazarian et al. [46] introduce a requirements specification framework, called Problem Decomposition Scheme (PDS), that helps to classify manually FRs into only five classes. These classes include input data, output data, data persistence, application rules and actions.

In [45], Ghazarian also used FRs classes in [46] as a starting point to assign manually a type for each system FR in the context of web-based enterprise systems. The resulting classification scheme is composed of 12 classes which can be expanded and modified as necessary.

- **Non-functional requirements classification**

In contrast to the FRs classification, most existing works focused on non-functional requirements (NFRs) classification. In Sommerville's textbook [107], NFRs are classified into three main types: external, product and organizational requirements.

Many approaches are proposed to automate the process of identifying and classifying the NFRs.

Rashwan et al. [93] proposed an ontology-based approach to detect and classify NFRs. They classify NFRs into five types (maintainability, reliability, portability, security and usability/utility) using a Support Vector Machine (SVM) Classifier.

In [103], Singh et al. proposed a rule-based technique to identify and classify NFRs provided in the PROMISE corpus into eight types. Their technique relies on linguistic relations among requirement statements to extract thematic roles which describe the thematic relations in the sentences written in natural language.

In [25], the authors proposed a machine learning based approach for classifying requirements into two types (functional requirements and non-functional requirements), eleven types (non-functional requirements sub-classes), and twelve types (functional requirements plus non-functional requirements sub-classes). They combined two text vectorization (Bag of Words (BoW) vs. Term Frequency–Inverse Document Frequency (TF-IDF) vs. Chi Squared (CHI2)) techniques with four machine learning algorithms (Logistic Regression (LR), Support Vector Machine (SVM), Multinomial Naive Bayes (MNB) and k-Nearest Neighbors (kNN)) to classify requirements. The data used to carry out the research was the PROMISE_exp, a recently made dataset that expands the already known PROMISE repository¹, a repository that contains labeled software requirements.

In [115], the authors presented a semi-supervised learning approach to extract and categorize NFRs from a list of software requirement specifications. The approach consists in using a word2vec model trained on wikipedia dump and then used to represent words in each requirement statements as well as keywords that represent each NFR type. The semantic similarity is then computed and compared to a threshold value in order to identify the requirements sub-categories.

In [91], the authors proposed a requirement identification framework using RNN variants in order to classify NFR into pre-defined categories. First, NFRs are pre-processed to eliminate unnecessary contents and. Then, the pre-processed text is vectorized using word2vec algorithm to fed in the neural network model RNN variants. The LSTM and GRU variants of the RNN have been applied as NFR classifier.

Requirements are processed to eliminate unnecessary contents, then features are extracted using word2vec to fed as input of RNN variants LSTM and GRU.

¹<http://promise.site.uottawa.ca/SERpository/>

Table 3.2 summarizes the techniques used by the existing requirements classification approaches, their inputs, their outputs as well as their degree of automation.

Table 3.2: Overview of the existing approaches for classifying natural language software requirements

Approach	Input	Output	Terms Weighting	Grouping Method	Automation
Rashwan et al., [93]	Requirement corpus	Four types of NFRs (maintainability, reliability, portability, security and usability/utility)		(SVM) Classifier	Semi-automatic
Singh et al., [103]	SRS document	Eight types of NFRs		Rule-based technique	Automatic
Canedo and Mendes [25]	PROMISE_exp (extension of the PROMISE dataset)	Categorization into twelve types (functional requirements plus non-functional requirements sub-classes)	Bag of Words (BoW), TF-IDF, Chi Squared (CHI2)	Logistic Regression (LR), Support Vector Machine (SVM), Multinomial Naive Bayes (MNB), k-Nearest Neighbors (kNN)	Automatic
Younas et al., [115]	tera-PROMISE and CCHIT datasets	Sub-categories of NFRs	word2vec model trained on wikipedia dump	Semantic similarity based on the word2vec model trained on wikipedia dump	Automatic
Rahman et al., [91]	PROMISE dataset	sub-categories of NFRs	word2vec	RNN variants: LSTM and GRU	Automatic

3.2.3 Summary

Automatic methods to categorize software requirements have become an essential task within software engineering. In fact, such categorization helps in decomposing the system into a set of smaller sub-systems and identifying architecture candidates at early stages.

As shown in Tables 3.1 and 3.2, most of the proposed approaches for this task are based on machine learning techniques which are mainly requirements classification and requirements clustering.

The Promise Repository has been widely used as a training dataset in software requirements classification tasks. It consists of 625 labeled natural language requirements (255 FRs and 370 NFRs). The labels classify the requirements first into FR and NFR. Within the latter category, eleven sub-categories are defined: (a) ten quality requirement categories: availability, look and feel, maintainability, operability, performance, scalability, security, usability, fault tolerance, and portability; (b)

one constraint category: legal and licensing. Although these supervised learning methods succeeded to provide relatively accurate results in most cases, they are labor-intensive and they have the overhead to train the model. Moreover, if the domain of application changes, experts have to pre-categorize manually a huge amount of requirements in order to build the new training dataset. Hence, a lot of effort is required to train the new model in order to get acceptable results.

Requirements clustering approaches, on their side, have recently seen a surge of interest as way to gather software requirements based on their similarity. Since they are based on unsupervised learning, requirements clustering methods do not require a training dataset. However, most of the existing requirements clustering approaches suffer from a lack of automation when defining the optimal number of clusters [7], others rely on the similarity between words or concepts in each requirement [73, 26]. Moreover, many works utilize traditional DSMs which are considered as “count” models such as Vector Space Model (VSM) [15, 96] and Latent Semantic Analysis (LSA) [60] to calculate the similarity. However, using traditional DSMs to identify the semantic similarity among sentences does not usually succeed to achieve accurate results. The main limitation of these techniques is that they rely on counting the co-occurrences among words by operating on co-occurrence matrices. Thus, sentences with similar context but different term vocabulary will not be considered as similar.

In summary, several machine learning based approaches have been proposed to organize and group requirements encompassing their strengths and their weaknesses in providing a preliminary decomposition of complex systems at early stages [7, 26, 15, 60, 96, 44, 93, 103, 25, 115, 91]. Nevertheless, only a few of the existing works focus on the software modeling phase [7, 26] which is a crucial phase for building new systems with solid foundations. Hence, in the next section, we complement our review by investigating the existing works focusing on automating the extraction of design models from natural language requirements.

3.3 From natural language requirements to visual models

The idea of extracting knowledge from natural language requirements and represent it with semi-formal models has also been investigated throughout these years. New tools and approaches have been proposed to support the modeling phase during the software development process. This reflects the use of object oriented design (OOD) paradigm [12] also called Component Added Software Engineering (CASE) [21] which encourages the use of Unified Modelling Language (UML) for modelling the user requirements. In a conventional OOD software modelling approach, the system analyst first has to spend a lot of time understanding the gathered requirements and then, based on the requirements analysis made, CASE tools are used to build the UML models.

Over the last 20 years, researchers focused on automating the process of extracting valuable information from natural language software requirements in order to process models and extract architecture relevant elements. In this context, organizations aim to automate the process of capturing models from software requirements in order to improve their efficiency, reduce time and costs, and/or reduce human beings errors. Hence, several natural language based CASE tools that utilize different levels, or combinations of levels of linguistic analysis have been proposed in order to generate OO models from the natural language requirements.

In what follows, we present some of the NLP based CASE tools and approaches that have been proposed in literature and that are interested in generating design models from natural language requirements.

3.3.1 Related works

In this section, we detail the existing CASE tools as well as approaches that are based on NLP techniques to extract visual design models.

- **LIDA**

The Linguistic Assistant for Domain Analysis (LIDA) [89] provides linguistic assistance to construct UML class model from natural language requirements. The proposed tool relies heavily on the analyst's intervention. The analyst imports the requirements documents to be analyzed and then she/he selects candidate classes, attributes, methods and roles. After this identification process, the analyst employs LIDA Modeler to graphically associate the identified attributes, methods and roles with the appropriate classes.

- **CM-Builder**

The Class Model Builder (CM-builder) [51] uses robust NLP techniques to analyze textual requirements and perform domain independent OO analysis. The tool constructs an integrated discourse model, represented in a Semantic Network (SN). By converting nouns into classes and verbs into relationships, The resulted SN is then used to automatically construct an initial UML class model. However, the CM-builder has a limitation in its linguistic analysis due to the ambiguity, fuzziness, and redundancy of NL.

- **UMGAR**

Deeptimahanti and Babar presented the semi-auto-matic tool UML Generator from Analysis of Requirements (UMGAR) [33] which assists developers in generating UML Use-case models,

Design class models, and Collaboration models from natural language requirements. The proposed tool has been developed using three efficient NLP technologies:

- Stanford Parser: to generate parse tree and extract relevant concepts like actors, use cases, classes, methods, attributes, and associations.
- WordNet2.1: is a large lexical database to perform morphological analysis.
- JavaRAP²: to replace all the possible pro-nouns with its correct noun form.

Although it can be used for large requirement document, the UMGAR tool relies on human interaction, for instance, to identify aggregation/composition relationships among objects.

• UMLG

The Unified Modeling Language Generator (UMLG) [12] relies on a rule-based approach to automatically analyze the natural language text, extract concepts such as classes, attributes and associations, and generate OO modeling (class model, sequence model, use case model..). UMLG is also enable the conversion of the object-oriented modeling information in several languages such as Java, C#.NET or VB.net.

• DC-Builder

The Diagram Class Builder (DC-Builder) [52] is an automated tool based on NLP techniques and domain ontologies that aims to analyze the users' requirements to facilitate the extraction of the class model. First, the requirements descriptions are analyzed using the Nearly-New Information Extraction System (ANNIE) for natural language processing of the GATE framework. Then, UML relevant concepts such as classes and attributes are extracted using a set of rules. The extracted model elements are stored in a XML file and domain ontologies are used to eliminate these irrelevant elements.

• Rapid

More and Phalnikar have proposed a tool called Requirement Analysis to Provide Instant Diagrams (RAPID) [86] that analyzes the natural language requirements, extracts relevant model elements and produces UML models. PAPIID is based is mainly based on NLP components such as:

- OpenNLP³: provides lexical and syntactical parsers of the natural language requirements.
- RAPID Stemming Algorithm: identifies the root of words from natural language text.
- WordNet2.1: provides semantic parsing that is used to validate the semantic correctness of the sentences generated at the syntactic analysis.

²JavaRAP, <http://www.comp.nus.edu.sg/qjul/>

³<http://opennlp.sourceforge.net/>

The tool extracts the relevant elements of UML class models using NLP rules and it uses ontologies to improve the identification of the extracted relevant model elements. One limitation of this tool is that each sentence in the requirements document has to satisfy a specific structure defined by the system.

- **ABCD**

The Automatic Builder of Class Diagram (ABCD) [63] is an automated tool implemented in Visual Basic.Net that generates UML class models from natural language user requirements. This tool performs on lexical and syntactical processing of natural language requirements using the Stanford NLP toolkit. The UML model relevant elements are extracted using a pattern-matching NLP techniques then, the UML class model is built using a CASE tool. One of the limitations of this tool is that it confuses the concepts of association and method identification and fails to deal with redundant information problem.

- **Gilson and Irwin** [47] proposed an automatic transformation from user stories to robustness models based on NLP techniques. The approach aims at helping requirements engineers and users to validate user stories and to perform structured analysis. It consists in parsing user stories using to generate a dependency tree. The generated tree is then chunked into tagged words to be parsed against linguistic rules to generate the named entity graph. The parsed dependency tree is transformed to generate the graph of connected named-elements and then, post-processed to be transformed a set of robustness model objects.
- **Elallaoui et al.** [40] proposed an approach that helps engineers to reduce ambiguity in requirements specifications in the Scrum processes. The approach consists of an algorithm that generates sequence models from user stories in order to generate test cases. It is based on a set of rules that are used to extract the relevant model elements from user stories. Then, an XML file defining the corresponding sequence model for each user story is generated. The resulting XML file format then, transformed into a sequence model using the UML2 tool SDK plugin for Eclipse. In [41], they propose an automatic transformation of user stories into UML use case models to assist the work of the development team and the Product Owner. The approach is based on a set of NLP heuristics that enables the extraction of the model concepts. However, the proposed approach fails to detect actors with compound nouns as well as inclusion and extension relations between use cases.
- **Arora et al.** [9] developed a domain model extractor from natural language requirements. They are able to identify classes, relations and attributes by using existing extraction rules in the

Table 3.3: Overview of the existing approaches for extracting design models from natural language software requirements

Approach	Input	Output	Automation	Limitations
LIDA [89]	Natural language requirements	UML class model	Semi-automatic	Needs extensive user intervention
CM-BUILDER [51]	Natural language requirements	UML class model	Automatic	Cannot capture candidate model objects
UMGAR [33]	Natural language requirements	Use case model, class model and collaboration models	Semi-automatic	Requires human intervention to eliminate irrelevant classes and to identify aggregation/composition
UMLG [12]	Text scenarios in natural language	Use case model, class model, sequence model and activity model	Automatic	The input requirement document is not free natural language text
DC-Builder [52]	Natural language requirements	Class model	Automatic	Not all concepts are identified
Rapid [86]	Natural language requirements	Class model	Semi-automatic	Each sentence in the requirements document has to satisfy a specific structure defined by the system
ABCD [63]	Natural language requirements	Class model	Automatic	-Fails to deal with redundant information problem -confuses the concepts of association and method identification
Gilson and Irwin [47]	Natural language user stories	Robustness model	Automatic	Model elements extraction is not accurate enough for longer sentences
Elallaoui et al., [40]	Natural language user stories	Sequence model	Automatic	Fails to detect actors with compound nouns as well as inclusion and extension relations between use cases
Arora et al., [9]	Natural language requirements	Candidate class model elements	Automatic	Lack of accuracy in terms of some extracted relations type and cardinalities

software engineering literature, extending these rules with complementary rules from the information retrieval literature, as well as proposing new rules to better exploit results obtained from modern NLP dependency parsers.

Table 3.3 provides an overview of the existing works that focus on extracting design models from natural language requirements. We observe that most of the existing approaches are rather immature, i.e., they suffer from lack of accuracy as most of the extracted design models does not capture all the relevant model elements, or, the generated design models are not accurate enough (e.g., [33], [52], [63], [47], [40], [9]). Moreover, some of current approaches fail to achieve a high degree of automation and rely heavily on the analyst's intervention (e.g., [89], [33], [86]).

In summary, existing approaches present several limitations that may hinder their applicability in particular for complex systems. Hence, we believe that starting by decomposing the complex system at early stages (as stated in section 3.2) is crucial as it helps to analyze each sub-system separately and provide a design model with different levels of abstraction. Then, it is necessary to complement this step by providing an NLP-based process that enhances the accuracy of the extracted design models.

3.4 Summary

In this chapter, we have presented the related work for using AI techniques to bridge the gap between natural language software requirements and architecture models in order to improve the efficiency, reduce time and costs of developing complex systems.

Table 3.4 summarizes the existing studies that aim to automate the transition from natural language system requirements and architecture design models. The cross (X) shows that the feature is not used by the study and the check mark (✓) shows that the feature is used by the study. As shown in Table 3.4, current approaches are mainly interested in two research directions, i.e., they focus either on decomposing the complex system at early stages or directly extracting design models from the system requirements. Additionally to their lack of accuracy and automation (see Tables 3.1, 3.2 and 3.3), existing approaches in both research directions present several weaknesses.

For instance, existing works interested in supporting the decomposition of complex systems based on early system requirements present potential limitations related to the used techniques that can be summarized in the followings:

- *L1*. Some proposed approaches are based on supervised classification of natural language requirements. Typically, these approaches focus on either identifying functional requirements and non-functional requirements or classifying non-functional requirements into sub-categories based on their type (see Table 3.2). However, among these approaches, there is no approach

Table 3.4: Summary table of recent work in automating the transition from natural language requirements and architecture design models

Approach/Tool	Support of the decomposition of the system based on system requirements					Support of the generation of design models
	use of external training dataset	use of traditional DSMs	semantic Similarity computation	classification	clustering	
Rashwan et al. [93], Singh et al. [103], Canedo and Mendes [25]	✓	✓	X	✓	X	X
Younas et al. [115]	X	X	✓	X	X	X
Rahman et al. [91]	✓	X	X	✓	X	X
Lucassen et al. [73], Casamayor et al. [26], Barbosa et al. [15], [60], Salman et al. [96], Galster et al. [44]	X	✓	✓	X	✓	X
Amannejad et al. [7]	X	✓	✓	X	✓	✓
LOLITA [81], D-H [35], LIDA [89], GOOAL [90], CM-Builder [51], UMGAR [33], UMLG [12], DC-Builder [52], Rapid [86], ABCD [63], Gilson and Irwin [47], Elallaoui et al. [40], [41], Arora et al. [9]			X			✓

that focuses on classifying early functional requirements: first because grouping functional requirements requires an important semantic analyses information extraction. Second, the classes of the generated groups of the outcomes are not always known in advance unlike NFRs, which are generally classified into eleven sub-categories [2].

- L2. Approaches that are based on supervised classification require an external training dataset to group requirements. For this, the PROMISE repository for requirements engineering has been used in most of these approaches either to classify requirements into FRs and NFRs or, to classify NFRs into subcategories.

However, when a classification task of requirements related to a specific domain of application needs to be executed (e.g., grouping FR requirements of a given system by functionality), chances are that no relevant dataset exists. Thus, high-quality dataset creation is needed if datasets containing requirements related to a specific domain of application are not readily available, which is often the case [48].

Hence, the domain experts have to prepare training data manually, that is, a huge amount of pre-categorized requirements is needed. Hence, a lot of effort is needed to train the model in order to get acceptable requirement classification results. In addition, if the domain of application changes, then analysts need to build and/or retrain the new model which is a labor-intensive task.

- L3. Unsupervised clustering based approaches have been also used to provide an early grouping of similar requirements. Such clustering based approaches may mitigate the issues raised by the classification based approaches since they do not require a training dataset. Thus, system requirements are grouped together based on their similarity. However, as shown in Table 3.4 requirements clustering approaches are based on traditional DSMs to compute the similarity of the system requirements. The shortcoming of applying traditional DSMs is that they are considered as "count models" as they count co-occurrences among words by operating on co-occurrence matrices. Consequently, they usually achieve worse results than neural-network based natural language processing architectures which can be seen as prediction models [16].

Since the arrangement of the grouped requirements provides a general overview of the complex system, it can be employed as a first step towards architecture design generation. However, there is no approach that focuses on the design phase among the approaches mentioned above. Hence, we investigated the related works interested in automating the generation of architecture design models from natural language requirements. We observed that most of the existing approaches in this research direction also suffer from several limitations that can be presented as follows:

- L4. The majority of the proposed approaches fail to achieve a high degree of automation as they heavily rely on the requirements analyst's intervention (see Table 3.3).
- L5. Moreover, automatic models extraction approaches are either poor, i.e., the extraction process does not capture all the relevant elements that are required to build the target design model, or, the generated architecture models are not accurate enough.

In summary, the existing approaches in both research directions present several limitations which may hinder their applicability in practice. To the best of our knowledge, there is no work that proposes an automated decomposition of complex systems based on early requirements as a starting point towards automated architecture design models generation.

Nevertheless, we were inspired a lot from the existing approaches as they succeeded to a certain degree in providing engineers with an early decomposition of the complex system and giving insight about its functionalities described by the generated design models. Herein lies the need to combine both approaches in order to benefit from their strengths, mitigate their limitations and empower their automation.

In this thesis, we propose an automated approach that aims to provide engineers with an architecture design of the complex system decomposing its functionalities based on early requirements. Hence, the proposed approach aims to: (i) benefit from the strengths of the existing approaches (see Table 3.4) interested in both decomposing the system into smaller sub-systems based on system requirements and, providing engineers with visual design models with further degrees of details describing each sub-system; (ii) mitigate the limitations addressed by these approaches by providing full automation of the process and improving the accuracy of the extracted features (i.e., both requirement groups and the extracted design models).

By achieving the above key goals, we provide engineers with insights on the decomposition of the complex system that enables them to better understand the features to be implemented, shorten the development process, and to reduce costs and errors.

3.5 Research questions

AI-based methods have shown their ability to enhance the software development process including the requirements modeling. Therefore, we aim to benefit from these advances to automate the transition from natural language software requirements to architecture design in the context of complex systems. The main research statement of our work is the following:

"Which methodology is sufficiently effective to deal with both the complexity of the soft-

ware systems and automating the generation of architecture design from natural language requirements?"

The main research question aims at defining a methodology based on AI components, that should deal with the challenges addressed by both (1) the complexity of systems due to the increasing number of requirements, which raises the need to break-down the system into a set of smaller sub-systems; (2) the automation of the generation of the UML architecture design models representing each sub-system in order to provide a holistic visibility of the target system. We decomposed the above research question into two finer-grained, more focused research questions that constitute the main concerns of this dissertation:

- **RQ1- "How to guarantee an accurate decomposition of the complex system at early stages of the development process?"**

The goal of this research question is to define a method that groups requirements based on their similarity. For this, it is crucial to:

- First, define a similarity computation method that ensures effective identification of the natural language requirements that share similar characteristics. In this context, many computation methods have been proposed in literature to compute the similarity between natural language requirements statements. However, existing methods are rather immature that is, they are lacking accuracy (regarding the identified clusters of requirements) and suffer from incomplete extracted semantic information. Thus, our goal is to propose a similarity computation module that enhances the extraction of the semantic information among the requirement statements and enables a high degree of automation of the process.

- Second, define a convenient machine learning based method that ensures an accurate grouping of natural language requirement statements based on the extracted semantic similarity among early requirements. The grouping solution should provide a high degree of automation, speed-up the grouping process and provide an accurate grouping of early requirements. The extracted groups of similar requirements will represent a first partition of the complex system into smaller sub-systems that can be exploded into architecture design models with further degrees of detail.

- **RQ2- "How to effectively automate the generation of preliminary architecture design denoting the system's decomposition based on early requirements?"**

Natural language is the predominant notation that practitioners use to represent system requirements. Albeit easy to read, natural language does not readily highlight key concepts and relationships such as dependencies. This contrasts with the inherent capability of design models to visualize a given system in a holistic fashion. Herein lies the importance of an automatic solution that bridges the gap between natural language requirements and architecture design models.

Indeed, after decomposing the system into a set of sub-systems, providing a clear holistic design of the resulted sub-systems is crucial. Such visual representation helps engineers to better understand the system features they are investigating and to speed-up the development process of the target system. However, due to lack of powerful tools, especially tools enabling high automation of the process, model-based approaches are still only marginally adopted by software engineers. Thus, the second research question is addressed by the definition of an AI-based solution that provides a high degree of automation enabling the generation of preliminary UML architecture design models describing the system's decomposition as well as its expected features.

3.6 Scope of contributions

Clearly, using AI-based methods to bridge the gap between natural language requirements and architecture design models is crucial to assist engineers in their tasks, reduce project failure rates and enhance the performance of the software development process. Our thesis contributions consist in proposing a new flow of AI components including their specific parametrization, enabling the automation to go from natural language requirements to the architecture design of complex systems. In this section, we present the scope of the contributions of this thesis work.

The input to this thesis work is a set of software requirement documents expressed in natural language. Software requirement documents commonly have two types of requirements one is functional requirements, which defines the feature of the system-to-be, and the other is non-functional requirements, which defines the quality attributes of the system features. In our research, we start by applying our contributions to functional software requirements. Indeed, developers emphasize more on the functional side of the software to understand the features to be implemented. The input requirements are from different domains, expressed in natural language and written in different styles, which allows us to assess the applicability of our contributions.

Our goal is to automatically generate the preliminary architecture design describing the features of the target system. For this, we use the Unified Modeling Language (UML) [21] as a visual language

to represent the output of our work. In fact, UML has been widely used to specify the features of software systems, and to reduce the ambiguity between the requirement specifications and the design, on the basis of models. Among these models, UML use case models are typically developed in the early stages of development. use-cases have been widely used to capture the requirements from the user's point of view. In addition, they are easy to understand and provide an excellent way for communicating as they provide a semi-formal framework for modeling (mainly functional) requirements [3]. Hence, we generate as output a UML use case model describing the primary specification of the functional requirements for the system.

The generated UML use case model contains a set of use-case descriptions in text, each describing one use case. Each use case description specifies a required functional service that the system is expected to provide for certain kinds of users called actors. The UML use case model may contain packages that are used to structure and decompose the model to simplify analysis, communications, navigation, development, maintenance and planning [59].

3.7 Contribution Overview

In real life scenarios, requirement engineering and architecture design may not be as close as they seem to be in theory. Inter-dependencies and constraints between architecture artifacts and natural language requirements are difficult to understand and model during software development. This is mainly due to the increasing complexity of the modern software systems as well as the lack of automation tools providing an accurate architectural design at early stages of the development process.

In our thesis work, we propose an ***AI-based methodology that enables the automation to go from natural language system requirements to architecture design decomposing the target system during early stages of the development process*** as shown in Figure 3.2.

Our methodology deals with the two research questions mentioned before. The first research question (1) ***"How to guarantee an accurate decomposition of the complex system at early stages of the development process?"*** is addressed by the definition and the implementation of a machine-learning based methodology that enables the grouping of similar software requirements and thereby, it enables the decomposition of the software system into a set of sub-systems at early stages. Typically, there exists two methods to achieve this goal: requirements classification and requirements clustering.

Nevertheless, high-quality training data that are needed for the requirements classification task are not always available for all domains of application. Hence, as our goal is to provide an applicable method for all domains of application, we propose a requirements clustering solution based on the

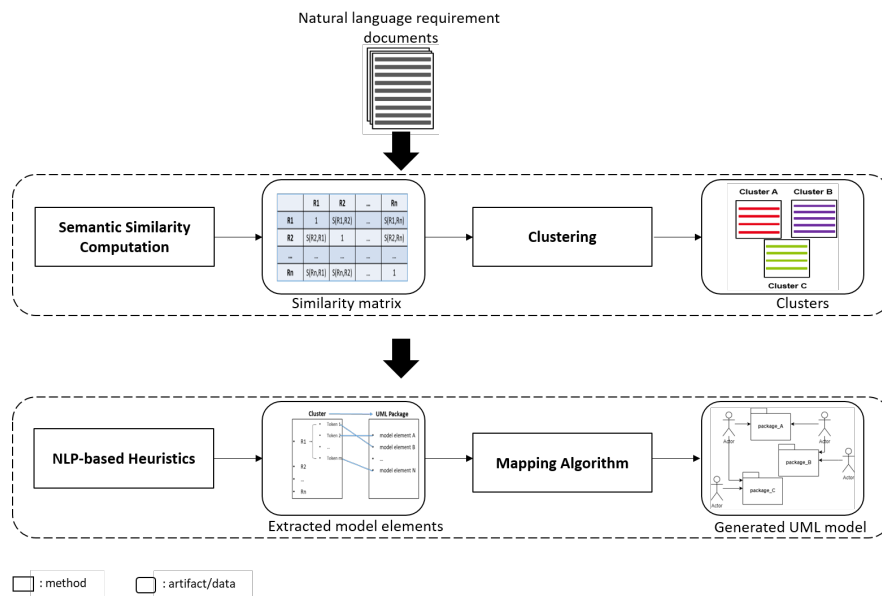


Figure 3.2: Overview of the proposed approach

semantic similarity of requirements. The proposed clustering solution aims to provide a first partition of the complex system into a set of sub-systems by grouping similar requirements in the same cluster. Thus, each cluster defines a sub-system that covers a particular functionality/characteristic of the complex system and can be exploded into architecture design components with further degrees of detail.

As shown in Figure 3.2, The proposed clustering solution is based on a semantic similarity computation module that extracts the semantic similarity of requirements. The similarity computation module is mainly based on word embeddings and it takes into account both word-level and requirement-level similarity in order to enhance the semantic similarity extraction among requirement statements. Accordingly, the semantic similarity computation includes two steps : (i) word-level similarity: we use a neural word embedding model, word2vec, as a predictive model to compute the semantic similarity between each pair of words in each different pairs of requirement statements. (ii) requirement-level similarity: we extend the word-level similarity to the requirement-level using a scoring function for text similarity. The used scoring formula takes into account the word-to-word semantic similarity and generates as output the similarity matrix of each pair of requirement statements in the document.

Additionally, we implemented an operation that identifies automatically the optimal number of clusters to be generated in order to reduce the manual intervention. Finally, requirements are grouped based on their semantic similarity scores using a clustering algorithm and the generated clusters are labelled automatically based on a key words ranking function.

The research question (2) **"How to effectively automate the generation of preliminary archi-**

ecture design denoting the system's decomposition based on early requirements?" is addressed by the definition and the implementation of a model extractor that automatically generates the UML use case model denoting the system's decomposition based on the identified clusters of requirements.

In order to achieve this goal, we first define and implement a set of specific NLP heuristics that extract the relevant design model elements from each requirement statement within each identified cluster. The preliminary generated architecture model consists of a UML use case model containing a set of packages of use-cases describing the initial decomposition of the target system. Each package represents a cluster (i.e., sub-system) covering a set of similar requirements. The goal is to extract the relevant model elements that are embedded within each cluster and that are needed to build the UML use case model. Hence, the outcome of this step provides further degrees of detail about the identified packages decomposing the complex system.

Then, the second step consists in implementing a mapping algorithm that maps the extracted model elements as well as the identified clusters of requirements into their corresponding ones in the UML use case model. Thus, we programmatically obtain a generated UML package break-down model denoting the system's decomposition, including a UML use case model providing a holistic view of the target system.

3.8 Conclusion

In this chapter, we presented the existing works interested in structuring natural language software requirements for architecture design as well as extracting visual design models from software requirements. We also detailed the investigated research questions and we gave an overview of the contributions of this thesis. In a nutshell, our contributions consist of a new flow of AI components including their specific parametrization, enabling the automation of the transition from natural language requirements to the architecture design of complex systems. In the following chapter, we describe in details the contributions of this thesis.

Chapter 4

Semantic Clustering of System Requirements

In the previous chapter we gave an overview of the contributions of this thesis, namely the semantic clustering of natural language system requirements and the automatic generation of the preliminary architecture design models denoting the system's decomposition. In this chapter, we detail the first contribution that consists of a semantic clustering solution that deals with decomposing the complex system into smallest sub-systems based on early system requirements. First, we give an overview of the proposed semantic clustering approach. Afterwards, we detail each step of the proposed approach.

4.1 The semantic clustering solution overview

Nowadays, modern software projects are many times larger and more complex than projects of the past. This is mainly due to exponential increasing of the number of system requirements so that traditional requirements management and organisation techniques become unwieldy.

Most of the classic problems of developing software products derive from this essential complexity and its non-linear increases with size [22]. One of the most challenging tasks is manually capturing design models from these large sets of requirements. Actually, this task becomes tedious, cost and time-consuming and error-prone as the number of software requirements are exponentially increasing. Moreover, the extracted models become quickly too large to be effectively explored by analysts. One strategy for reducing the complexity of the developed systems is the well-known “divide-to-conquer” strategy i.e., decomposing the target system into smallest sub-systems and treat each sub-

system separately. However, there is a lack of automatic solutions to group software requirements and even the existing tools/methodologies are rather immature, that is, they lack of accuracy.

In order to overcome these limitations, we propose an automatic clustering approach that groups natural language software requirements based on their semantic similarity. Such automatic requirements clustering solution helps to break-down the target system into a set of sub-systems at early stages of the development process. Eventually, each sub-system covers a set of related software requirements and could be developed by a separate and specialized developer team. Additionally, these groups of software requirements help to generate the software requirements specifications, which can be used for validating and verifying the final software system. Furthermore, the identified clusters help to design the preliminary architecture of the software system as they represent components or sub-systems that should be implemented and reused.

As shown in Figure 4.1, the proposed clustering approach consists of four major steps that are based on machine learning and NLP techniques.

Initially, the natural language software requirements are preprocessed. Text preprocessing is an essential step in the pipeline of a NLP system as it transforms the text into a form that is predictable and analyzable for machine learning tasks.

Then, we implement a semantic similarity module that computes the semantic similarity between each pair of the requirements, taking as input the preprocessed requirements generated in the first step. The similarity computation module includes two levels: (i) word-level similarity computation, in which we use a predictive word embedding model, word2vec, to compute word-to-word semantic similarity, (ii) then, we extend it to the requirement-level similarity computation by means of the Mihalcea scoring formula for documents similarity computation [83]. The output of this module is a requirement semantic similarity matrix that includes the semantic similarity between each pair of requirements in the input requirement document.

The obtained similarity matrix is then fed into the Hierarchical Agglomerative Clustering algorithm (HAC) which generates the semantic clusters of requirements. Finally, each identified cluster is labelled in order to bring a semantic information about the features that it embeds. In what follows, we detail each step of the proposed semantic clustering of requirements approach.

4.2 Preprocessing

The very first step of our clustering approach is requirements preprocessing. Preprocessing the natural language requirements consists in cleaning and normalizing the natural language text for further processing such as requirements similarity computation and requirements clustering. Requirements

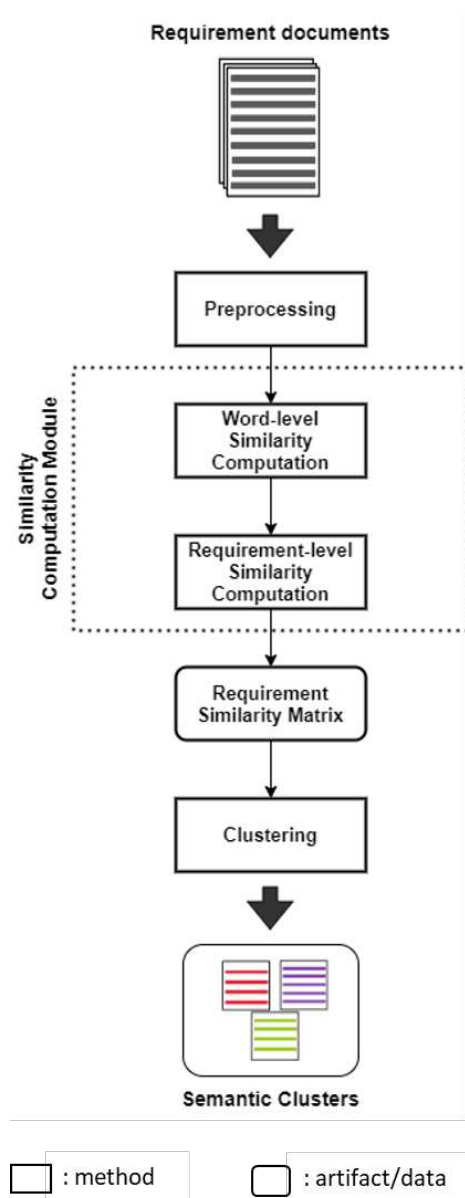


Figure 4.1: The requirement semantic clustering approach

normalization is achieved by performing four tasks: cleaning, tokenization, annotation and normalization. These tasks are executed according to the following order by means of the Natural Language Toolkit NLTK [19].

4.2.1 Cleaning

The input requirement texts usually comprise various types of data, such as words, punctuation, symbols, etc, while not all the data is helpful for a particular task. In order to facilitate further analysis, we just keep the relevant information of the requirement text. For this, the performed operations are as follows:

Stop words and punctuation removal

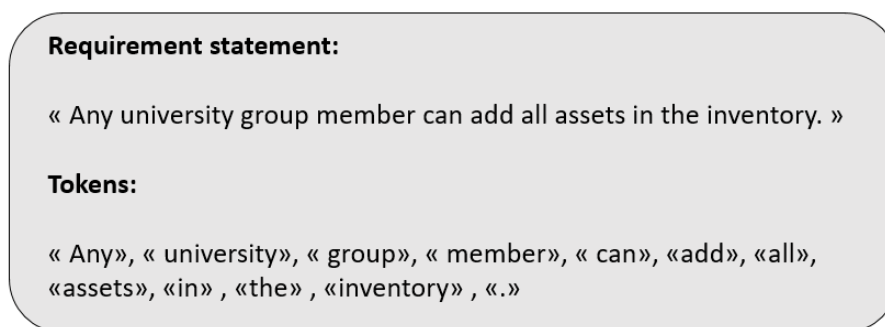
Usually, natural language requirements contain lots of noise such as punctuation marks, and many frequent words that do not have an impact on the general orientation of it such as stop-words (e.g., "as", "of", "the", "to", etc...). Keeping those words makes the dimensionality of the problem high and hence the clustering more difficult. Hence, this type of tokens are removed.

Lower case

Letters and words are often written either in upper case or lower case. For example, the letter at the beginning of the sentences is capitalized. In NLP, words in lower case are usually regarded as the standard form in order to simplify the analysis process.

4.2.2 Tokenization

In the tokenization task, each set of requirements are divided into individual statements using comma and dot delimiters. Then, each statement is divided into tokens (individual words) based on white space. An example of tokenizing a requirement statement is as follows:



Requirement statement:

« Any university group member can add all assets in the inventory. »

Tokens:

« Any», « university», « group», « member», « can», «add», «all», «assets», «in» , «the» , «inventory» , «.»

Figure 4.2: An example of a tokenized sentence.

According to the example above, the original requirement statement is transformed into a list of tokens that can be seen as meaningful units for further analysis process.

4.2.3 Annotation

Annotation in NLP helps to annotate the meaningful words from the sentence to make it usable for machine learning tasks understand texts.

Part-of-Speech tagging

Part-of-Speech tagging (POS tagging) is the most popular annotation task for preprocessing. It aims to assign a meaningful tag to each word in a sentence, while the tag presents a certain kind of

linguistic information of the word. By using POS tagging, the part of speech of each word or token can be assigned, for example, noun, verb, adjective, etc. The tag as a sort of a priori knowledge helps the machines to process and understand natural language. In Figure 4.3, the line under the tokenized requirement statement presents the POS tags of each word.

Tokens:	Any	university	group	member	can	add	all	assets	in	the	inventory
POS tags:	DET	NOUN	NOUN	NOUN	MOD	VERB	DET	NOUN plural	IN	DET	NOUN

Figure 4.3: An example of a parsed sentence with POS tags.

4.2.4 Normalization

A word can be changed into different forms in terms of the way of being used, such as presenting different tenses (i.e., the past, present and future tense). However, different forms of a word (i.e., the inflected words) may increase the complexity of processing a sentence for some specific NLP tasks. Normalization aims to convert a list of words to a more uniform sequence. This is useful in preparing text for later processing. When we normalize text, we attempt to reduce its randomness, bringing it closer to a predefined “standard”.

Stemming

Reducing tokens to their stems in information retrieval is known as stemming [78]. Stemming is a normalization technique that aims to remove inflectional endings and to return the base or dictionary form of a word. For example, "adding" and "adds" are reduced to its root "add". In Figure 4.4, we present an example of a stemming operation for the requirement statement "Any university group member can add all assets in the inventory".

Tokens:	Any	university	group	member	can	add	all	assets	in	the	inventory
Tokens after stemming:	any	univers	group	member	can	add	all	asset	in	the	inventori

Figure 4.4: An example of tokens after conducting stemming.

After preprocessing, there are further processing steps i.e., mainly the requirements similarity computation and the clustering. In the following section, we describe in details the similarity compu-

tation module.

4.3 Semantic similarity computation module

As we aim to group textual requirements in order to decompose the target system, we need to identify textual requirements that are semantically similar to one another.

Traditional approaches to compute the similarity between two text segments consist in using lexical matching method, and producing a similarity score based on the number of lexical units that occur in both input segments. However, these lexical similarity methods cannot always identify the semantic similarity of texts as they aim to determine whether the words in two texts have similar spellings [83]. For example, the “US” would be closer to the “UK” this way, than it would be to the “States”.

Going beyond these traditional methods, and in order to enhance the semantic similarity extraction among requirements, we compute and analyze the semantic information at two levels: locally, for each word contained in a requirement description, but also globally at the statement level. Moreover, we use advances in neural word embedding models in order to enhance the extraction of semantic information.

4.3.1 Word-level similarity computation

Before computing the word-level similarity, the preprocessed natural language requirements need to be prepared for further processing. This task consists in transferring the preprocessed requirement statements from natural language to a machine-readable and analyzable format that work with machine learning algorithms. Hence, words should be transformed into numerical vectors and this is known as text vectorization.

As stated in Chapter 2, the recent surge of interest in deep learning methods over the machine learning field has led to many attempts to change the way text vectorization is done and find better ways to represent text than traditional DSMs. Inspired by the work of Mikolov et al. [85], we use the word2vec model, a two-layer neural network that is used to produce word embeddings (i.e., vectors).

The input of word2vec is a text corpus. Given enough text data and contexts, word2vec can achieve highly accurate semantics of the words appearing in the corpus and establish a word’s association with other words in the semantic space. Moreover, word embedding models have shown to outperform traditional DSMs which are considered as “count” models as they count co-occurrences among words by operating on co-occurrence matrices [16]. The output is a set of vectors, that is, vectors of words are grouped together in a semantic vector space. Hence, by using word2vec as

prediction model, we gain more accurate vector representations of words, compared with traditional DSMs such as Vector Space Model (VSM) and Latent Semantic Analysis (LSA) [34].

Then, we measure the semantic similarity between each pair of the obtained word vectors belonging to two different requirement statements using the cosine similarity. The cosine similarity principle consists in computing the cosine of the angle between two words vectors. The range of the cosine value is between -1 and 1. Similar words vectors have a cosine value close to 1, and close to 0 otherwise, i.e., the smaller the angle, the higher is the similarity. Given two words w_1 and w_2 belonging to two pairs of requirement statements, we denote their semantic similarity as the cosine similarity between their learned word embeddings as shown in Equation 4.1:

$$\text{wordSim}(w_1, w_2) = \text{cosine}(w_1, w_2) \quad (4.1)$$

This is simply the inner product of the two vectors, normalized by their Euclidean norm. The cosine similarity mainly focuses on the direction differences of two vectors instead of their absolute numerical differences.

There are different metrics that can be used to compute the similarity of words. For example, Euclidean distance measures absolute difference in the numerical characteristics among the dimensions of vectors. However, using Euclidean distance might lead to a problem, that is, even if two words are similar, they may be of a large Euclidean distance, especially for the large size of requirements. In this case Euclidean distance is not able to represent the similarity of requirements accurately.

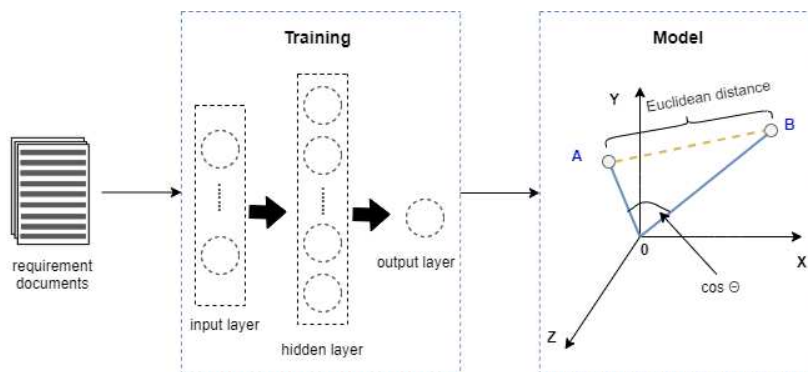


Figure 4.5: Simplified process of obtaining word vectors using word2vec.

As shown in Figure 4.5 the cosine similarity measures the angle between two vectors in the vector space and attaches importance to the differences reflected in the direction rather than the position. For example, if the position of point A is fixed, point B is far away from origin of coordinates along

the original direction. We can see that the angle between the two vectors is not changed, that is the cosine similarity is not changed, while the distance between point A and point B has already been varied, which is the basic difference between cosine similarity and the Euclidean distance.

Hence, using the cosine similarity, even if the two similar requirements are far apart by the Euclidean distance (due to the size of the requirements), chances are they may still be oriented closer together. As a result, cosine similarity helps to properly compute the similarity between words among different sizes of requirements.

4.3.2 Requirement-level similarity computation

After obtaining the word-level similarity, we extend it at the global statement-level. Some approaches capture the meaning of longer pieces of text by taking the means of the individual term vectors [56, 105]. However, means or sums are rather poor ways of describing the distribution of word embeddings across a semantic space. It would be desirable to capture more properties of the two texts, especially with respect to the semantics of words that do or do not match.

We overcome the above-mentioned limitations by deriving the statement-level similarity from the word-level similarity based on two characteristics: the distribution of words in each requirement statement; and the specificity of each word in the requirements document. To do that, we got inspiration from the work of Mihalcea et al. [83], to derive the statement-level semantic similarity from the word-level semantic similarity. Hence, we used the Mihalcea's scoring function for text similarity computation to compute the similarity of each pair of requirement statement (see Equation 4.4).

First, we identify for each word w_1 in the text requirement R_1 , the word w_2 in the text requirement R_2 that have the highest semantic similarity $maxSim(w_1, R_2)$ (Equation 4.2), based on the word-to-word semantic similarity $wordSim(w_1, w_2)$ using word2vec. Next, the same process is applied to determine the most similar word in R_1 starting with words in R_2 as indicated by the example in Figure 4.6.

$$maxSim(w_1, R_2) = \max_{w_2 \in R_2} wordSim(w_1, w_2) \quad (4.2)$$

In addition to the similarity of words, we also take into account the specificity of words using the *Inverse Document Frequency* (*idf*). The specificity of a word w , $idf(w)$ (Equation 4.3) has been defined as the log of the total number of documents in the corpus divided by the total number of documents including that word [62].

$$idf(w) = \log(\text{Total number of documents} / \text{Number of documents with word } w \text{ in it}) \quad (4.3)$$

The word similarities are then weighted with the corresponding word specificity using the *Inverse Document Frequency* (*idf*) weighting technique to capture the specificity of a word. In a nutshell, this technique aims to measure how much a word contributes to the relevance of two texts. The weighted word similarities are then summed up and normalized with the length of each text segment. The resulting similarity scores are combined using a simple average and thus, the semantic similarity of two requirements R_1 and R_2 is computed as follows:

$$Sim(R_1, R_2) = \frac{1}{2} \times \left(\frac{\sum_{w \in R_1} \max Sim(w, R_2) \times idf(w)}{\sum_{w \in R_1} idf(w)} + \frac{\sum_{w \in R_2} \max Sim(w, R_1) \times idf(w)}{\sum_{w \in R_2} idf(w)} \right) \quad (4.4)$$

The achieved similarity score of each pair of requirements is a potentially good indicator of the semantic similarity of two input texts as it combines both the semantic information of word-to-word similarity based on word2vec and the word specificity in a document.

		$w_1^{R_2}$	$w_2^{R_2}$	$w_3^{R_2}$	$w_4^{R_2}$	$w_5^{R_2}$	
	w^{R_1} / w^{R_2}	customer	make	transaction	bank	system	← Preprocessed R_2
$w_1^{R_1}$	operator	0.25	0.17	0.44	0.41	0.34	← Word similarity between « operator » and each word of R_2
		$\max Sim(\text{operator}, R_2) = 0.44$					
		$w_1^{R_1}$	$w_2^{R_1}$	$w_3^{R_1}$	$w_4^{R_1}$	$w_5^{R_1}$	
	w^{R_2} / w^{R_1}	operator	do	cash	operation	machine	← Preprocessed R_1
$w_1^{R_2}$	customer	0.32	0.05	0.12	0.21	0.15	← Word similarity between « customer » and each word of R_1
		$\max Sim(\text{customer}, R_1) = 0.32$					

Figure 4.6: An example for computing $\max Sim(w, R)$

The resulting requirement similarity has a value between 0 and 1, with a value of 1 indicating identical requirement text segments, and a value of 0 indicating no semantic overlap between the two segments. Ultimately, given a document containing N requirements, the output of the similarity computation module is a $N \times N$ semantic similarity matrix that contains the semantic similarity score of each pair of requirements.

4.4 Requirements clustering

In order to generate clusters of similar requirements, we fed the obtained requirements similarity matrix into a clustering algorithm. For this, we use the Hierarchical Agglomerative Clustering algorithm (HAC) [117] as it does not require us to pre-specify the number of clusters in advance which helps to reduce the manual intervention.

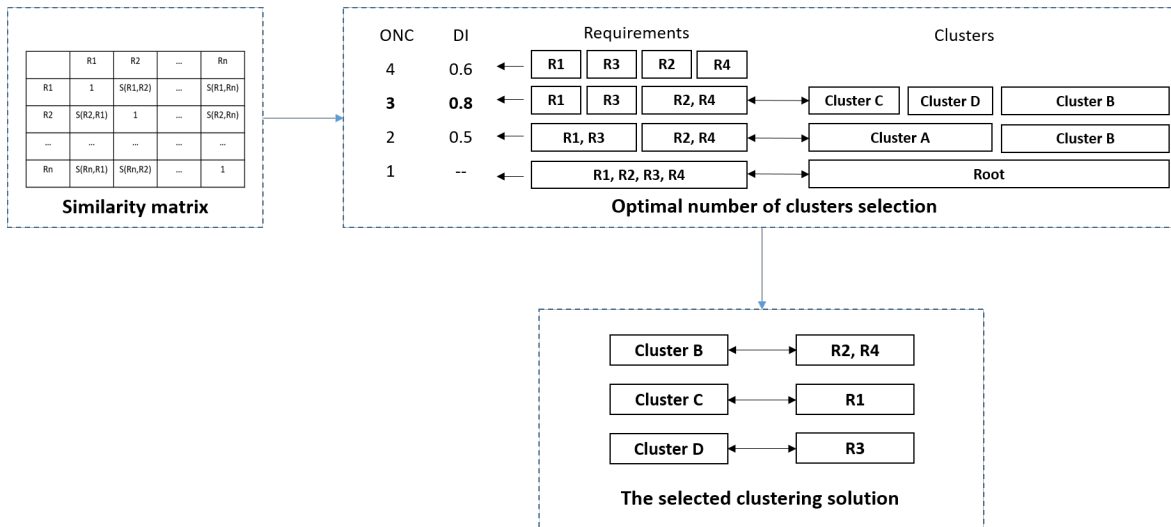
The HAC algorithm works in a bottom-up manner, each requirement statement is initially considered as a single-element cluster (leaf). At each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster (node). This procedure is iterated until all requirements are member of just one single big cluster, resulting in a hierarchical clustering tree.

However, identifying the optimal number of clusters is not a trivial task. It might be subjective as it can heavily rely on the analyst's knowledge. In order to automate this task, we implement an operation that identifies automatically the best number of clusters based on the inconsistency threshold coefficient. More precisely, we compute an inconsistency value between a cluster node and all its descendants in the hierarchical clustering tree. If this value is less than or equal than an inconsistency threshold, then all its leaf descendants belong to the same cluster of requirements. Hence, defining an appropriate inconsistency threshold plays a crucial role in clusters extraction by the HAC algorithm, since the inconsistency threshold makes a huge difference regarding the number of clusters and the accuracy of the extraction process.

In clustering theory, internal validity indices are utilized to evaluate the clustering results when the ground truth of clusters is unknown. Thus, in order to achieve an optimal clustering solution, we apply an internal validity index to estimate the accuracy of the extracted clusters by the clustering algorithm in terms of different candidate number of clusters.

We use Dunn index [37] as internal validity index to evaluate the goodness of the extracted clusters. For a given set of clusters extracted by the clustering algorithm, a higher Dunn index indicates better clustering solution. Therefore, we compute the Dunn index value each time, while varying the candidate cluster numbers. A higher Dunn index indicates better clustering solution. Consequently, to estimate the optimal number of clusters, we select the candidate cluster number for which we have a higher Dunn index.

We explain the process of identifying the optimal clustering solution through the example in Figure 4.7 using four requirements (R1-R4). First, the similarity matrix of requirements is fed into the HAC algorithm to cluster similar requirements. And then, the optimal number of clusters corresponding to the optimal clustering solution is achieved (i.e., $ONC = 3$), since the value of the corresponding Dunn index is the largest one (i.e., $DI = 0.8$). After clustering, we obtain three clusters (i.e., clusters



$S(R_m, R_n)$: similarity value of Requirement m and Requirement n ;
 ONC: optimal number of clusters;
 DI: Dunn index.

Figure 4.7: A simplified example of clusters identification.

B, C, D). The resulted clusters B, C and D represent a set of sub-systems of the software system, that will be mapped into a package break-down model denoting the preliminary architecture design of the target system.

4.5 Labelling

The result of applying the clustering algorithm is a set clusters that group the semantically similar requirements. These clusters present the set of sub-systems describing the target system's decomposition.

In terms of architecture design, the resulted clusters represent the architecture packages. These packages give insights on a possible decomposition of the complex system and can be refined afterwards to provide engineers with further details about the features expected by the target system. However, without a name that summarizes what a cluster (i.e., package) is about, it would be difficult distinguish the clusters from each other. Hence, it is crucial to assign a label for each cluster that describes what each architecture package embeds in terms of functionalities or characteristics. However, assigning the adequate name to each cluster is not a trivial task and it requires a considerable human intervention. One approach to automate this task consists in extracting the most common representative key words within each cluster.

For this, we use *Gensim library* for text summarization¹. *Gensim* is a free Python library designed

¹https://radimrehurek.com/gensim/auto_examples/tutorials/run_summarization.html

to automatically extract semantic topics from documents. This summarizer is based on the ranks of text sentences using a variation of the TextRank algorithm proposed by Mihalcea et al. [82] and it summarizes a given text, by extracting one or more important sentences from the text. In a similar way, it can also extract keywords, i.e., the set of terms that best describe a document. Thus, we label each generated cluster using the Gensim's summarizer by extracting the best representative keywords that are embedded in the cluster.

As a result, we obtain a set of labeled clusters of requirements that describe the architecture design packages as well as their content. Indeed, the obtained architecture packages help engineers to better understand the composition of the complex system at a first level of abstraction. Then, by applying further analyses of the requirement statements within each cluster, we can refine the packages with further degrees of details in order to generate the preliminary use case model describing the target system.

4.6 Conclusion

In this chapter, we presented the requirements semantic clustering approach that we proposed in order to decompose the target system as a first step towards preliminary architecture design generation.

The proposed clustering approach is based on a semantic similarity computation module that takes into account both word-level and requirement-level similarity among requirement statements. First, we used advances in neural word embedding models particularly, we used the word2vec model as prediction model to generate word vectors. We used the cosine similarity to compute the semantic similarity between each pair of word vectors of each pair of requirements. Second, we derive the requirement-level similarity from the word-level similarity using the Mihalcea's scoring formula for text similarity. The output is a requirement similarity matrix that contains the semantic similarity score between each pair of requirements. Afterwards, we fed the generated matrix into the Hierarchical Agglomerative Clustering algorithm (HAC) to generate clusters of requirements. Finally, the identified clusters are labelled in order to give insights on a possible candidate architecture packages decomposing the complex system.

In the next chapter, we describe the solution that we proposed to automatically generate preliminary architecture design models describing the system's decomposition.

Chapter 5

Automatic generation of the preliminary architecture design models

In the previous chapter, we presented our first contribution that consists in decomposing the system into a set of sub-systems based on the semantic similarity of early requirements.

In this chapter, we present our second contribution that consists of a model extractor that automatically generates the preliminary architecture design denoting the decomposition of the complex system. The proposed model extractor uses the identified clusters of requirements as a starting point towards generating UML package of use-cases model and thus, it helps to provide a holistic view of the system's decomposition at early stages.

In what follows, we detail the approach used by the proposed model extractor to automatically generate preliminary architecture design models from early requirements.

5.1 Linking requirements to models

The process of software development is the process of modelling a real world problem and transforming it into a number of refined models, ending with the executable code. MBSE methods promise to support software systems developments by fostering a holistic view of design and empowering high quality and maintainable software architecture on the basis of models. Such methods help to enhance the realization of reliable and efficient systems. Indeed, models play a crucial role in the description and the abstraction of the systems to be developed and help to increase their comprehen-

sibility from the early stages of development. Moreover, design models help engineers understand the features expected by the future system and reduce errors especially in the context of complex systems.

However, the integration of model-based methods in the software engineering process, is facing many challenges [67]. Particularly, linking early requirements to models is not a trivial task especially with the increasing complexity of the developed systems. In what follows, we present the challenges raised by requirements engineering and we highlight the importance of models in mitigating them.

5.1.1 Requirements engineering challenges

The purpose of the requirements engineering activities in a software project is to describe precisely what to build. Although it seems like a simple task, requirements are often considered to be the biggest software engineering challenge [42]. A great challenge that continues to confront software engineering at early stages, is how to proceed easily and efficiently from a set of system requirements to a design that will satisfy those requirements.

One problem in requirements engineering is the variability of natural language in which the requirements are expressed. Moreover, in modern software systems, the number of requirements is exponentially increasing due to the increasing of the number of stakeholders involved in the process. This exponential growth in the number of requirements hinders the efficient management and the understanding of the gathered requirements and consequently, it hinders the construction of high-quality systems.

Additionally, with the shift to Agile development, requirements are continuously changing, which makes it difficult to capture all requirements for a non-trivial system before development starts [69]. It is therefore critical to keep track of the requirements throughout the project.

Herein lies the importance of adopting models as they ensure having a clear crystal view of the expectation and scope of the system to be designed [22].

In fact, adopting models helps to represent and communicate what is important among stakeholders, keep track of the gathered requirement throughout the project, and helps developers deal with the complexity of the problem being investigated or the solution being developed.

In the previous chapter, we presented a solution that helps to reduce the system's complexity by enabling a semantic clustering of early requirements requirements. In this chapter, we provide a solution that automatically generates the preliminary architecture design denoting the system's decomposition throughout the project.

Eventually, each sub-system covering a set of requirements could be expressed in a semi-formal

manner using a UML package including a use case model. Such representation helps to provide an abstraction of the system's decomposition, and a clear view on its big picture and its expected features.

5.1.2 Use case modeling

The Unified Modeling Language (UML) [21] has been employed as a visual language that supports requirements engineering. It has been widely used to specify the features of software systems, and to reduce the ambiguity between the requirement specifications and the design, on the basis of models.

Among these models, UML use case models are typically developed in the early stage of development. They have been widely used to capture the requirements from the user's point of view. In addition, they are easy to understand and provide an excellent way for communicating as they provide a semi-formal framework for modeling (mainly functional) requirements [3]. A use case model is a model of how different types of users interact with the system to solve a problem. As such, it describes the goals of the users, the interactions between the users and the system, and the required behavior of the system in satisfying these goals. A use case model consists of a set of use case descriptions in text, each describing one use case. Each use case description specifies a required functional service that the system is expected to provide for certain kinds of users called actors. It describes a use case in terms of a pattern of interactions between the actors and the system.

An actor of a use-case can be any entity external to the system. It interacts with the system by calling a system operation to request a service of the system. The system may also require services from actors to carry out a requested service.

Associations are used to describe the relationships between actors and the use cases they participate in. This relationship is commonly known as a "communicates-association".

Therefore, a use case model specifies the systems's required functional services, the users of these services, and the dependency relationships among these services. A library system, for example, has use cases to, "Borrow a copy", "Make a Reservation" and "Validate User Identification" for the actor called "User". Both "Borrow a Copy" and "Make a Reservation" includes "Validate User Identification".

The use case model may contain packages that are used to structure and decompose the model to simplify analysis, communications, navigation, development, maintenance and planning. In large scale systems, use-case packages can be employed to further structure the use-case model. In fact, partitioning a use-case model into use-case packages can serve in agile development by reflecting the order, configuration, or delivery units in the finished system thus supporting iteration planning.

In addition, employing use-case packages supports the « divide-to-conquer » strategy as they enable the representation of the decomposition of the system into smaller sub-systems to reduce the complexity.

Use-cases are further specified by a number of use case scenarios (also referred to as use case instances). These scenarios, which describe the interaction between a system and its actors, can be described using UML sequence models and activity models [21]. Typically, for each use-case in a use case model, there is also a corresponding use-case realization in a design model [20]. As shown in Figure 5.1, a use-case realization is a description of how different design elements collaborate to solve a specific use-case [69]. The main purpose of a use-case realization is to provide a bridge between requirements modeled as a use-case and a systems design (i.e. traceability). Use-case realizations are often described using UML Sequence or Collaboration models.

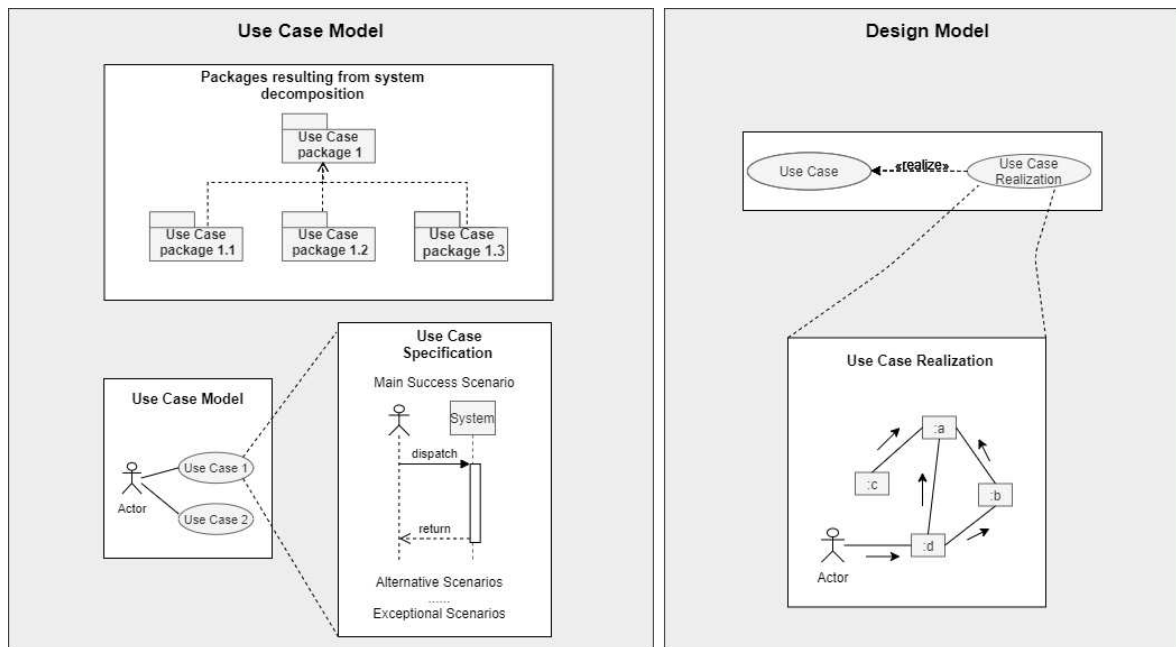


Figure 5.1: Use case model and design model.

Use case models play a broader role in describing systems specifications and they serve as a unifying thread throughout the system development. They are used as the primary specification of the functional requirements for the system, as the basis for analysis and design, as an input to iteration planning, as the basis of defining test cases and as the basis for user documentation and architecture system validation [59]. We believe that extracting UML use case models from early requirements helps to derive architecture design since a use case model could serve as an intermediate model towards analysis models generation (i.e., class model, sequence model, etc...).

In this chapter, we propose a model extractor that automatically derives the preliminary archi-

texture design models of the complex system based on early requirements. The proposed model extractor takes a starting point the identified semantic clusters of requirements and generates as output a UML use case model including packages of use-cases denoting the resulted sub-systems. In what follows, we give an overview of the approach used by the proposed model extractor.

5.2 Overview of the model extractor approach

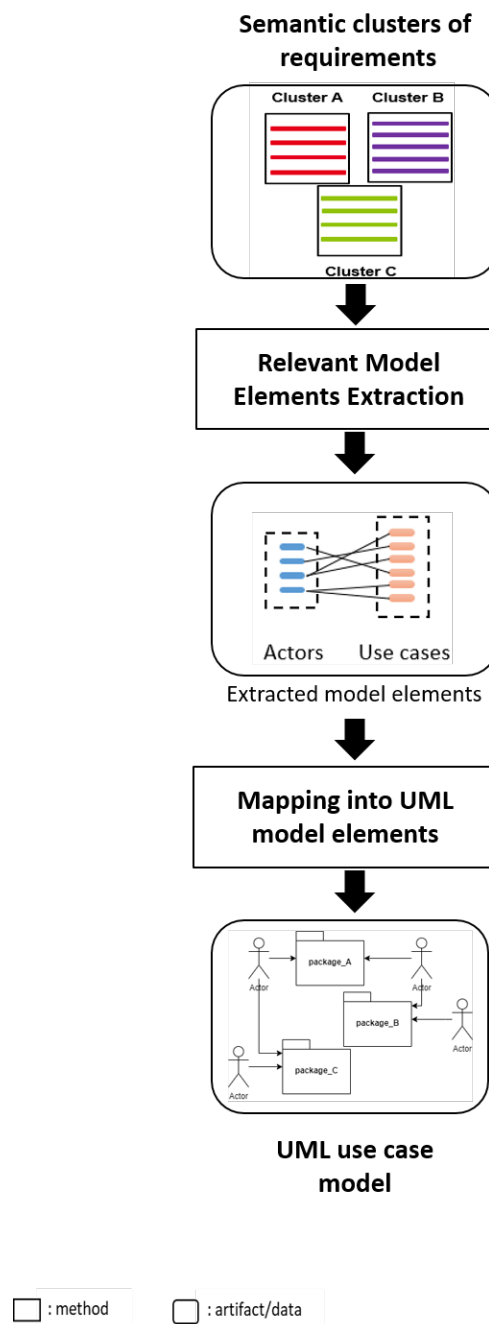


Figure 5.2: Overview of the model extractor approach.

In large scale systems, manually building architecture design model is a laborious, time-consuming and error-prone task. Several approaches were proposed in literature to assist engineers with this task, whereby candidate design model elements are automatically/semi-automatically extracted using Natural Language Processing (NLP) rules. Despite the existing work on design models extraction, important facets remain under-explored: (1) the existing extraction rules do not adequately extract relevant elements, i.e, the extracted elements lack accuracy; and (2) the rules developed by the information retrieval community for information extraction remains unutilized for building UML design models.

Motivated by addressing the above limitations, we developed a model extractor that automates the generation of preliminary architecture models representing the system's decomposition. It takes as input the identified clusters of similar requirements describing the system's decomposition (see previous chapter) and generates as output the UML use case model describing the system as well as its sub-systems.

As shown in Figure 5.2, the model extractor approach is based on two pillars: First, we extract from each cluster the relevant elements that are needed to build the use case model using NLP techniques. Then, we implement a mapping operation that maps the extracted relevant elements into their corresponding ones in the UML use case model. The generated UML use case model encompasses use case packages representing the identified semantic clusters and thus, representing the system's decomposition.

In what follows, we describe in details the NLP techniques that we used to extract relevant model elements as well as the mapping algorithm that we implemented to map the identified model elements into their corresponding ones in the UML use case model.

5.2.1 Extracting relevant use case model elements

Building a design model is an important step for transitioning from informal requirements expressed in natural language to precise and analyzable specifications [116] expressed in semi-formal language. In order to meet this goal, we propose an NLP based module to extract the relevant model elements that are needed to construct the use case model describing the target system.

As shown in Figure 5.3, the proposed NLP module takes as input a cluster of requirements written in natural language and generates as output the relevant use case model elements. For each requirement statement, we initially perform the following operations: tokenization, punctuation and stop words removal (see Section 4.2 in Chapter 4) using the *NLTK* library¹. These operations help to clean the input text and to eliminate insignificant words for our task. Then, we apply pos-tagging to

¹<https://www.nltk.org/>

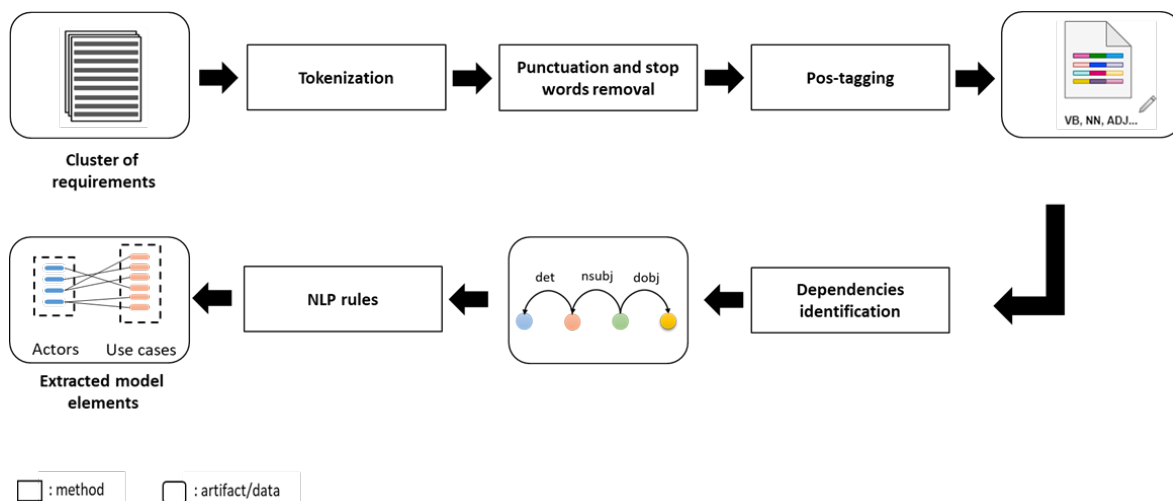


Figure 5.3: Extracting relevant use case model elements.

perform the structure parsing of the sentence by inferring the structural units of the requirement statement (see Section 4.2.3 in Chapter 4). In our work, the units of interest are noun phrases and verbs. A noun phrase (NP) is a unit that can be the subject or the object of a verb. A verb (VB) appears in a verb phrase (VP) alongside any direct or indirect objects, but not the subject. In addition to the sentence structure parsing, dependencies identification is also required. In fact, It aims at identifying grammatical dependencies between the individual words in a sentence. In contrast to phrase structure parsing, which identifies the structural constituents of a sentence, dependency parsing identifies the functional constituents, e.g., the subject and the object. Hence, for each requirement statement, we perform the dependency parsing using the *Spacy* library².

For example, the top part of the graph of Figure 5.4 shows the output of dependency parsing over requirements statement *"The user can modify his profile."*. An example typed dependency here is *nsubj(modify, user)*, stating that *"user"* is the subject of the verb *"modify"*. The output of the dependency parsing is represented as a directed acyclic graph, with labeled (typed) dependency relations between words as shown in Figure 5.4.

The final step of the proposed module consists in applying NLP heuristic rules in order to extract relevant model elements. The implemented rules basically rely on both structure parsing and dependencies identification to infer the relevant elements. Moreover, depending on the of the input requirement statement that can be either user stories or expressed in plain text, we implement two categories of NLP heuristics. Indeed, this makes our model extractor flexible to supports different templates and structures of natural language requirements.

In the next subsections, we detail the proposed NLP heuristics that we used to derive preliminary

²<https://spacy.io/>

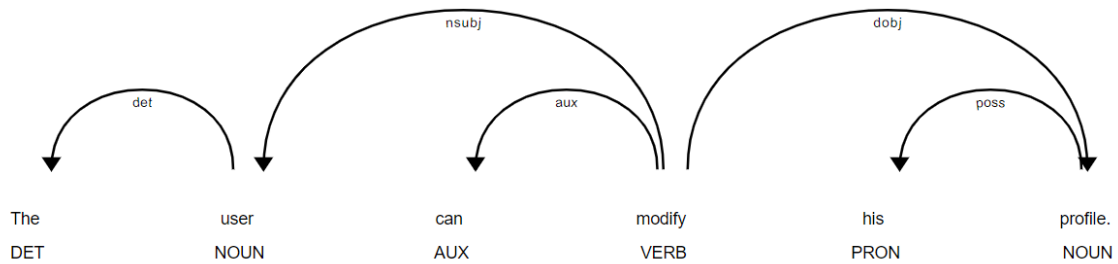


Figure 5.4: Example of the result of the dependency parsing using *Spacy*.

UML design models from system requirements. As our goal is to generate UML use case models, the elements that we aim to extract via the proposed NLP rules are actors, use-cases, and the association relationships between them. We detail the NLP rules that we implemented to derive these elements from both user stories and requirements written in plain text.

The used NLP rules to extract relevant use case model elements from requirements expressed in plain text

The most basic NLP heuristic that we proposed specifies that each (1) noun in the requirement statement denotes an entity that could be a candidate actor or a system in interaction with the actor and, (2) the main verb that occurs between two entities is more likely to be a relationship. For verbs, we exclude modal verbs such as "can", "shall", "must", etc. This prompts us to define the first entity and relationship heuristic as follows:

R1: "Every noun is a potential entity."

R2: "The verb that occurs between two entities is a potential relationship."

Example: Consider the requirement statement "An administrator can manage accounts." that comprises two nouns ("administrator" and "account"), and one verb ("manage") when we exclude the modal verb "can". Rule *R1* specifies to create two entities "administrator" and "account" and rule *R2* originates a relationship between these entities named as the verb:

manage(administrator, account) (R2)

To form relationships between entities, a sentence should contain three elements: the subject, the object and the verb (phrase) linking the previous two. The subject is certainly essential: in an active sentence, for instance, the subject represents the *actor* element in a use case model as it is the initiator of the main action performed. For actors, we ignore every proper noun such as (Location name, Person name, etc.). The verb phrase which contains the verb and the object of a sentence is potentially a candidate use case. For use-cases, we exclude verb phrases in which the verb is

included in the following list: *[include, involve, consist of, contain]*. The identified actor and use-case in a sentence are linked by an association relationship. Therefore, we define the following rules:

R3: "If a noun is a subject then, it is an actor."

R4: "The verb phrase linking the subject and the object is a use-case."

R5: "Each actor and use case are linked by an association relationship."

Example: Considering the same requirement example above "An administrator can manage accounts.", "administrator" is a noun that represents the subject of the sentence. Rule *R3* implies that "administrator" is the actor. The subject and the object are linked via the verb "manage" (if we exclude "can"). Hence, *R4* denotes that the verb phrase (VP) "manage accounts" denotes the use-case and rule *R5* originates an association between the identified actor and use-case.

Most often, the actor can be a compound noun, that is, a noun composed of multiple words. The compound noun can be sequences of nouns or adjectives followed by one or more nouns. To accurately identify the actor element, we consider the whole compound noun as an entity. If a sequence of nouns or adjectives followed by one or more nouns exists and the last noun is not included within the following set of words: $S = [number, no, code, date, type, volume, birth, id, address, name]$, and the compound noun represents a subject, then it may be an actor. Consequently, we define the following NLP rule:

R6: "If a compound noun is a subject and the last noun in that compound noun is not in «S» then, noun compounds are taken together to form an actor."

Example: Let us consider the following requirement statement "A newly registered user can change his password." The sentence comprises a compound noun "a newly registered user" which is the subject. By applying *R6*, "newly registered user" represents an actor.

The used NLP rules to extract relevant use case model elements from user stories

After being introduced in the early 2000s, Agile software development methods such as Scrum, have become widespread in the industry, even surpassing the "classical" waterfall method according to some surveys [38, 109]. Among Agile requirement engineering practices, user stories are widely adopted [102], involving potential stakeholders in the requirement elicitation process by writing their needs in natural language [64].

A user story is a requirement expressed from the perspective of an end-user perspective. It is a semi-structured natural language description of requirements. The structure of user stories follows a compact template that describes the type of user, what they want and (optionally) why [113]. Although many different templates exist to express user stories, 70% of practitioners use the template [74]: "As

a « type of user », I want « goal », [so that « some reason »]”. Following these templates, user stories can capture three distinct aspects of a requirement:

- *Who* wants the functionality.
- *What* functionality the end users or stakeholders want the system to provide.
- *Why* the end users and stakeholders need this functionality (optional).

User stories have been identified as being effective at developing the “right software” since they require a detailed functional decomposition and increase the perceived productivity [74]. The first objective in the Scrum process is to define the product backlog, that is, a list of elicited and prioritized requirements that should be implemented. Then, Scrum processes in sprints. Each sprint contains the tasks to achieve by the developer teams.

The backlog grooming is mainly a manual task, which may become time-consuming and error-prone as the amount of stories grows [17]. This growth in number tends to dilute the technical knowledge across the whole backlog [53], such that visual models may be required to provide a clear view of the target system and to keep stakeholders involved in requirement engineering tasks [53].

However, the adoption of models in Agile methodologies has been hard to perform due to the lack of powerful automation tools as well as the focus of teams on the implementation rather than analysis or documentation [72]. In order to overcome these challenges, it’s critical to provide a bridge between user stories and preliminary design models since it helps stakeholders gain a better understanding of the target system as the system evolves during development.

Since user stories are concise statements about the functionality of a system, not all NLP rules that we proposed in the previous subsection are equally relevant. In fact, user stories are written in a semi-controlled natural language that relatively facilitates the identification of the relevant model elements, thereby making the previous NLP rules poorly relevant.

Accordingly, our model extractor accepts user stories that use the indicators as identified by Wautelet [113]: “As / As a(n)” for the role, “I want (to) / I can / I am able / I would like” for the means, and “so that” (optional) for the ends part. Otherwise, syntactically invalid user stories are not processed.

“R7: “Each first noun, noun plural or compound noun is an actor.”

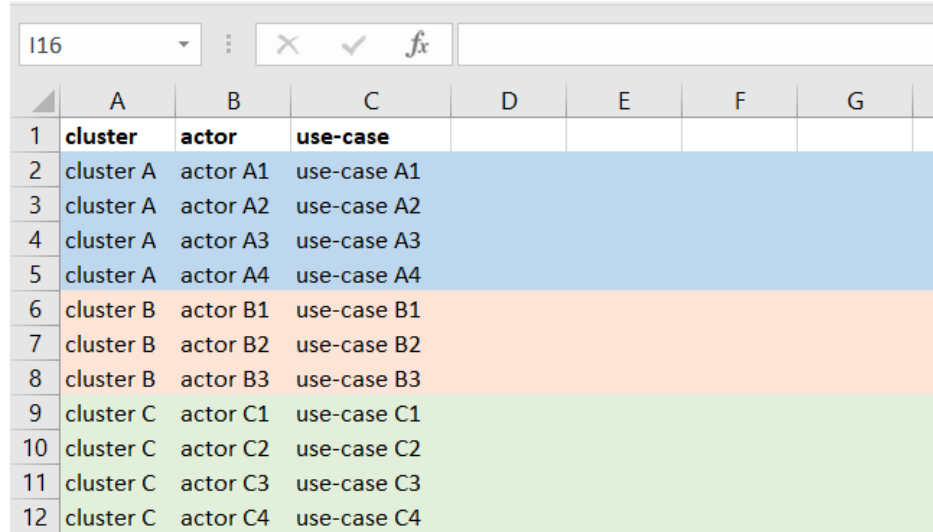
“R8: The use-case is the verb phrase that exists between the patterns: «I’m able to» / «I want»/«I would like to» and «so that/so» if it exists.”

The association is the relationship linking the actor and the use case which makes R3 from the previous subsection relevant.

Example: Let us consider the following user story “As a visitor, I want to register in the event.” By applying *R7*, “visitor” specifies the actor since it is the first noun in the sentence. “register in the event” is the verb phrase that exists after the pattern “I want to” consequently, it represents the use case *R8*.

In summary, depending on the writing style of the input requirement statement, our model extractor performs either the proposed NLP rules for requirements expressed in plain text or those that we proposed for user stories.

The output of these rules is the set of relevant model elements that are needed to construct the UML use case model. The resulted model elements are stored in a *CSV* file for further processing. In this file, we associate the extracted model elements with their corresponding cluster’s name. Figure 5.5 shows an overview of the output of this step. For each cluster, we specify the set of use case model elements that it contains, i.e, the actor and the use-case elements. The actor and the use-case that are in the same row are linked by an association relationship. For example, in the *cluster A*, *actor A1* and *use-case A1* are linked by an association relationship.



	A	B	C	D	E	F	G
1	cluster	actor	use-case				
2	cluster A	actor A1	use-case A1				
3	cluster A	actor A2	use-case A2				
4	cluster A	actor A3	use-case A3				
5	cluster A	actor A4	use-case A4				
6	cluster B	actor B1	use-case B1				
7	cluster B	actor B2	use-case B2				
8	cluster B	actor B3	use-case B3				
9	cluster C	actor C1	use-case C1				
10	cluster C	actor C2	use-case C2				
11	cluster C	actor C3	use-case C3				
12	cluster C	actor C4	use-case C4				

Figure 5.5: Example of the structure of the resulted CSV file grouping the use case model elements.

The resulted file grouping the extracted model elements serves as input for the mapping algorithm. In what follows we explain in details how the proposed mapping algorithm proceeds in order to generate the UML package break-down model including the UML use case model from the extracted model elements.

5.2.2 Mapping into preliminary UML design models

In order to build a visual representation of the preliminary architecture decomposing the system, we implement a mapping algorithm that maps each identified cluster and each extracted use case model element to their corresponding ones in the UML package break-down model including the UML use case model.

For each identified cluster, the mapping algorithm takes as input the cluster's label, and the extracted model element corresponding to that cluster, i.e., the actor, the use-case and the association relationship linking them. Table 5.1 shows the source and the target elements that are processed by the mapping algorithm.

Table 5.1: Model elements mapping

Source	Target
cluster element	UML package
actor element	UML actor
use case element	UML use case
association relationship element	UML association

First, the clusters are mapped into UML packages in the target UML use case model, providing then a holistic view of the decomposition of the system. Then for each cluster, the extracted actors, use cases and associations are mapped into their corresponding UML elements and they are grouped together in the package representing their reference cluster.

The resulted UML use case model including the packages of use cases is generated in the Eclipse Papyrus modeling environment³. We use Papyrus as it is an open source Model-Based Engineering tool that provides an integrated environment for editing UML models. Moreover, It has notably been used successfully in industrial projects and is the base platform for several industrial modeling tools.

Specifically, we use the UML2 tool SDK for Eclipse Papyrus⁴ to automatically generate UML use case models from the extracted use case model elements. The main goals of this component are the provisions of useful implementation of UML for the support of modeling tool development, as well as common XML schema for the facilitation of the exchange of semantic models. UML2 tool SDK for Eclipse enables the creation of models (and their contents) both programmatically and by using the sample UML editor. Accordingly, we use this plugin to programmatically create UML use case models. We implement a java mapping algorithm that maps the identified clusters and the extracted model element to their corresponding ones in the target UML use case model. Figure 5.6 shows an overview of the implemented mapping algorithm.

³<https://www.eclipse.org/papyrus/>

⁴<https://wiki.eclipse.org/MDT-UML2Tools>

Algorithm 2: Mapping algorithm

```

Data {ModelElementsCSVFile}
begin
1  row ← 0
2  actor ← emptyString()
3  actorSet ← emptyList()
4  packageName ← emptyString()
5  UMLUseCaseModel ← ElementFactory.createModel("useCaseModel")
6  open ModelElementsCSVFile
7  modelElement ← row.split(',')
8  while (!EOF(ModelElementsCSVFile)) do
9      /* modelElement[0] refers to the cluster name in the CSV file */
10     if (!modelElement [0].equalsIgnoreCase(packageName)) then
11         packageName ← modelElement [0]
12         UMLPackageElement ← ElementFactory.createPackage (UMLUseCaseModel, packageName)
13     endif
14     /* modelElement[1] refers to the actor element in the CSV file */
15     if (!modelElement [1].equalsIgnoreCase(actor)) then
16         actor ← modelElement [1]
17         /* if the actor does not exist in the list of actors (actorSet) then create a new UML actor element */
18         if (!actorSet.contains(actor)) then
19             UMLActorElement ← ElementFactory.createActor(UMLUseCaseModel, actor)
20             actorSet.add(actor)
21         endif
22     endif
23     /* modelElement[2] refers to the use case element in the CSV file */
24     UMLUseCaseElement ← ElementFactory.createUseCase(modelElement [2],UMLPackageElement)
25     ElementFactory.createAssociation(UMLActorElement,UMLUseCaseElement,UMLUseCaseModel)
26 endWhile
27 close ModelElementsCSVFile
28 end

```

Figure 5.6: Pseudo-code of the mapping algorithm.

Finally, the output of his mapping algorithm is a UML use case model that contains packages of use-cases decomposing the system. Each package of use-cases encompasses further details on use case model elements that it contains. Indeed, grouping these model elements in packages provides a better readability of the UML use case model, especially for complex systems.

5.3 Conclusion

In this Chapter, we presented our model extractor that automatically generates a holistic view of the system's decomposition based on early natural language requirements. The proposed model extractor takes as input the identified set of requirement clusters and generates as output the corresponding UML use case model including packages of use-cases describing the system's decomposition.

First, we proposed a set of NLP heuristics to extract from each requirement statement within each cluster, the relevant model elements that are needed to build the target UML model. Then, the extracted elements are mapped into their corresponding ones in the UML use case model. The final output is a UML use case model in which each cluster is represented as a UML package of

use-cases.

The generated UML use case model helps to provide the preliminary architecture design since it serves as an intermediate towards analysis models generation (i.e., class model, sequence model, etc.).

Chapter 6

Evaluation of the requirements semantic clustering solution

In this chapter, we demonstrate the applicability and benefits of the proposed requirements semantic clustering solution in real-world case studies. For this purpose, we conduct seven case studies containing different writing styles of natural language requirements to evaluate the proposed approach: three case studies containing user stories and four case studies containing functional requirements written in plain text.

This chapter is organized as follows: Section 6.1 presents the Key Performance Indicators (KPIs) that are investigated in order to evaluate the approach. Section 6.2 describes the conducted case studies. Afterwards, in Section 6.3 we present the results analysis and the evaluation. In Section 6.4, we present the assessment of the KPIs. Then, in Section 6.5, we discuss the raised threats to validity and Finally, Section 6.6 concludes the chapter.

6.1 Key Performance Indicators (KPIs)

In this section, we present the KPIs that we investigated in order to assess the applicability of the proposed semantic clustering solution in practice. The identified KPIs are as follows:

- **KPI1: The accuracy of the clustering solution**

Motivation. For this KPI, we aim at determining the accuracy of the proposed clustering solution in order to assess whether our approach succeeded to identify semantic clusters that reflect the domain functionalities embedded in a given requirements document. For this purpose, we compare our results of the clustering with the ground truth clusters. For each case

study, ground truth requirements clusters are manually clustered by analysts.

Approach. To evaluate this KPI, we assess the proposed clustering solution using two validation criteria as follows:

– **The correctness of the identified semantic clusters:**

This validation criterion aims at verifying whether the identified semantic clusters are close to the ground truth clusters. We match the identified cluster (i.e., their requirement statements) with the ground truth clusters of the case study of interest. For this, we rely on two well-known measures in the Information Retrieval (IR) field. These metrics are *precision*, *recall* [78]. Finally, to put *precision* and *recall* in relation, we use *F-measure* as a harmonic average of both of these measures.

Let True Positive (TP) elements be the similar requirements correctly assigned to the same cluster, False Positive (FP) elements be dissimilar requirements assigned to the same cluster and False Negative (FN) elements be similar requirements incorrectly assigned to different clusters. The evaluation metrics are defined as follows:

$$Precision = TP / (TP + FP)$$

$$Recall = TP / (TP + FN)$$

$$F - measure = 2 * Precision * Recall / (Precision + Recall)$$

– **The clustering gap (C_GAP):**

For this validation criterion, we aim to verify whether the identified number of clusters is close to the ground truth number that is manually identified by analysts. This is recognized as the clustering gap (*C_Gap*). The *C_Gap* compares the identified number of clusters with the ground truth number of clusters of the case study of interest. Hence, this metric is computed by applying the following equation:

$$C_Gap = |number_{identifiedClusters} - number_{groundTruthClusters}|$$

• **KPI2: Applicability of the proposed clustering solution in realistic settings**

Motivation. For this KPI, we aim to establish whether our approach is scalable. Particularly, the goal is to check how well the clustering solution performs when increasing the number of requirements.

Approach. In order to evaluate this KPI, we assess the following validation criterion:

– **The end-to-end execution time of the clustering solution:**

This validation criterion consists in measuring the impact of the number of requirement statements of each case study on the execution time of the clustering solution. Hence, it aims to check whether the proposed clustering solution runs within reasonable time for larger number of requirements in realistic settings.

• **KPI3: Applicability of the clustering solution for different domains**

Motivation. For this KPI, we put emphasis on the applicability of our approach for different domains. If the clustering approach would be sensitive for a certain domain, it lacks generalizability for different domains in practice, and thus, would have a limited applicability.

Approach. In order to address this KPI, we assess the applicability of our approach on seven real-world case studies from different domains and with different requirements writing style that we will detail in the following section.

6.2 Case studies description

In order to assess the generalizability of the proposed clustering approach, we conduct seven real-world case studies from different domains and containing different writing styles of functional requirements.

We conduct three real-world case studies containing user stories for Agile development. The user stories in each case study follow different templates. Moreover, we evaluate our approach on four other case studies from open-access software projects containing functional requirements written in plain text. In what follows, we detail the conducted case studies.

6.2.1 User stories

The conducted real-world case studies of user stories are available in the University of Bath's Institutional Repository (IRDUS1). The three case studies belong to different domains following different templates of user stories and having different number of statements. In the following, we introduce each in more detail:

• **CMS Company:**

This case study involves a company from a mid-sized software company located in the Netherlands with 120 employees and 150 customers. They supplied 34 user stories for a complex

CMS product for large enterprises; those stories represent a snapshot of approximately a year of development in 2011. The user stories follow the template proposed by Cohn [32] defined as follows 'As a [type of user], I want to [some goal] so that [some reason]'.

Despite the smaller size, this case study contains lengthy user stories with non-trivial sentence structuring such as: "As an editor, I want to search on media item titles and terms in the Media Repository in a case insensitive way, so the number of media item results are increased, and I find relevant media items more efficiently".

- **Web Company:**

This case study comes from a Dutch company that creates tailor-made web applications for businesses. The team consists of nine employees who iteratively develop applications in bi-weekly Scrum sprints. The Web Company supplied 84 user stories covering the development of an entire web application focused on interactive story telling that was created in 2014. The user stories in this case study rely on the template: "As a [type of user], I'm able to [some goal] so that [some reason]". However, the structure of the user stories in this case study is more complex than in the CMS Company case study.

- **Archive Space**

Archive Space is an open-source software product created by archivists. The user stories of this project are available online(<http://archivesspace.atlassian.net>). It contains 51 user stories that rely on the template "As a [type of user], I want [some goal]". All user stories in this collection omit the "so that" token. Moreover, many user stories contain unnecessarily capitalized words, compound nouns, and idiosyncratic phrases such as "...either the [creator | source | subject] of an [Accession | Resource | Resource Component]..."

Table 6.1: Characteristics of the case studies of user stories

Case study	Number of user stories	Number of clusters
CMS Company	34	4
Web Company	84	6
Archive Space	51	5

For these case studies, the ground truth clusters is not provided. Hence, they are manually created by our team experts. Table 6.1 shows the characteristics of each case study in terms of number of requirements as well as the ground truth number of clusters.

6.2.2 Functional requirements written in plain text

In addition to the conducted case studies of user stories, we evaluate our approach on four open-access projects containing functional requirements written in plain text in order to assess the generalizability of our clustering solution. The conducted case studies are from different domains and have different number of requirements. In what follows, we describe each in more detail:

- **E-Store System**

E-store is a software that consists of online sales, distribution and marketing of electronics [88]. It contains 62 functional requirement statement.

- **WASP System**

WASP system presents architecture specifications of context-aware mobile telecommunication services [101]. It contains 66 functional requirement. These requirements provide several wishes for functionality in the 3G platform from the point of view of the WASP platform.

- **UUIS system**

The UUIS system - Unified University Inventory System - is used to integrate three faculties' databases providing a web interface that allows user to access and manage the integrated inventory [88]. It guarantee a secure access to the data from outside university at any time during working hours. This case study contains 25 functional requirement.

- **MHC-PM System**

The MHC-PM system is a Mental Health Care Patient Management System [110]. It allows users to manage patient's data, retrieve data, and generate reports. It contains 19 functional requirement statement.

Table 6.2: Characteristics of the case studies of functional requirements written in plain text

Case study	Number of functional requirements	Number of clusters
E-store System	62	20
WASP System	66	14
UUIS System	25	11
MHC-PM System	19	6

The four case studies mentioned above are provided in a Software Requirement Specification (SRS) document. In this document, functional requirements sharing similar functionalities are grouped together. Hence, the SRS documents of these case studies serve as the ground truth for our clustering approach evaluation.

Table 6.2 shows the characteristics of each case study in terms of number of requirements as well as the ground truth number of clusters.

6.3 Results analysis and evaluation

In this section, we present and evaluate the results of applying our approach to the chosen case studies. First, we describe the experimental settings that we considered to apply the approach. Then, we evaluate the results of the application of the clustering approach to both case studies of user stories and functional requirements written in plain text. Namely, we evaluate the accuracy of the semantic clustering and the C_Gap stated in Section 6.1. Moreover, we compare the clustering results of functional requirements written in plain text with a related approach.

In our experimentation, we assume that the requirements in the conducted case studies are unambiguous, understandable and consistent [11].

6.3.1 Experimental settings

In this section, we briefly introduce the key parameters setting that we applied for the semantic clustering approach. For the word-level similarity computation, we were confronted with two limitations in the conducted case study:

1) A word embedding model trained with a large corpus is supposed to be of high quality. However, the chosen case studies are small datasets, which leads to a lack of enough data for training a word embedding model with high quality.

2) Although we can achieve an accurate word similarity with the word2vec model, it still exhibits the limitation that a vector space is not capable of gaining the association of all words we consider, if a word does not exist in the training corpus. That is to say, even if the size of one corpus is large enough, it hardly contains all relevant words for the different case studies, especially for domain-specific words. Hence, the similarity value of a pair of words can not be determined, if one of these words does not exist in the training corpus. The common method to handle out-of-vocabulary words is to assign a random vector for out-of-vocabulary words [66]. Unfortunately, a random vector not only lacks the ability to vectorize a word accurately but also leads to an uncertainty of the results.

In order to overcome these limitations, we use two different word2vec models to compute the word-level similarity. First, we use the pre-trained word2vec model from Mikolov et al. [84] trained on Google News dataset¹ including 100 billion words to mitigate the limitation raised by small training

¹<https://code.google.com/archive/p/word2vec>

datasets. Second, we also train a complementary word2vec model on the requirements of each case study. The first pre-trained model is utilized to gain the general meaning of a word. The complementary model is utilized to handle out-of-vocabulary words. If a word is absent in the main model, then, it probably belongs to domain-specific terms and the vector representation of this word is obtained by applying the complementary model.

6.3.2 Evaluation of the semantic clustering of the user story case studies

In this subsection, we present the results of applying the proposed clustering approach to the conducted user story case studies. Then, we assess the accuracy of the semantic clustering of the identified clusters of user stories using *precision*, *recall* and *F-measure* metrics as stated in Section 6.1.

In Tables 6.3, 6.4 and 6.5, we present examples of the identified semantic clusters and the corresponding ground truth clusters for the three user story case studies. The user stories shown in bold in the identified cluster are irrelevant user stories in that cluster.

Table 6.6 shows the average *precision* and *recall* values of the identified clusters for each user story case study. These values indicate that the user stories of these identified clusters are semantically grouped together to form semantic clusters. Indeed, *precision* values take a high-range (0.80–0.89), *recall* values take a reasonable range (0.76–0.86) and *F-measure* values take a range (0.77–0.87) across different case studies.

In Figure 6.1, we graphically display the average *precision*, *recall* and *F-measure* values of the identified clusters from each case study against the number of user stories. As shown in this figure, the average values for *precision* are relatively close to each other. In addition, the average values for *recall* are relatively close to each other. Consequently, the average *F-measure* values are also close to each other. This represents an indicator that our proposed approach works accurately regardless of the number of user stories.

The second criterion that we use to evaluate our clustering solution is the *C_GAP*. In fact, the accuracy of the clustering solution depends on the choice of the number of clusters. We automated the identification of the optimal number of clusters using Dunn index as stated in Chapter 4. We run the HAC algorithm with different number of clusters for each run and we check Dunn index of the outcome. Thus, we carried out 6 runs for each case study. The best number of clusters is the number of clusters with the largest Dunn index value.

For example, the best clustering result for the "CMS Company" case study was achieved in the fourth run, followed by the fifth, third and second ones. Thus, as shown in Figure 6.2 the optimal

Table 6.3: An example of a semantic cluster identified from the "CMS Company" case study.

Identified Cluster Members	Ground Truth Cluster Members
As a marketer I want to set the title attribute of a link so I can improve the SE ranking of the website.	As a marketer I want to set the title attribute of a link so I can improve the SE ranking of the website.
As a marketer I want to create friendly URLs for my nested product pages so I can improve the SE ranking of the product section of my website.	As a marketer I want to create friendly URLs for my nested product pages so I can improve the SE ranking of the product section of my website.
As a marketer I want to set the rel attribute of external links so I can make sure that SE bots do not affect SE rankings for pages with many external links.	As a marketer I want to set the rel attribute of external links so I can make sure that SE bots do not affect SE rankings for pages with many external links.
As a marketer I want to set canonical tags to individual pages so I can avoid duplicate content easily without having to set permanent redirects and thereby will improve the SE ranking of the website.	As a marketer I want to set canonical tags to individual pages so I can avoid duplicate content easily without having to set permanent redirects and thereby will improve the SE ranking of the website.
As a marketer I want to solve URL conflicts immediately so I avoid not-friendly URLs and thereby will positively influence the overall SE ranking of the website.	As a marketer I want to solve URL conflicts immediately so I avoid not-friendly URLs and thereby will positively influence the overall SE ranking of the website.
As an editor I want to assign page sections to pages using the current page section structure so I can easily and efficiently manage page sections assigned to pages.	As an editor I want to assign page sections to pages using the current page section structure so I can easily and efficiently manage page sections assigned to pages.
As a marketer I want to switch URLs when URL duplication occurs between pages so that I can solve conflicts between pages easily without having to search all pages in the tree.	

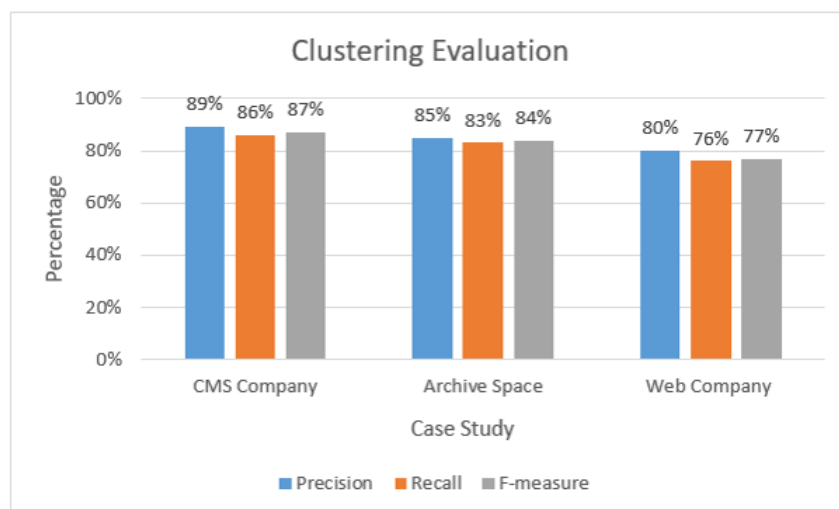


Figure 6.1: Bar graph of the clustering accuracy across user story case studies.

number of clusters for the "CMS Company" case study is four clusters. Moreover, it is possible to graphically verify the conformance of the distribution of the user stories and the optimal number of

Table 6.4: An example of a semantic cluster identified from the "Web Company" case study.

Identified Cluster Members	Ground Truth Cluster Members
As an Administrator I'm able to delete content (which a user added) from a person's profile page.	As an Administrator I'm able to delete content (which a user added) from a person's profile page.
As a User I'm able to delete content (which I added) from a person's profile page so that I remove information that I no longer want to share.	As a User I'm able to delete content (which I added) from a person's profile page so that I remove information that I no longer want to share.
As a User I'm able to edit the content that I added from a person's profile page to update the information.	As a User I'm able to edit the content that I added from a person's profile page to update the information.
As a User I'm able to add a description to a person's profile page.	As a User I'm able to add a description to a person's profile page.
As a User I'm able to add an audio fragment (Soundcloud link) to a person's profile page.	As a User I'm able to add an audio fragment (Soundcloud link) to a person's profile page.
As a User I'm able to add a video (YouTube Vimeo link) to a person's profile page.	As a User I'm able to add a video (YouTube Vimeo link) to a person's profile page.
As a User I'm able to add an image to a person's profile page.	As a User I'm able to add an image to a person's profile page.
As a User I'm able to add text to a person's profile page.	As a User I'm able to add text to a person's profile page.
As a User I'm able to add content to the selected profile.	As a User I'm able to add content to the selected profile.
As a Visitor I'm able to view the added stories (if any) on the profile page so that I can learn more about the person.	As a Visitor I'm able to view the added stories (if any) on the profile page so that I can learn more about the person.
As a Visitor I'm able to view the profile of a particular person so that I can identify that person and add additional content to their profile.	
As a Visitor I'm able to click on a person's profile card so that I can open their profile page.	As a Visitor I'm able to click on a person's profile card so that I can open their profile page.
As an Administrator I'm able to manage people so that I can add edit or delete profiles.	

clusters that we identified based on the dendrograms as shown in Figure 6.3. In this figure, the horizontal red line crosses the hierarchy in two cutting points. Therefore, four semantic clusters are identified.

Table 6.7 shows, for each case study, the number of the identified clusters, the ground truth number of clusters as well as the C_Gap value.

We note that the number of identified clusters is very close to the ground truth number of clusters (that are shown in Table 6.2) of each user story case study. Hence, we obtain a small C_Gap value (between 0 and 1) which represents a good indicator about the accuracy of the proposed approach.

Table 6.5: An example of a semantic cluster identified from the "Archive Space" case study.

Identified Cluster Members	Ground Truth Cluster Members
As an archivist, I want to import EAD files that were exported by Archon.	As an archivist, I want to import EAD files that were exported by Archon.
As an archivist, I want to import Accessions data in CSV.	As an archivist, I want to import Accessions data in CSV.
As an archivist, I want to import Resources from MARCXML records.	As an archivist, I want to import Resources from MARCXML records.
As an archivist, I want to import Agent information from EAC-CPF records.	As an archivist, I want to import Agent information from EAC-CPF records.
As an archivist, I want to export agent records as EAC-CPF.	As an archivist, I want to export agent records as EAC-CPF.
As an archivist, I want to import only Agent and Subject information from MARCXML records.	As an archivist, I want to import only Agent and Subject information from MARCXML records.
As an archivist, I want to import EAD files that were exported by the Archivists' Toolkit.	As an archivist, I want to import EAD files that were exported by the Archivists' Toolkit.
As an archivist, I want to export a description as EAD.	As an archivist, I want to export a description as EAD.
As an archivist, I want to upload an EAD for import from within the frontend application.	As an archivist, I want to upload an EAD for import from within the frontend application.
As an archivist I want to import EAD data.	As an archivist I want to import EAD data.
As an archivist, I want to create and edit Agent records.	

Table 6.6: Accuracy of the generated semantic clusters for the user story case studies.

	CMS Company	Web Company	Archive Space
Average Precision	89%	80%	85%
Average Recall	86%	76%	83%
Average F-measure	87%	77%	84%

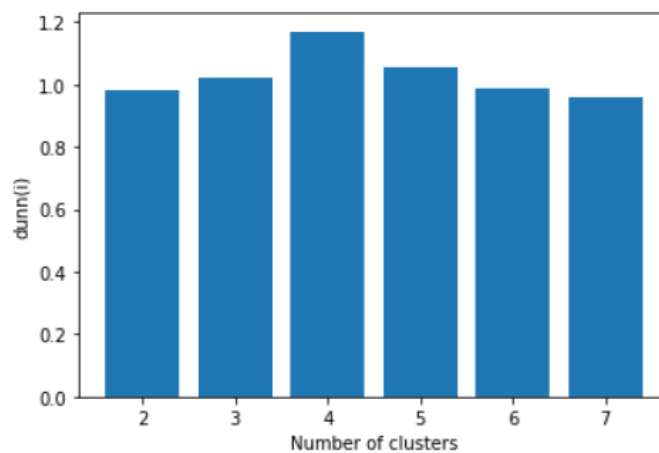


Figure 6.2: Dunn index bar graph of the "CMS Company" case study.

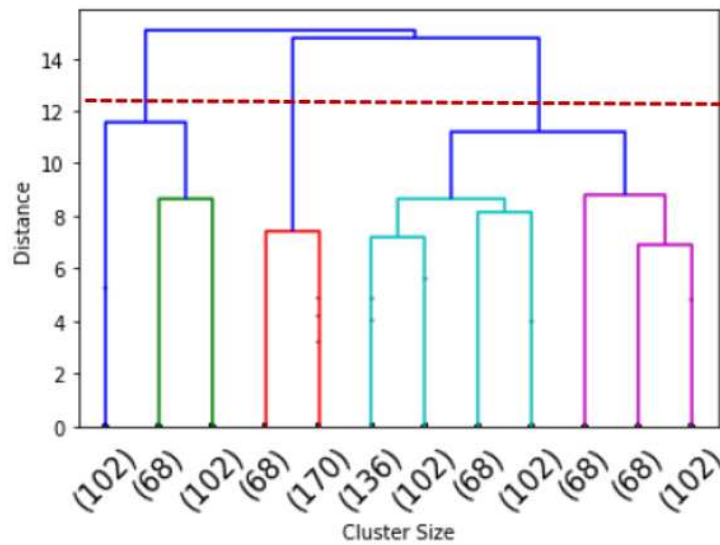


Figure 6.3: Dendrogram of the identified clusters of the "CMS Company" case study.

Table 6.7: Identifying the C_Gap for the user story case studies.

	CMS Company	Web Company	Archive Space
Number of the identified clusters	4	7	4
Number of the ground truth clusters	4	6	5
C_Gap	0	1	1

6.3.3 Evaluation of the semantic clustering of the case studies of functional requirements written in plain text

In this subsection, we present the results of applying the semantic clustering approach to the conducted case studies of functional requirements written in plain text. Then, we assess the accuracy of the semantic clustering of the identified clusters using *precision*, *recall* and *F-measure* metrics presented in Section 6.1.

In Tables 6.8, 6.9, 6.10 and 6.11, we present examples of the identified semantic clusters and the corresponding ground truth clusters for the four case studies. The functional requirements shown in bold in the identified cluster are irrelevant requirements in that cluster.

Table 6.8: An example of a semantic cluster identified from the "E-store system" case study.

Identified Cluster Members	Ground Truth Cluster Members
The system shall allow user to create profile and set his credential.	The system shall allow user to create profile and set his credential.
The system shall authenticate user credentials to view the profile.	The system shall authenticate user credentials to view the profile.
The system shall allow user to update the profile information.	The system shall allow user to update the profile information.
The system shall allow user to register for newsletters and surveys in the profile.	

Table 6.9: An example of a semantic cluster identified from the "WASP system" case study.

Identified Cluster Members	Ground Truth Cluster Members
The WASP platform MUST allow end-users to set an alert on an event.	The WASP platform MUST allow end-users to set an alert on an event.
The WASP platform SHOULD allow the end-user to specify the notification type when setting an alert.	The WASP platform SHOULD allow the end-user to specify the notification type when setting an alert.
The WASP platform MUST maintain a list of events the end-user can be notified about.	The WASP platform MUST maintain a list of events the end-user can be notified about.
The WASP platform SHOULD be able to decide how to notify the user of an alert for which an event was set.	The WASP platform SHOULD be able to decide how to notify the user of an alert for which an event was set.
The WASP platform MUST actively monitor all events.	The WASP platform MUST actively monitor all events.
The WASP platform MUST allow the end-user to remove previously set alerts on events.	The WASP platform MUST allow the end-user to remove previously set alerts on events.
If the user cannot be notified of the event the first time, the WASP platform SHOULD retry to notify the user of the occurrence of the event, until the user has been notified or a specified time-out elapses.	If the user cannot be notified of the event the first time, the WASP platform SHOULD retry to notify the user of the occurrence of the event, until the user has been notified or a specified time-out elapses.
The WASP platform MUST notify the end-user about the occurrence of an event for which an alert was set, as soon as the event occurs.	The WASP platform MUST notify the end-user about the occurrence of an event for which an alert was set, as soon as the event occurs.
It MUST be possible for a WASP application to send out messages to users using the WASP platform.	

In order to evaluate the applicability of our approach on functional requirements written in plain text, we compute the average *precision*, *recall*, *F-measure* and *C_Gap* values of the identified semantic clusters for each case study.

In addition, we compare our results to the work in [96]. In fact, the approach proposed by Salman et al. in [96] closely relates to our work as it proposes a method to cluster functional requirements using the same case studies. Thus, we use the work in [96] as a baseline to assess the identified clusters. Table 6.12 shows the evaluation results of our approach as well as a comparison with the

Table 6.10: An example of a semantic cluster identified from the "UUIS system" case study.

Identified Cluster Members	Ground Truth Cluster Members
Any DA group member or authorised inventory group member asset is owned by the department.	Any DA group member or authorised inventory group member asset is owned by the department.
Any faculty member can add all related departments inventory.	Any faculty member can add all related departments inventory.
Any university group member can add all assets in the inventory.	Any university group member can add all assets in the inventory.
	A bulk entry can be used to add many assets.

Table 6.11: An example of a semantic cluster identified from the "MHC-PM system" case study.

Identified Cluster Members	Ground Truth Cluster Members
Patient should be able to request their own personal.	Patient should be able to request their own personal.
Clinical Staff should be able to look up patient information including appointment history, diagnosis history, prescriptions history.	Clinical Staff should be able to look up patient information including appointment history, diagnosis history, prescriptions history.
Receptionist should be able to search for records of individual patients.	Receptionist should be able to search for records of individual patients.
Nurses going into the field should be able to download record on to their laptop and after upload back the records will modified/updated to the system.	Nurses going into the field should be able to download record on to their laptop and after upload back the records will modified/updated to the system.
System shall generate a list of patient conditions, treatments and the current care provider.	
Within 3 months of death the patient's record will be removed from the MHC-PMS system and be stored in the old data record (none use of data) or archive system.	

baseline [96] in terms of *precision*, *recall*, *F-measure* and *C_GAP*. The best values are displayed in bold.

Precision values take a high-range (74% – 87%), *recall* values take a reasonable range (63% – 75%) and *F-measure* values take a high-range (73% - 78%) across different case studies. In most of these case studies, we achieved better *precision*, *recall* and *F-measure* values compared with the baseline [96]. Figures 6.4, 6.5 and 7.10 show the bar graphs of the *precision*, *recall* and *F-measure* values achieved by our clustering solution compared with the baseline [96].

Table 6.12: Accuracy of the generated semantic clusters for the case studies of functional requirements written in plain text.

		E-store System	WASP System	UUIS System	MHC-PM System
Our approach	Average Precision	83%	87%	74%	84%
	Average Recall	68%	63%	75%	73%
	Average F-measure	75%	73%	74%	78%
	C_Gap	0	2	1	1
The baseline [96]	Average Precision	80%	83%	72%	78%
	Average Recall	61%	54%	60%	57%
	Average F-measure	69%	65%	65%	66%
	C_Gap	1	4	2	0

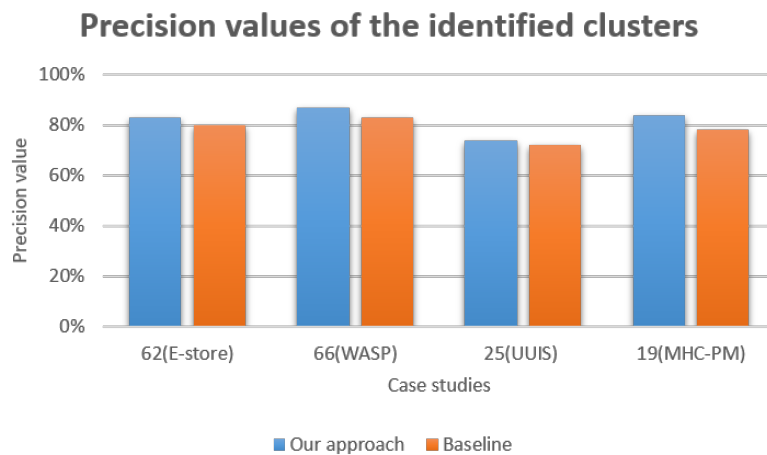


Figure 6.4: Bar graph of the precision values of the identified clusters.

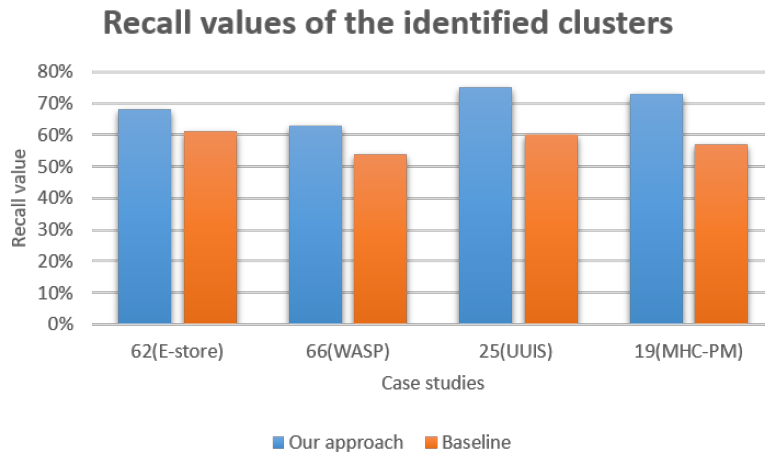


Figure 6.5: Bar graph of the recall values of the identified clusters.

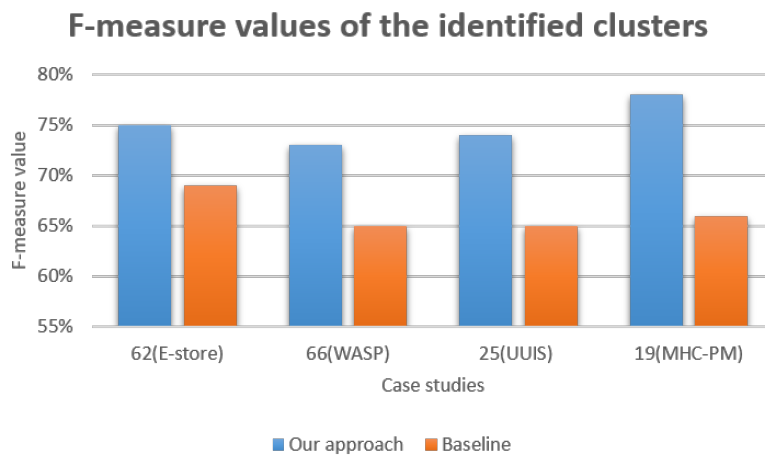


Figure 6.6: Bar graph of the F-measure values of the identified clusters.

For example, for the "MHC-PM System" case study our approach achieves better *precision*, *recall* and *F-measure* values by 6, 16 and 12 percentage points respectively. Thus, the evaluation shows clustering results with relatively high quality with better *precision*, *recall* and *F-measure* values in most case studies compared with the baseline [96].

In addition, the proposed clustering approach achieves closer C_Gap values compared with the work in [96]. Figure 6.7 shows an illustration of the number of clusters identified by both the baseline and our approach, against the number of requirements for each case study of functional requirements. Clearly, Figure 6.7 shows that the identified number of clusters is more closer to the ground truth compared with the work in [96]. In addition the C_GAP value varies between 0 and 2 for the four case studies.

Consequently, this indicates that our approach succeeded to achieve an accurate identification of

the number of clusters regardless the number of requirements of the case study of interest.

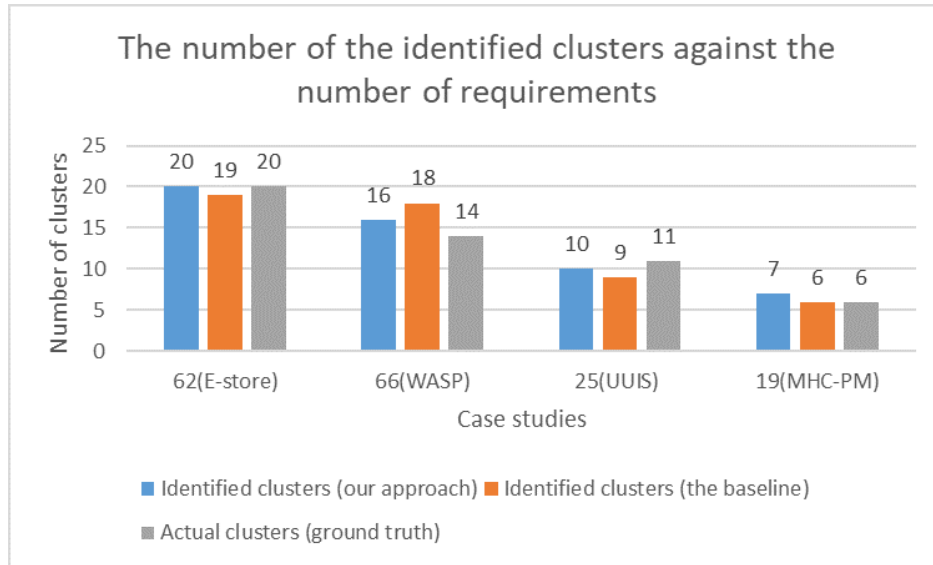


Figure 6.7: The number of the identified clusters against the number of the requirements for each case study of functional requirements.

6.4 Assessing KPIs

For KPI1, firstly, our approach provides relatively high *precision*, *recall* and *F-measure* values for the identified semantic cluster across different cases studies containing different writing styles of requirement statements (i.e., both user story case studies and case studies of functional requirements). As shown in Tables [6.6](#) and [6.12](#), the average *F-measure* values take a high-range (73% - 87%) across the seven conducted case studies.

Moreover, the *C_GAP* is close to the ground truth number of clusters and in some cases it is the same as the ground truth as shown in Tables [6.7](#) and [6.12](#). Particularly, by comparing our clustering solution with the related approach proposed in [\[96\]](#), we observe that our approach provides better accuracy results in terms of *precision*, *recall* and *F-measure* (see Table [6.12](#)). The *C_Gap* values are also relatively accurate compared with the *C_Gap* values identified in the baseline. In addition our approach succeeded to achieve an accurate identification of the number of clusters regardless the number of requirements of the case study of interest (see Figure [6.7](#)). Accordingly, our approach improves the semantic clusters extraction for the case studies of functional requirements compared with the baseline [\[96\]](#).

At the light of these results, it is noticeable that our semantic clustering solution succeeded to achieve relatively accurate results that can be applicable regardless the writing style of the input natural language requirements. This assessment is based on the average *precision*, *recall*, *F-measure*

and C_Gap values and their statistics shown in Tables 6.6 and 6.12 as well as the comparison of our clustering results with a related approach [96].

For KPI2, we assess the applicability of the proposed clustering solution by measuring the end-to-end execution time across different case studies. The experiments were carried out on a laptop with a 2.10 Ghz Intel (R) Core (TM) i7-4600U CPU and a 8GB of memory. In Tables 6.13 and 6.14, we present the impact of the number of requirements on the end-to-end execution time across the seven chosen case studies. As shown in the Tables 6.13 and 6.14, the semantic clustering approach takes few seconds to few minutes (between 20s and 1m7s) for the seven conducted case studies. Moreover, both Figures 6.8 and 6.9 show a linear growth trend for the impact of the number of requirements on the execution time. Given such linear relation and the fact that the end-to-end execution time takes few seconds to few minutes, we conclude that our approach runs in reasonable time. Hence, we anticipate that our semantic clustering solution should be practical for more larger requirements documents.

Table 6.13: The execution time of the clustering approach for the user story case studies

Case Study	Phase	Execution time in seconds
CMS Company	preprocessing	5
	Clustering (similarity computation+HAC+labelling)	28
	Total	33
Web Company	preprocessing	11
	Clustering (similarity computation+HAC+labelling)	56
	Total	67
Archive Space	preprocessing	7
	Clustering (similarity computation+HAC+labelling)	35
	Total	42

Table 6.14: The execution time of the clustering approach for the case studies of functional requirements written in plain text

Case Study	Phase	Execution time in seconds
MHC-PM System	preprocessing	5
	Clustering (similarity computation+HAC+labelling)	15
	Total	20
UUIS System	preprocessing	6
	Clustering (similarity computation+HAC+labelling)	18
	Total	24
E-store System	preprocessing	9
	Clustering (similarity computation+HAC+labelling)	32
	Total	41
WASP System	preprocessing	13
	Clustering (similarity computation+HAC+labelling)	39
	Total	52

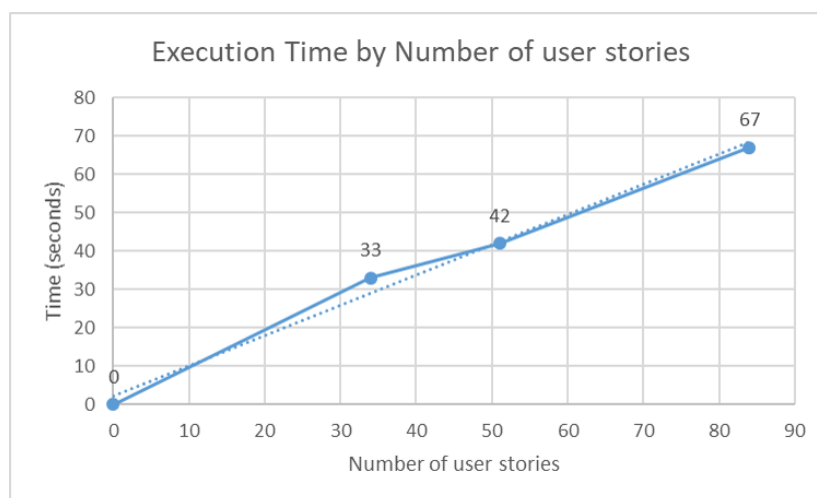


Figure 6.8: Execution time by number of user stories.

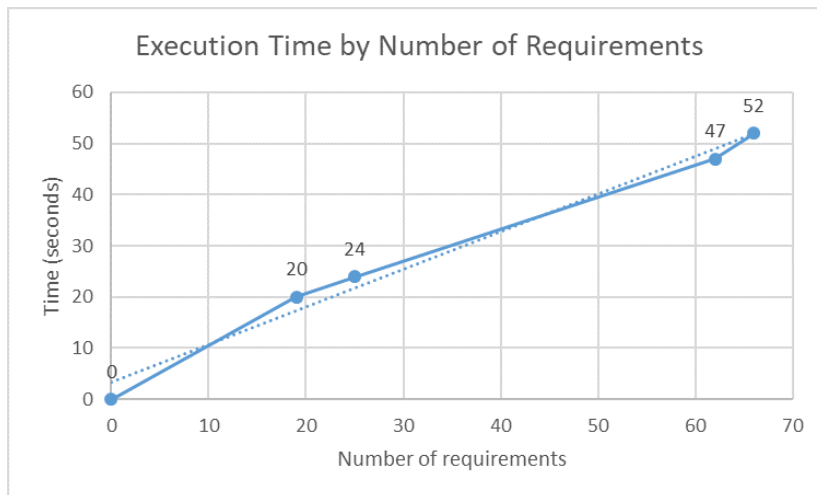


Figure 6.9: Execution time by number of functional requirements written in plain text.

Regarding KPI3, comparing the results of the seven case studies from different domains and containing requirements with different writing styles, we evaluate whether our approach works properly for different domains. We first conduct three real-world case studies of user stories from different domains. Second, we conduct four open-access case studies containing functional requirements written in plain text.

For the two types of the conducted case studies, the evaluation of the proposed clustering approach reveals accurate identification of the semantic clusters (KPI1) as well as a reasonable execution time (KPI2). Accordingly, we conclude that the proposed semantic clustering approach can be applicable in realistic settings.

6.5 Threats to validity

In this section, we discuss the limitations of the proposed approach in terms of internal threats, construct threats, external threats and conclusion threats. These threats are as follows:

Internal Validity. We apply semantic similarity of requirements to extract a package break-down model by means of a clustering approach, regarding requirements with similar functionality as a package. Although this method may lead to bias of identifying packages, we still achieve relatively accurate system decomposition compared with ground truth.

Construct validity. With regard to assessing the accuracy of the proposed clustering approach, the definition of what a "better" cluster is might be a subjective procedure for the user story case studies since there is no baseline in the literature to compare our clustering results with. Although this may lead to bias of evaluating clusters, we still achieve relatively accurate results compared with

the ground truth clusters that are manually created by the team experts. Moreover, the evaluation of the conducted case studies of functional requirements of which the ground truth is available reveals accurate semantic clusters identification.

External validity. Our approach is capable of generating clusters from short text requirements. However, if a requirement describing a functionality is of too many sentences, our approach may not be able to provide an accurate result. Hence, in this case, some manual semantic analyses may still be needed to overcome this limitation.

Conclusion validity. We evaluate the applicability of our approach on seven real-world case studies from different domains and with different writing styles. Although the evaluation results are promising, conducting industrial case studies with larger number of requirements statement may be needed for a better evaluation.

6.6 Conclusion

In this chapter, we first demonstrate the applicability of the proposed requirements clustering approach that aims to break-down the system into a set of sub-systems at early stages. Then, we conducted seven real-world case studies to evaluate the accuracy of the proposed approach. The conducted case studies are from different domains and they contain different writing styles of requirements, i.e., three user story case studies and four case studies of functional requirements written in plain text.

The evaluation results reveal relatively accurate results for the identified semantic clusters as well as a reasonable execution time which demonstrates the generalizability and the applicability of the proposed approach in realistic settings.

In the next chapter, we evaluate the approach that we proposed to automatically generate the preliminary UML architecture design model.

Chapter 7

Evaluation of the automatic generation of the preliminary architecture design models

In this chapter, we demonstrate and assess the applicability of the second pillar of this thesis work that consists in generating automatically the preliminary UML design model from the identified clusters of requirements.

For this purpose, we use the seven real-world case studies that were used to assess the clustering solution. These case studies contain different types of natural language requirements to evaluate the proposed approach: three case studies containing user stories and four case studies containing functional requirements written in plain text.

This chapter is organized as follows: Section [7.1](#) presents the key performance indicators (KPIs) that we investigated in order to evaluate the approach. Section [7.2](#) presents the results analysis and the evaluation. Afterwards, in Section [7.3](#), we assess the investigated KPIs. Then, in Section [7.4](#), we discuss the raised threats to validity and Finally, Section [7.5](#) concludes the chapter.

7.1 Key Performance Indicators (KPIs)

In this section, we present the KPIs that we investigated in order to assess the applicability of the proposed automatic models generation approach in practice. Through these KPIs, we aim to evaluate the generated UML package break-down model containing the UML use-cases representing the preliminary design model. Hence, we formulate the following three KPIs:

- **KPI1: Accuracy of the generated UML use case models**

Motivation. For this KPI, we aim at determining whether our approach succeeded to correctly identify and generate the UML use case models from the identified clusters of requirements. For this purpose, we compare the results of our approach with the ground truth UML use case models. For each case study, ground truth UML use case models are manually created by analysts.

Approach. To assess KPI1, we evaluate the generated UML use case models by matching the identified model elements with the ground truth UML use case models of the case study of interest. For this, we rely on the well-known measures in the Information Retrieval (IR) field which are *precision*, *recall* and *F-measure* [78].

For each generated use case model, let True Positive (TP) elements are the elements identified both manually and by the automatic approach, False Positive (FP) elements be the elements identified by the automatic approach but not manually and False Negative (FN) elements be the elements identified manually but not by the automatic approach. The evaluation metrics are defined as follows:

$$Precision = TP / (TP + FP)$$

$$Recall = TP / (TP + FN)$$

$$F - measure = 2 * Precision * Recall / (Precision + Recall)$$

- **KPI2: Applicability of the proposed approach in realistic settings**

Motivation. For KPI2, we aim to establish whether our approach is scalable. Particularly, the goal is to check how well the automatic models generation approach performs when increasing the number of requirement statements.

Approach. In order to evaluate KPI2, we assess the following validation criterion:

- **The end-to-end execution time of the UML use case model generation approach:**

This validation criterion consists in measuring the impact of the number of requirement statements on the execution time of our approach. Hence, it aims to check whether the UML use case models generation process runs within reasonable time for larger number of requirements in realistic settings.

- **KPI3: Applicability of the models generation approach for different domains**

Motivation. For this KPI, we put emphasis whether the proposed approach works properly for different domains, possibly with different writing styles. Hence, we aim to assess the generalizability of the proposed approach.

Approach. In order to address this KPI, we assess the applicability of our approach on seven real-world case studies from different domains and with different writing style of requirements that we detailed in Section 6.2 in the previous chapter.

7.2 Results analysis and evaluation

In this section, we present and evaluate the results of applying our approach to the seven case studies described in Section 6.2 in the previous chapter . First, we present the obtained results. Then, we evaluate the accuracy of the generated UML use case models across the two types of case studies (i.e., user story case studies and case studies of functional requirements). We also compare the accuracy of the generated UML use case models for the user story case studies with a related approach.

7.2.1 Relevant model elements extraction

In this section, we present and evaluate the results of applying the proposed NLP techniques to extract the relevant elements from each cluster of requirements in order to build the UML use case model describing the system's decomposition.

- **Experimental results**

In this subsection, we show the results of applying each step of our approach (see Section 5.2.1 chapter 5) on the conducted case studies.

The first step consists in extracting from each cluster of a case study of interest the relevant elements that are needed to build the UML use case model namely, actors, use-cases and associations. For this, we developed a model extractor that is based on NLP techniques to extract the relevant model elements. We store the extracted elements in a CSV file which contains three columns: the cluster label, the actor and the use case. An actor and a use-case that are on the same row are linked together by an association relationship. Additionally, an associated actor and a use-case belong to the cluster that is on the same row.

In Figures 7.1, 7.7, we show examples of the extracted use case model elements for each case study that we described in the previous chapter (see Section 6.2).

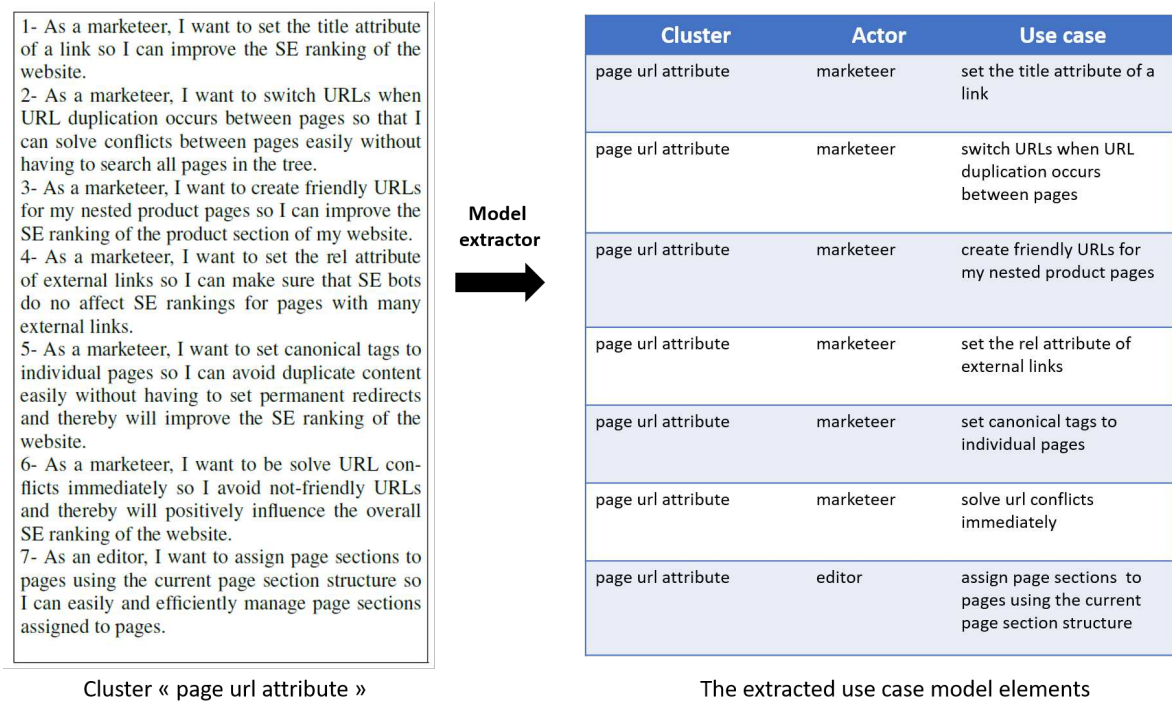


Figure 7.1: Example of extracted elements for the "CMS Company" case study.

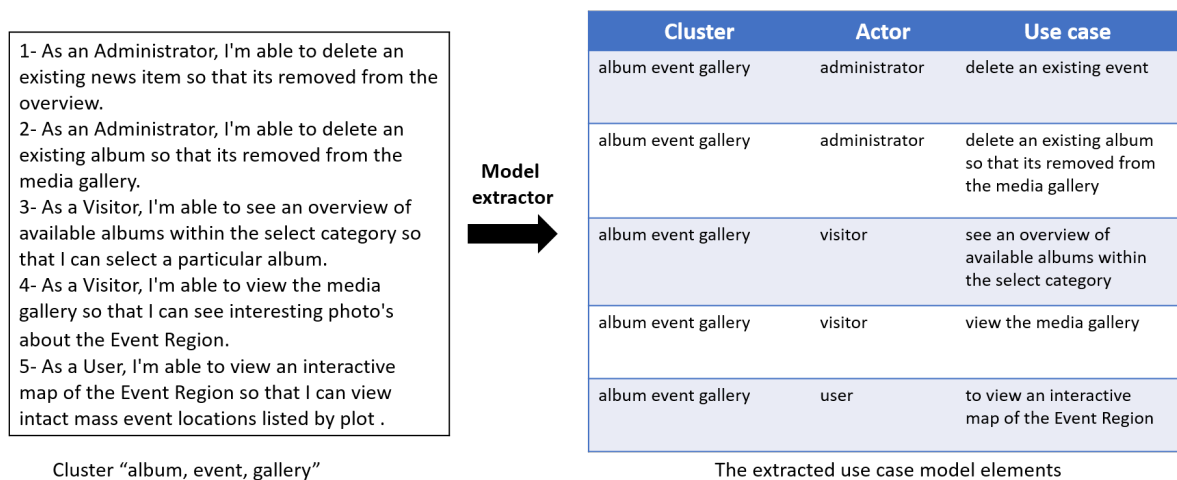


Figure 7.2: Example of extracted elements for the "Web Company" case study.

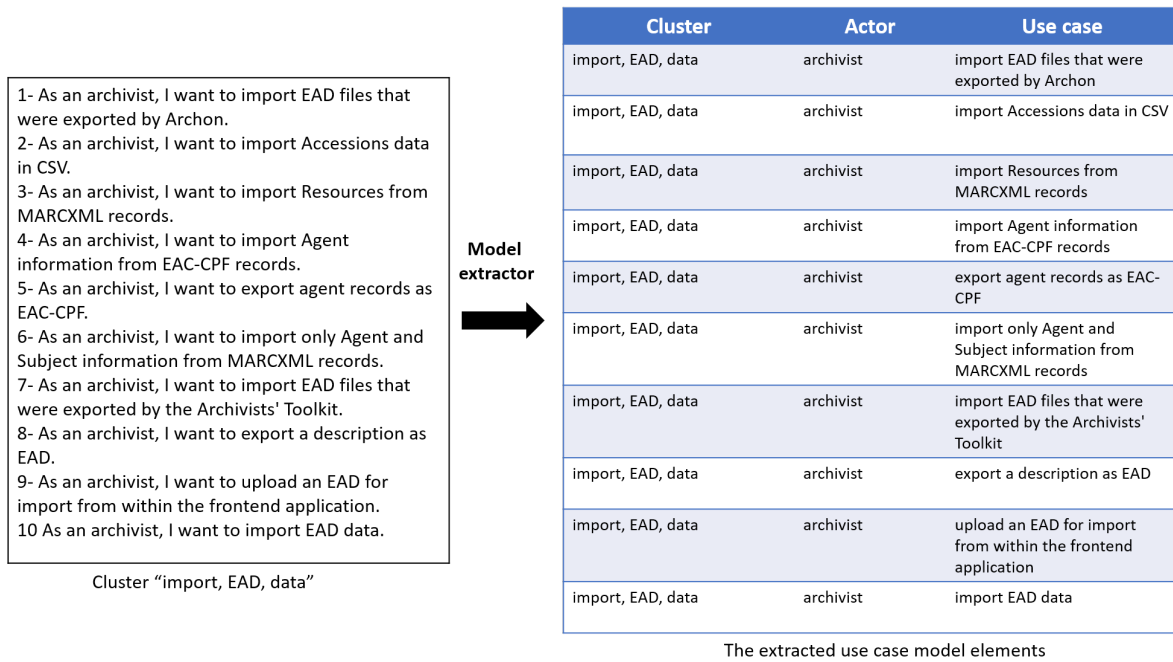


Figure 7.3: Example of extracted elements for the "Archive Space" case study.

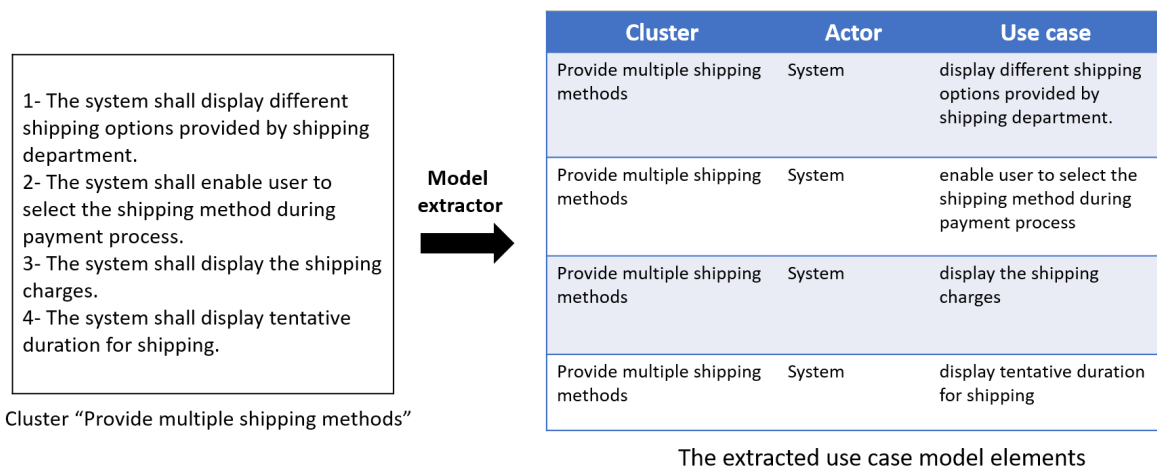


Figure 7.4: Example of extracted elements for the "E-store System" case study.

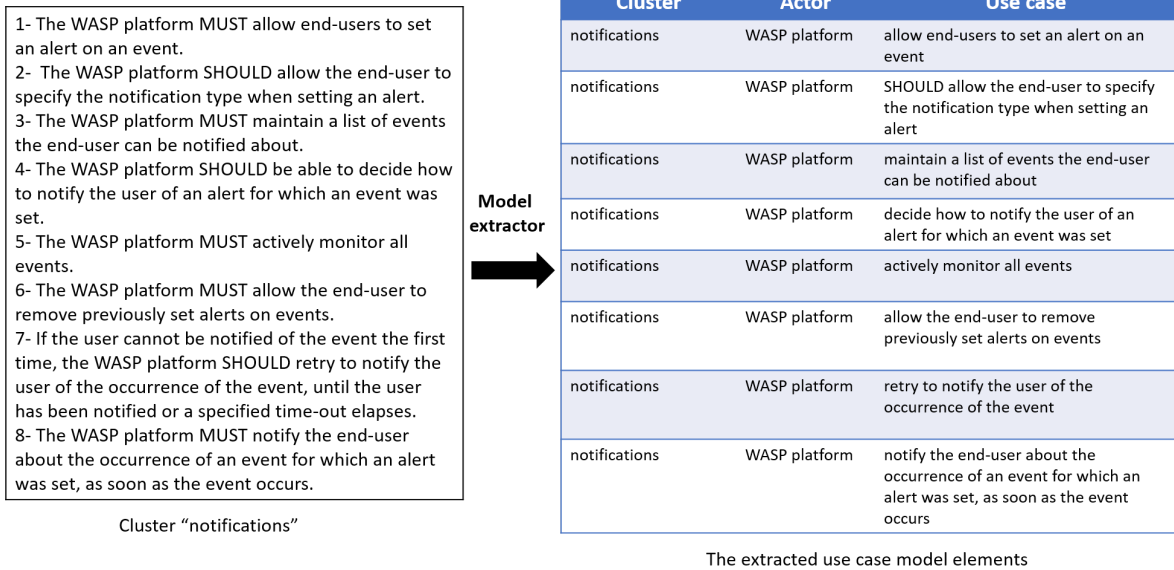


Figure 7.5: Example of extracted elements for the "WASP System" case study.

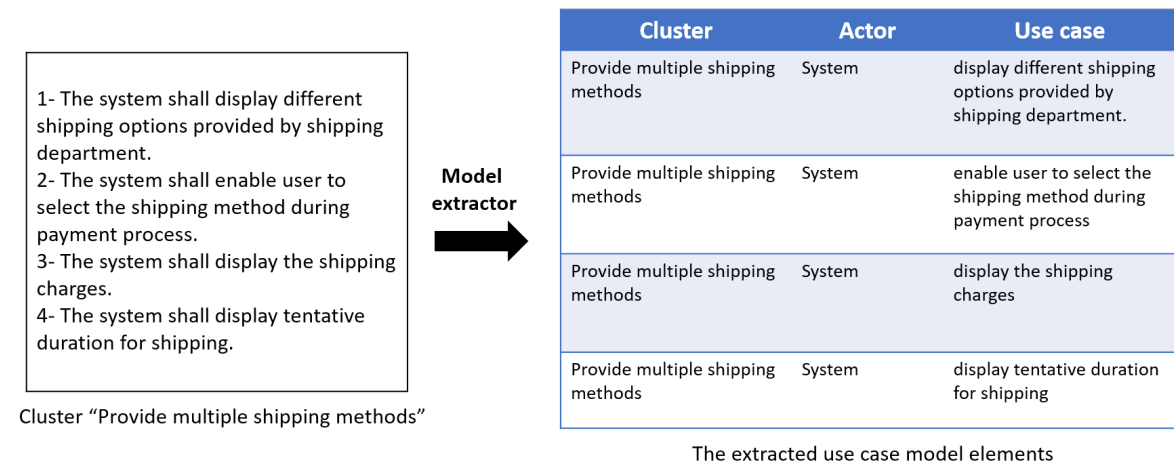


Figure 7.6: Example of extracted elements for the "UUIS System" case study.

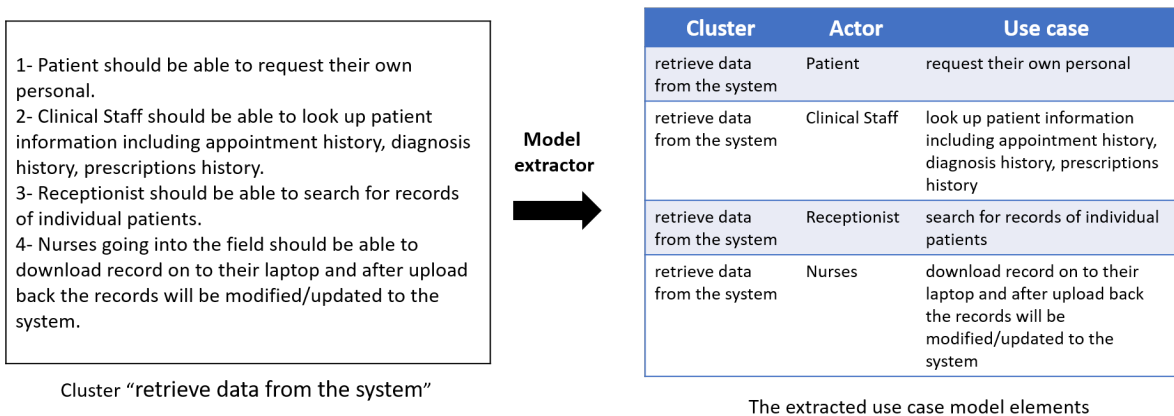


Figure 7.7: Example of extracted elements for the "MHC-PM System" case study.

Afterwards, the extracted use case model elements as well as the identified clusters are fed into the mapping algorithm that programmatically generates the UML package break-down of use-cases. The resulted UML use case models containing the packages break-down are generated upon Papyrus. Figure 7.8 shows an example of the generated UML use case model for the "CMS company" case study. As shown in the figure, the case study contains seven actors: *editor*, *main editor*, *marketeer*, *online channel manager*, *system administrator* and *developer*. The three highlighted packages "*content media language*", "*flash presentation*" and "*page url attribute*" correspond the identified clusters. Each package contains a set of use-cases associated with an actor.

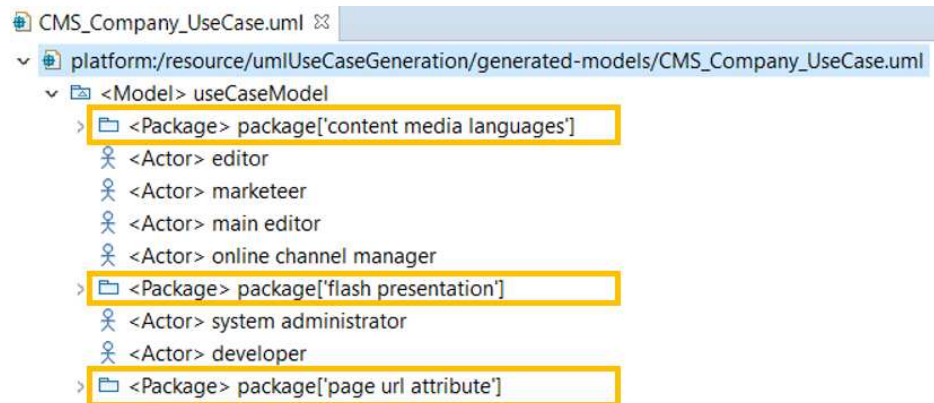


Figure 7.8: The generated UML use case model for the CMS Company.

In Figure 7.9, we zoom in the package "*page url attribute*" in order to show in more details an example of the content of a UML package element. The highlighted part of the generated UML use case model represents the UML package of use-cases corresponding to the cluster "*page url attribute*". As shown in the figure below, the package contains a set of use-cases and their association with the actors. Finally, a visual UML package of use-cases could be displayed to illustrate the features that are embedded in a resulted sub-system. Such visual representation of each resulted UML package of use-cases would help engineers to have a holistic view of the target system.

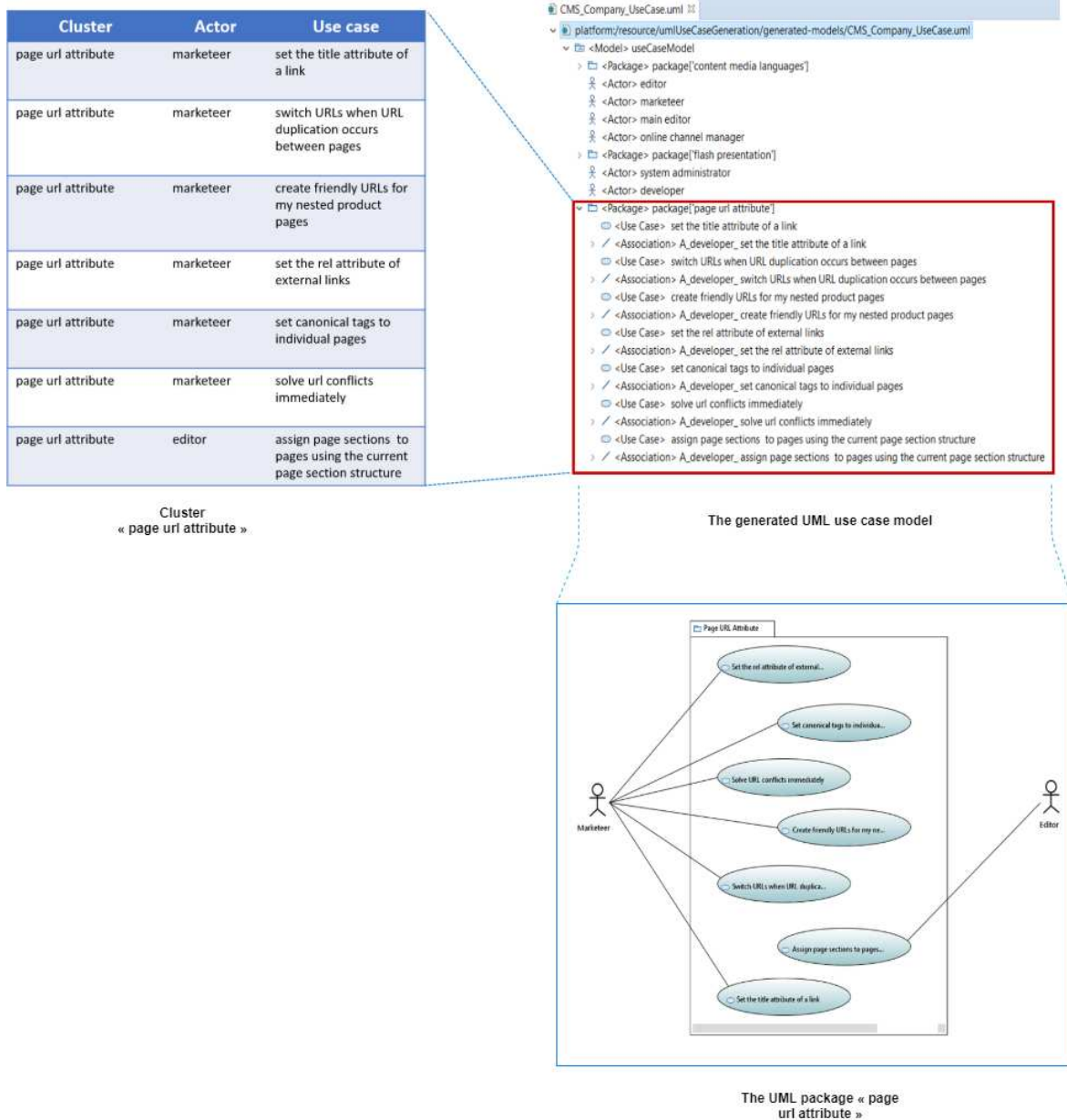


Figure 7.9: Example of a mapping of a cluster into a use case package for the CMS Company case study.

• Results evaluation

In this subsection, we evaluate the results of applying the automatic models generation approach.

For this, we specified True Positive (TP), False Negative (FN) and False Positive (FP) elements applied to actors, use-cases and their relationships. Tables 7.1-7.7 summarize the evaluation of the generated UML use case models in terms of *precision*, *recall* and *F-measure* for the seven conducted case studies. Moreover, we compare the obtained results for the conducted "Web Company" case studies with a related work proposed by Elallaoui et al. [41]. We consider the work in [41] as our

baseline because it closely relates to our work since it aims to extract use case models from the user stories in the "Web Company" case study using a set of NLP rules.

Table 7.1 shows the evaluation results of our approach as well as a comparison with the baseline [41] in terms of *precision*, *recall* and *F-measure*. The best values are displayed in bold.

Table 7.1: Accuracy of the generated UML use case model for the "Web Company" case study.

		Actors	Use Cases	Relationships
		TP FP FN	TP FP FN	TP FP FN
Our approach		84 0 0	84 2 7	69 17 10
	Precision	100%	97%	97%
	Recall	100%	92%	90%
	F-measure	100%	94%	93%
The baseline [41]		TP FP FN	TP FP FN	TP FP FN
		90 1 1	143 20 25	81 20 25
	Precision	98%	87%	87%
	Recall	98%	85%	85%
	F-measure	98%	85%	85%

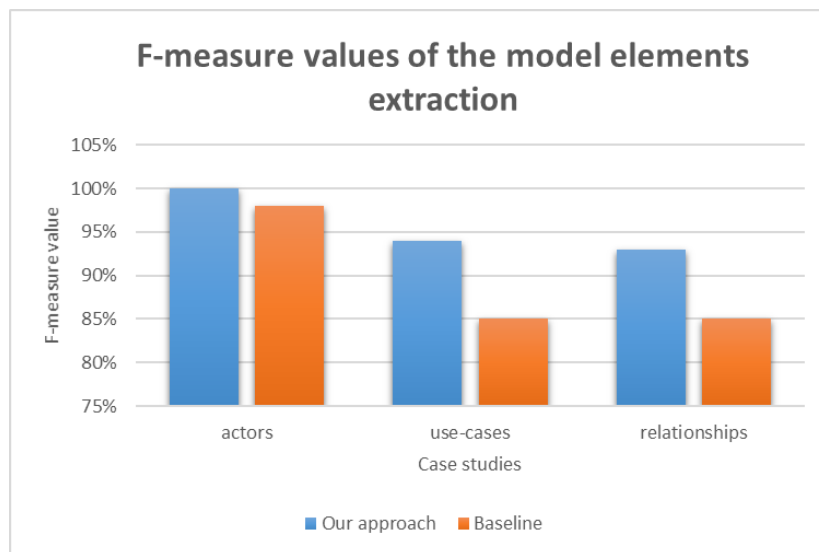


Figure 7.10: Bar graph of F-measure values of the extracted model elements.

Table 7.2: Accuracy of the generated UML use case model for the "CMS Company" case study.

		Actors	Use Cases	Relationships
		TP FP FN	TP FP FN	TP FP FN
		34 0 0	32 2 2	28 4 4
Precision		100%	94%	88%
Recall		100%	94%	88%
F-measure		100%	94%	88%

Table 7.3: Accuracy of the generated UML use case model for the "Archive Space" case study.

	Actors	Use Cases	Relationships
	TP FP FN	TP FP FN	TP FP FN
	56 0 0	51 1 6	33 11 6
Precision	100%	98%	75%
Recall	100%	89%	84%
F-measure	100%	93%	79%

Table 7.4: Accuracy of the generated UML use case model for the "E-store System" case study.

	Actors	Use Cases	Relationships
	TP FP FN	TP FP FN	TP FP FN
	62 0 0	59 9 11	57 12 15
Precision	100%	86%	82%
Recall	100%	84%	79%
F-measure	100%	84%	80%

Table 7.5: Accuracy of the generated UML use case model for the "WASP System" case study.

	Actors	Use Cases	Relationships
	TP FP FN	TP FP FN	TP FP FN
	66 0 0	60 10 15	57 13 18
Precision	100%	85%	81%
Recall	100%	80%	76%
F-measure	100%	82%	78%

Table 7.6: Accuracy of the generated UML use case model for the "UUIS System" case study.

	Actors	Use Cases	Relationships
	TP FP FN	TP FP FN	TP FP FN
	25 0 0	17 6 8	19 8 10
Precision	100%	73%	70%
Recall	100%	68%	65%
F-measure	100%	70%	67%

Table 7.7: Accuracy of the generated UML use case model for the "MHC-PM System" case study.

	Actors	Use Cases	Relationships
	TP FP FN	TP FP FN	TP FP FN
	19 0 0	16 4 6	15 3 7
Precision	100%	80%	83%
Recall	100%	72%	68%
F-measure	100%	76%	74%

As shown in Tables [7.1](#)–[7.7](#), our approach succeeded to achieve relatively accurate results for the extracted use case model elements for the seven conducted case studies. For all the conducted case studies, we succeeded to extract all the actor elements even actors with compound nouns such as

"read Only user" and "repository Manager". Consequently, *precision*, *recall* and *F-measure* values for the actors extraction is equal to 100% for the seven case studies.

We evaluate the accuracy of the extracted model elements based on the *F-measure* metric as it combines both precision and recall values.

For use-cases and relationships extraction, *F-measure* values take a high-range (93%-94%) and (79%-93%) respectively for the case studies of user stories.

Additionally, as shown in Figure 7.10 evaluation results indicate that our approach succeeded in extracting all the actors while the approach proposed in the baseline [41] failed to extract actors with compound nouns such as "newly registered user". Moreover, for use-cases and relationships, our approach achieves better *F-measure* results than the baseline [41] by 9 and 8 percentage points respectively.

Regarding the case studies of functional requirements, *F-measure* values of use-cases and relationships extraction take a reasonable-range (70%-84%) and (67%-80%) respectively.

For use-cases and relationships extraction, the values are slightly affected by the fact that our approach does not support inclusion, extension and generalization relationships between use-cases.

We also observe that the accuracy of use-cases and relationships extraction for the case studies of functional requirements is slightly lower than the accuracy values for the case studies of user stories. This is mainly due to the variability of natural language as the considered case studies of functional requirements are expressed in plain text contrary to the semi-controlled natural language used to express user stories which facilitates the elements identification process.

7.3 Assessing KPIs

Regarding KPI1, our approach provides relatively high *precision*, *recall* and *F-measure* values for the identified use case model elements across different cases studies containing different writing styles of requirements (see Tables 7.1-7.7). Moreover, by comparing our approach with the related work proposed in [41] (Table 7.1), we observe that our approach provides better *precision*, *recall* and *F-measure* values for the "Web Company" case study.

At the light of these results, it is noticeable that our automatic model elements extraction approach succeeded to achieve relatively accurate results that can be applicable regardless the writing style of the input natural language requirements.

In order to evaluate KPI2, we assess the applicability of the proposed clustering solution by measuring the end-to-end execution time across different case studies. The experiments were carried out on a laptop with a 2.10 Ghz Intel (R) Core (TM) i7-4600U CPU and a 8GB of memory.

Table 7.8: The execution time of the automatic models generation approach for the user story case studies.

	CMS Company	Web Company	Archive Space
Execution time in seconds	5	8	6

Table 7.9: The execution time of the automatic models generation approach for the case studies of functional requirements.

	E-store System	WASP System	UUIS System	MHC-PM System
Execution time in seconds	7	8	5	6

In Tables 7.8 and 7.9, we present the impact of the number of requirements on the execution time across the seven chosen case studies. As shown in the Tables 7.8 and 7.9, the automatic models generation approach takes few seconds (between 5s and 8s) for the seven conducted case studies.

Hence, we conclude that our approach runs in a reasonable time and we anticipate that it should be practical for more larger requirements documents.

Regarding KPI3, we conduct seven case studies from different domains and containing requirements with different writing styles. We first conduct three real-world case studies of user stories. Second, we conduct four open-access case studies containing functional requirements written in plain text. For both types of writing styles of the conducted case studies, the evaluation of the accuracy of the proposed models generation approach in terms of *precision*, *recall* and *F-measure*, reveals relatively accurate result that can be applicable in realistic settings.

7.4 Threats to validity

In this section, we discuss the limitations of the proposed models generation approach in terms of internal threats, construct threats and external threats. These threats are as follows:

Internal validity. The proposed use case models generation approach does not support the extraction of inclusion, extension and generalization relationships. Although, this might slightly affects the accuracy of the extracted relationships, we still achieve relatively accurate results for the relationships identification.

Construct validity. With regard to assessing the automatic models generation approach, the evaluation procedure might be a subjective since there is no baseline in the literature for all the considered case studies to compare our results with, except for the "Web Company" case study. Although this may lead to bias of evaluating the extracted model elements, we still achieve relatively

accurate results compared with ground truth that was created manually by analysts.

External validity. We evaluate the applicability of our approach on seven real-world case studies. Although the evaluation gives promising results, we need to evaluate the approach on further case studies including industrial ones.

7.5 Conclusion

In this chapter, we firstly demonstrate the applicability of the proposed UML use case models generation approach describing the system's decomposition from early requirements. Given as input a set of clusters of similar natural language requirements, the proposed approach automatically generates the UML use case model containing a package break-down of the target system.

To evaluate the accuracy of the proposed approach, we conducted the seven real-world case studies that were used in the previous chapter. The evaluation results reveal relatively accurate results for the generated UML use case models as well as a reasonable execution time which demonstrates the applicability and the generalizability of the proposed approach in realistic settings.

Chapter 8

Conclusion and future work

In this dissertation, we proposed a first step towards applying AI for MBSE to optimize the adoption of MBSE and resolve some of its challenges. Specifically, we addressed the issue of automatically deriving preliminary architecture design models expressed in UML from natural language requirement in the context of complex systems.

We have provided a new flow of AI components including their specific parametrization, enabling the automation to go from natural language requirements to preliminary UML architecture design. A prototype has been developed to support the contributions and validate them. The resulted UML design models describing the system of interest are generated upon *Papyrus*.

This chapter summarizes the main contributions of the work presented in this dissertation, recapitulates how we validated them, and finally identifies future research work.

Summary of contributions.

Our first contribution was to provide an early decomposition of the complex system into a set of sub-systems based on the semantic similarity between requirement statements. In fact, such early decomposition supports the "divide-to-conquer" strategy i.e., building smaller pieces to reduce the complexity and help developers to better understand and realize the target software project. In addition, it helps to design a preliminary architecture design describing the systems decomposition as the identified groups of requirements represent components or sub-systems that should be implemented and reused.

For this, we proposed a semantic similarity computation module that analyzes and computes the semantic similarity of both word-level and requirement-level of each pair of requirement statements. First, the semantic similarity module computes the word-level similarity of requirements using the

neural word embedding model word2vec as a prediction model. Second, the requirement-level semantic similarity are derived from the obtained word-level semantic similarity using the Mihalcea's scoring formula for text similarity.

Third, we employed the Hierarchical Agglomerative Clustering algorithm to group the similar requirements and provide a set of semantic clusters denoting the early decomposition of the system.

The second contribution of this thesis consists in generating automatically the UML preliminary architecture design model that consists of a UML package break-down of use-cases describing the system's decomposition. For this, we developed a model extractor that uses a set of NLP heuristics to extract from each cluster of requirements, the relevant model elements that are needed to build the use cases model describing the features of target the system. Then, we implemented a mapping algorithm that programmatically generates the UML package break-down model denoting the system's decomposition including the UML use case model describing its expected functionalities.

Validation.

The results of our contributions have been validated with literature searching, case studies and feedback obtained during the elaboration and presentation of peer-reviewed scientific publications.

Firstly, we have started our work with a literature search of existing research results, techniques, and tools that are related to our work. We continued to review other new results during the three years of this Ph.D. thesis. Among the existing works, we found two existing approaches [96, 41] that closely relate to our work. Hence, we used these two approaches as baselines to assess our proposal.

Secondly, and in order to evaluate the generalizability of our approach, we conducted seven real-world case studies from different domains, with different number of requirements and containing different writing styles of natural language requirements. Among the considered case studies, we conducted three case studies of user stories in the context of Scrum process, and four open-access case studies containing functional requirements written in plain text. Indeed, the diversity of the conducted case studies helps to evaluate whether our approach works properly for different domains, different number of requirements as well as for different writing styles.

Then, we defined a set of Key Performance Indicators (KPIs) that aim to assess whether our proposal provides accurate results that could be applicable in realistic settings.

The evaluation results of both semantic clustering approach and the automatic generation of preliminary design model approach reveal relatively accurate results in terms of *precision*, *recall*, *F-measure* and execution time. Moreover, the comparison of the obtained results with the two baselines

shows that our proposal succeeded to achieve better *precision*, *recall* and *F-measure* results for both semantic clustering of requirements and automatic generation of preliminary design model.

To conclude, the evaluation of the obtained results demonstrates the accuracy of our proposal as well as its applicability and generalizability in realistic settings.

Future work

From the work we have accomplished in this dissertation, we see several research directions worth investigating. Several improvements can also be considered for the prototype that has been implemented.

Improvement of the current prototype. In this dissertation work, we have implemented a prototype that automatically generates a package break-down model including a use case model, from natural language requirements as a proof of concepts. Although our prototype supports different writing styles of input requirements, it supports only three templates of user stories.

In a future work, we can extend our prototype to support other possible templates of user stories as input. Moreover, conducting industrial case studies with larger number of requirements statement may be needed for a better evaluation of our prototype. Indeed, this could help to generalize the applicability of our approach in both the industrial context and Scrum processes.

Another direction that could be investigated in a future work is the extension of our prototype to support the extraction of extension, inclusion and generalization relationships for the generated UML use case model. To this end, specific NLP heuristics with further semantic analysis among requirement statements have to be implemented.

Deriving initial architecture design models from non-functional requirements. Requirement documents commonly have two types of requirements one is functional requirements, which defines the feature of the system-to-be, and the other is non-functional requirements, which defines the quality attributes of the system features. Such attributes enforce operational constraints on different aspects of the system's behavior, such as its usability, security, reliability, performance, and portability [30].

In our research, we applied our prototype on functional requirements. In fact, developers emphasize more on the functional side of the software underrating the non-functional quality attributes such as development time and cost which later on leads to project failure.

In future work, we could consider the issue of deriving initial architecture design from non-functional requirements in the context of complex systems.

Indeed, most of the work focusing on automating non-functional requirements categorization use supervised learning techniques requiring huge training datasets, which are not always available for

all domains. The PROMISE repository for requirements engineering has been used in most of these approaches either to classify requirements into FRs and NFRs or, to classify NFRs into subcategories [76], [104]. However, when a classification task of requirements related to a specific domain of application needs to be executed (e.g., grouping FR requirements of a given system by functionality), chances are that no relevant dataset exists. Thus, high-quality dataset creation is needed if datasets containing requirements related to a specific domains of application are not readily available, which is often the case [48].

In order to mitigate these limitations, we could extend our clustering solution to categorize non-functional requirements according to their type. So far, employing clustering to categorize non-functional requirements did not provide sufficient accuracy compared with existing works which rely on supervised learning methods. Hence, we could integrate additional methods taking into account popular key words describing each non-functional requirement type to enhance the categorization process.

Regarding the automatic models generation from the clusters of non-functional requirements, we could implement specific NLP heuristics to extract relevant elements that are needed to build the initial architecture design. The NLP heuristics to be implemented will depend heavily on the choice of the UML design model to be generated which should describe efficiently the non-functional requirements of the target system.

Appendix A

Résumé

A.1 L'intelligence artificielle pour automatiser l'ingénierie logicielle

L'industrie d'ingénierie logicielle est toujours à la recherche de moyens meilleurs et efficaces pour créer des produits logiciels de meilleure qualité.

L'ingénierie des systèmes basée sur les modèles (MBSE), telle que définie par le conseil international d'ingénierie des systèmes (INCOSE) [112], est *"l'application formalisée de la modélisation pour prendre en charge les exigences du système, la conception, l'analyse, la vérification et les activités de validation commençant dans la phase de conception et se poursuivant tout au long des phases de développement et du cycle de vie ultérieur"*. MBSE comprend plusieurs concepts de modélisation : processus, langages, méthodes et outils pour produire un ou plusieurs modèles de système. Par conséquent, il promet de soutenir les entreprises d'ingénierie logicielle en permettant la réalisation de systèmes logiciels réussis en favorisant une vision holistique de la conception et en permettant une architecture logicielle maintenable et de haute qualité.

Malgré ses avantages théoriques, plusieurs études ont démontré que la méthodologie du MBSE reste difficile à appliquer [87, 10]. Par conséquent, le MBSE n'est pas encore largement adopté dans les applications du monde réel car il est toujours aux prises avec d'énormes défis [5, 27, 28] que ni la partie basée sur les modèles ni la partie ingénierie des systèmes ne sont capables de gérer. Nous pouvons résumer tous les défis liés à l'adoption du MBSE en un seul : ses avantages ne l'emportent pas sur ses coûts [24].

Les récents progrès technologiques concernant la big data, le développement de systèmes, d'algorithmes et d'outils plus complexes ont permis aux industries d'avoir de nombreuses oppor-

tunités d'utiliser l'intelligence artificielle (IA). La discipline de l'IA est généralement reconnue depuis plus de sept décennies. Il peut être décrit comme la science de l'imitation des facultés mentales humaines sur un ordinateur [54].

Les systèmes d'IA comprennent des modules qui permettent de générer des types d'apprentissage. Par exemple, l'apprentissage automatique (ML) est un type de technique d'intelligence artificielle qui prend des décisions ou des prédictions basées sur des données. Le traitement du langage naturel (TALN) est la branche de l'IA qui permet aux ordinateurs de comprendre, d'interpréter et de manipuler les langues humaines. En fait, les techniques d'IA associées à une technologie appropriée ont permis aux systèmes de percevoir, de prédire et d'agir pour aider les humains dans un large éventail d'applications.

En 2016, l'IA a fait une percée majeure lorsqu'un système informatique développé par les chercheurs de Google DeepMind, AlphaGo, a battu le meilleur joueur humain du monde au Go, un jeu bien plus compliqué que les échecs. Au cours de la même année, la société Microsoft a lancé son chatbot IA Tay pour mieux comprendre la façon dont les adolescents parlent via Twitter. Tay visait à apprendre à mieux parler au fil du temps à travers des conversations, cependant, il a été fermé seulement 16 heures après son lancement lorsqu'il a commencé à publier des tweets incendiaires et offensants. Bien que ces deux exemples ne puissent pas être comparés en raison de leurs domaines d'application différents, il est crucial de reconnaître à la fois les avantages et les inconvénients de l'application de l'IA et de la garder sous contrôle.

Bien que les disciplines de l'IA et de l'ingénierie logicielle se soient développées séparément, elles ont de nombreux points communs. En fait, les deux disciplines traitent de la modélisation d'objets du monde réel à partir du monde réel comme les processus métier, les connaissances d'experts ou les modèles de processus [94].

De nos jours, plusieurs directions de recherche des disciplines de l'IA et de l'ingénierie logicielle se rapprochent et commencent à construire de nouveaux domaines de recherche, l'ingénierie logicielle automatisée en faisant partie. L'ingénierie logicielle automatisée est un domaine de recherche qui développe constamment de nouvelles méthodologies et technologies afin de créer des systèmes logiciels qui présentent une certaine forme d'intelligence humaine. Il vise à assister ou à automatiser les activités de l'ingénierie logicielle afin d'améliorer l'efficacité, de réduire le temps et les coûts du processus de développement du système.

A.2 Automatisation du MBSE en utilisant l'IA - les problèmes potentiels

Au cours des dernières années, les contraintes liées à la conception des systèmes évoluent de plus en plus et nécessitent l'intégration d'un plus grand nombre d'intervenants dans les projets pour gérer diverses nouvelles préoccupations - telles que la sécurité, la sûreté, les coûts et la durabilité - plus tôt dans le processus, au moment de la spécification. Par conséquent, les projets logiciels modernes sont de plus en plus volumineux et donc plus complexes que dans le passé. En particulier, la croissance exponentielle du nombre d'exigences système soulève des difficultés pour gérer manuellement les exigences et avoir une vision claire et cristalline des attentes et de la portée du système à concevoir [22].

Les méthodes de développement basées sur des modèles garantissent l'application formalisée de la modélisation pour la spécification du système, au lieu de le faire simplement en utilisant un texte informel ou une description de dessins. Cependant, les modèles de conception d'architecture sont toujours extraits manuellement par les ingénieurs, ce qui est devenu une tâche fastidieuse, chronophage et sujette aux erreurs, en particulier avec la croissance exponentielle des exigences du système et la nécessité de tout retracer tout au long du cycle de vie du produit à concevoir. Ainsi, cette tâche est critique car les erreurs introduites au début du développement sont les plus difficiles et les plus coûteuses à corriger [23].

De plus, la demande croissante d'agilité et le développement de systèmes plus complexes ont conduit les praticiens à se concentrer sur la programmation plutôt que sur la gestion des exigences, la planification, la spécification, l'architecture, la conception et la documentation [80]. Par conséquent, l'adoption de méthodes basées sur des modèles pour développer des systèmes complexes devient un défi car cela nécessite beaucoup de temps, de coûts et d'investissements en ressources [72]. En effet, le ROI (Retour sur Investissement) déployant le MBSE est en effet plus perçu sur le long terme que sur le court terme [13]. Le manque d'expertise humaine ainsi que des outils d'automatisation puissants sont souvent cités comme les principaux obstacles clés qui ralentissent encore la diffusion de l'approche du MBSE et présentent des obstacles importants pour démontrer son ROI. Par conséquent, il va de soi que les progrès de l'IA peuvent apporter une grande valeur pratique pour atténuer certains des défis soulevés par l'adoption du MBSE.

Dans les années 1990, Ian [106] a posé la question : « *Pourquoi l'IA n'a-t-elle jamais été mentionnée dans une discussion de séminaire sur l'avenir de l'ingénierie logicielle ?* » Après près de 30 ans, et en regardant dans le SE Handbook [112] et le récent inspirant SE Vision 2025 [57], il est également surprenant que l'application de l'IA pour l'ingénierie logicielle ne soit pas clairement men-

tionnée. On se demande donc, ***au lieu d'ignorer de nombreuses méthodes MBSE bien connues, pourquoi ne pas automatiser la conception des systèmes à partir des exigences logicielles à l'aide de solutions intelligentes ?***

Notre recherche englobe l'intégration des domaines du MBSE et de l'IA - en particulier des techniques de ML et de TALN - pour automatiser la génération de conception d'architecture de systèmes complexes et ainsi, améliorer l'efficacité, réduire le temps et les coûts du processus de développement.

A.3 Objectif de la thèse

Dans les premières étapes du processus de développement logiciel, les ingénieurs découvrent et collectent les exigences des clients, puis les enregistrent manuellement dans un document de spécification des exigences. Les exigences rassemblées décrivent différents aspects du logiciel cible dans un langage naturel informel. L'élicitation et la gestion des exigences ont un impact significatif sur la qualité et le succès des systèmes d'information, car les erreurs introduites au début du développement sont les plus difficiles et les plus coûteuses à corriger [23]. Les chiffres de l'industrie indiquent qu'une ingénierie des exigences insuffisante est à l'origine de l'échec de plus de 50% des projets logiciels [70]. Par conséquent, il est crucial que les exigences définies soient bien comprises et bien gérées par les ingénieurs [118].

Ces dernières années, les contraintes de conception du système évoluent de plus en plus, nécessitant d'intégrer davantage d'intervenants dans les projets pour gérer diverses nouvelles préoccupations - telles que la sécurité, la sûreté, les coûts et la durabilité - plus tôt dans le processus, au moment de la spécification. Par conséquent, les ensembles d'exigences utilisés dans l'analyse et la conception de tels systèmes sont souvent si vastes que les techniques traditionnelles de gestion des exigences et d'organisation deviennent lourdes. De plus, les projets logiciels modernes deviennent beaucoup plus grands et plus complexes.

En effet, de nombreux défis classiques du développement de produits logiciels découlent de cette complexité et de ses augmentations non linéaires avec la taille [22]. En particulier, la croissance exponentielle du nombre d'exigences pose des difficultés pour gérer manuellement les exigences et avoir une vision cristalline claire des attentes et de la portée du système à concevoir. Bien que les méthodes du MBSE restent le point central qui assure la transition des exigences de langage naturel à la conception de l'architecture, son adoption fait toujours face à des obstacles importants pour démontrer son ROI, en particulier avec la complexité croissante des systèmes développés.

Les méthodes basées sur l'IA de leur côté ont montré leur capacité à améliorer le processus de

développement logiciel, y compris la modélisation des exigences. Par conséquent, nous visons à tirer parti de ces avancées pour automatiser la transition des exigences logicielles exprimées en langage naturel aux modèles de conception d'architecture préliminaires dans le contexte de systèmes complexes.

La principale question de recherche de notre travail est la suivante : ***"Quelle méthodologie est suffisamment efficace pour faire face à la fois à la complexité des systèmes logiciels et à l'automatisation de la génération de conception d'architecture à partir des exigences du système exprimées en langage naturel ?"***

La principale question de recherche vise à définir une méthodologie basée sur les composants de l'IA, qui devrait faire face aux défis abordés à la fois par (1) la complexité des systèmes en raison du nombre croissant d'exigences, ce qui soulève la nécessité de décomposer le système en un ensemble de sous-systèmes; (2) l'automatisation de la génération des modèles de conception d'architecture exprimés en UML décrivant la décomposition du système cible afin de fournir une visibilité holistique du système complexe. Nous avons décomposé la question de recherche ci-dessus en deux questions de recherche sous-jacentes plus précises et plus ciblées qui constituent les principales préoccupations de cette thèse :

- **RQ1- "Comment garantir une décomposition précise du système complexe aux premiers stades du processus de développement ?"**

L'objectif de cette question de recherche est de définir une méthode qui décompose le système en sous-systèmes en se basant sur les similarités sémantiques entre les exigences logicielles du système cible. Pour cela, il est crucial de :

-Premièrement, définir une méthode de calcul de similarité qui assure un regroupement efficace des exigences exprimées en langage naturel qui partagent des fonctionnalités ou des caractéristiques similaires. Dans ce contexte, de nombreuses méthodes ont été proposées dans la littérature pour calculer la similarité entre les énoncés d'exigences exprimées en langage naturel. Cependant, les méthodes existantes sont plutôt immatures, c'est-à-dire qu'elles manquent de précision (en ce qui concerne les groupes d'exigences identifiés) et souffrent d'informations sémantiques extraites incomplètes. Ainsi, notre objectif est de proposer un module de calcul de similarité qui améliore l'extraction de l'information sémantique à partir des énoncés d'exigences et améliore la précision des groupes d'exigences identifiés.

-Deuxièmement, définir une méthode pratique basée sur l'apprentissage automatique qui garantit un regroupement précis des déclarations d'exigences en langage naturel en fonction de la similarité sémantique extraite parmi les exigences logicielles. La solution de regroupement

doit offrir un degré élevé d'automatisation afin d'accélérer le processus et d'améliorer son efficacité. Les groupes identifiés d'exigences similaires représenteront une première partition du système complexe en un ensemble de sous-systèmes qui pourront être éclatés en composants de conception d'architecture avec des degrés de détail supplémentaires.

- **RQ2- "Comment automatiser la génération de modèles de conception d'architecture à partir des exigences du système exprimées en langage naturel ?"**

Le langage naturel est la notation prédominante que les praticiens utilisent pour représenter les exigences du système. Bien que facile à lire, le langage naturel ne met pas facilement en évidence les concepts et les relations clés tels que les dépendances. Cela contraste avec la capacité inhérente des modèles de conception à visualiser un système donné de manière holistique. C'est là que réside l'importance d'une solution automatique qui comble le fossé entre les exigences du langage naturel et les modèles de conception d'architecture.

Après avoir décomposé le système en un ensemble de sous-systèmes, il est crucial de fournir une vision claire et holistique des différents sous-systèmes décomposant le système cible. En fait, une telle représentation visuelle permet aux ingénieurs de mieux comprendre les caractéristiques et la portée du système cible et contribue à accélérer le processus de développement. Cependant, en raison du manque d'outils puissants, en particulier d'outils permettant une automatisation élevée des processus, les approches basées sur des modèles ne sont encore que marginalement adoptées par les ingénieurs logiciels. Ainsi, la deuxième question de recherche est abordée par la définition d'une solution qui permet un haut degré d'automatisation de la génération d'une conception d'architecture préliminaire exprimée en UML décrivant le système ainsi que ses sous-systèmes.

A.4 Contributions

Dans des scénarios réels, l'ingénierie des exigences et la conception de l'architecture peuvent ne pas être aussi proches qu'elles semblent l'être en théorie. Les interdépendances et les contraintes entre les artefacts architecturaux et les exigences initiales sont difficiles à comprendre et à modéliser lors du développement logiciel. Cela est principalement dû à la complexité croissante des systèmes logiciels modernes ainsi qu'au manque d'outils d'automatisation offrant une conception architecturale précise aux premiers stades du processus de développement.

Dans cette thèse, nous visons à contribuer à une première étape vers l'application de l'IA pour le MBSE afin d'optimiser l'adoption du MBSE et de résoudre un ensemble de ses défis. En par-

ticulier, nous proposons une *méthodologie basée sur l'IA qui permet d'automatiser la transition des exigences du système écrites en langage naturel à la conception d'architecture décomposant le système cible à des stades précoces* comme l'illustre la Figure [A.1](#)

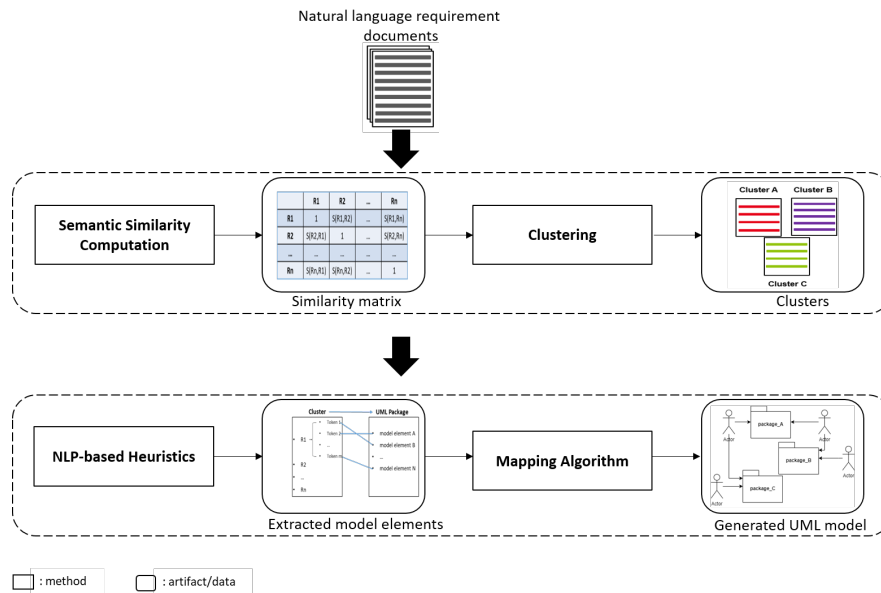


Figure A.1: Un aperçu de l'approche proposée

Notre méthodologie répond aux deux questions de recherche mentionnées précédemment. La première question de recherche (1) "**Comment garantir une décomposition précise du système complexe aux premiers stades du processus de développement ?**" est abordée par la définition et la mise en œuvre d'une méthodologie basée sur l'apprentissage automatique qui permet le regroupement d'exigences logicielles sémantiquement similaires et, par conséquent, permet la décomposition du système logiciel en sous-systèmes à des stades précoces. En apprentissage automatique, il existe deux méthodes pour atteindre cet objectif : la classification des exigences (apprentissage supervisé) et le clustering des exigences (apprentissage non supervisé).

Néanmoins, les données d'entraînement de haute qualité nécessaires à la tâche de classification des exigences ne sont pas toujours disponibles pour tous les domaines d'application. Par conséquent, comme notre objectif est de fournir une méthode applicable à tous les domaines d'application, nous proposons une solution de clustering basée sur la similarité sémantique des exigences logicielles. La solution de clustering proposée vise à fournir une première partition du système complexe en un ensemble de sous-systèmes en regroupant des exigences similaires dans le même cluster. Ainsi, chaque cluster définit un sous-système qui couvre une fonctionnalité/caractéristique particulière du système complexe et qui représente un composant

de conception d'architecture avec des degrés de détail supplémentaires.

Comme le montre la figure [A.1](#), la solution de clustering proposée est basée sur un module de calcul de similarité sémantique qui extrait la similarité sémantique entre les exigences. Le module de calcul de similarité est principalement basé sur l'incorporation de mots et prend en compte à la fois la similarité au niveau du mot et au niveau de l'énoncé de l'exigence afin d'améliorer l'extraction de la similarité sémantique entre les paires d'exigences. Par conséquent, le calcul de similarité sémantique comprend deux étapes : (i) similarité au niveau des mots : nous utilisons un modèle de prolongement lexicaux neuronaux, word2vec, comme modèle prédictif pour calculer la similarité sémantique entre les paire de mots dans chaque paire d'énoncés d'exigences. (ii) similarité au niveau des énoncés des exigences : nous étendons la similarité au niveau des mots au niveau des exigences. Pour cela, nous utilisons une fonction de notation pour la similarité de texte qui prend en compte la similarité sémantique mot à mot obtenue dans l'étape précédente, et génère en sortie la matrice de similarité de chaque paire d'énoncés d'exigences.

De plus, nous avons mis en place une opération qui identifie automatiquement le nombre optimal de clusters à générer afin de réduire l'intervention manuelle. Enfin, les exigences sont regroupées en fonction de leurs scores de similarité sémantique à l'aide d'un algorithme de clustering et les clusters générés sont étiquetés automatiquement en utilisant une fonction d'identification et de classement des mots clés.

La question de recherche (2) "**Comment automatiser efficacement la génération de modèles de conception d'architecture en UML partir des exigences système exprimées en langage naturel ?**" est abordée par la définition et la mise en œuvre d'une méthode qui génère automatiquement les modèles de conception d'architecture décomposant le système complexe en se basants sur les groupes d'exigences identifiés.

Afin d'atteindre cet objectif, nous définissons et implémentons d'abord un ensemble d'heuristiques de TALN qui extraient les éléments pertinents pour construire le modèle préliminaire de conception à partir de chaque exigence au sein des clusters identifiés. Le modèle préliminaire d'architecture généré consiste en un modèle UML de paquets de cas d'utilisation décrivant une décomposition initiale du système cible. En effet, l'adoption d'un modèle de paquets fournissant une décomposition initiale du système permet de représenter et de communiquer ce qui est important entre les parties prenantes et les développeurs, donne un aperçu des fonctionnalités attendues du système cible et permet de suivre les exigences recueillies tout au long du projet.

Cette étape vise à extraire les relations intra-clusters des concepts du modèles de conception à partir de chaque cluster. Par conséquent, le résultat de cette étape fournit des degrés de détail supplémentaires sur les composants d'architecture identifiés qui décomposent le système complexe.

Ensuite, la deuxième étape consiste à implémenter un algorithme de mapping qui transforme les éléments pertinents du modèle en leurs correspondants dans le modèle de paquet de cas d'utilisation UML. Par conséquent, la sortie est un modèle préliminaire de paquet de cas d'utilisation en UML. Ainsi, le modèle de conception généré fournit une vue holistique du système cible en montrant les relations intra-cluster entre les concepts du modèle.

A.5 Validation

En plus de l'implémentation des prototypes, les résultats de nos contributions ont été validés par des recherches bibliographiques, des études de cas et de la présentation de publications scientifiques évaluées par des pairs.

Premièrement, nous avons commencé notre travail par une revue de littérature des résultats de recherche, des techniques et des outils existants qui sont liés à notre travail. Nous avons continué à examiner d'autres nouveaux résultats au cours des trois années de cette thèse. Parmi les travaux existants, nous avons identifié deux approches existantes [96, 41] qui sont étroitement liées à notre travail. Par conséquent, nous avons utilisé ces deux approches comme référence pour évaluer les résultats de nos travaux.

Deuxièmement, et afin d'évaluer la généralisabilité de notre approche, nous avons mené sept cas d'étude appartenant à différents domaines, avec un nombre différent d'exigences logicielles et présentant différents styles d'écriture en langage naturel. Parmi les études de cas considérés, nous avons mené trois cas d'étude réels d'histoires d'utilisateurs dans le contexte du processus Scrum, et quatre cas d'étude en accès libre contenant des exigences fonctionnelles écrites en texte brut. En effet, la diversité des cas d'étude menés permet d'évaluer si notre approche fonctionne correctement pour différents domaines, différents nombres d'exigences ainsi que pour différents styles d'écriture.

Ensuite, nous avons défini un ensemble d'indicateurs de performance clés qui visent à évaluer si notre prototype fournit des résultats précis qui pourraient être applicables dans des contextes réalistes.

Les résultats de l'évaluation de l'approche de clustering sémantique et de l'approche de génération automatique du modèle préliminaire de conception révèlent des résultats relativement précis en termes de *précision*, de *rappel*, de *F-mesure* et de temps d'exécution. De plus, la comparaison des résultats obtenus avec les travaux existants [96], [41] montre que notre prototype a réussi à obtenir de meilleurs résultats pour le regroupement sémantique des exigences ainsi que pour la génération automatique du modèle de conception préliminaire.

Finalement, l'évaluation des résultats obtenus démontre que notre proposition pourrait être applicable et généralisée dans des contextes réalistes.

Appendix B

Technical demonstration of the developed tool

In this technical demonstration, we use the "CMS Company" (6.2.1) case study as an input to the tool in order to illustrate the execution steps of the prototype that we developed.

B.1 The semantic clustering approach

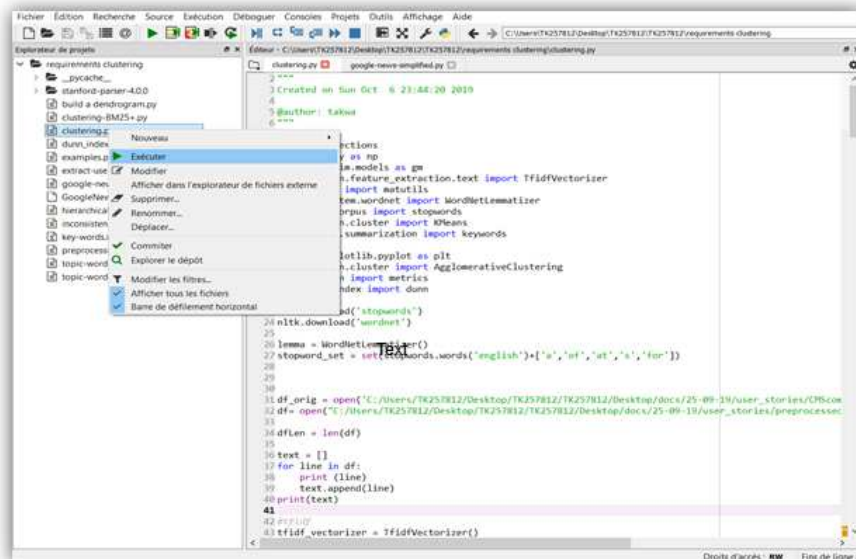
The first step of our prototype consists in generating the semantic clusters of requirements from the input "CMS Company" case study. This task is implemented in *Python 3.7.4*. In what follows we present the software and the libraries that we used to implement the proposed approach as well as a demonstration of the execution.

B.1.1 Software and libraries

Task	Software / Library
Integrated Development Environment (IDE)	Spyder IDE: https://www.spyder-ide.org/
word2vec model	https://code.google.com/archive/p/word2vec/
Tokenization	<code>nltk.word_tokenize</code>
TF-IDF vectorizer	<code>sklearn.feature_extraction.text.TfidfVectorizer</code>
Stemming	<code>nltk.stem.porter.PorterStemmer</code>
Pos-tagging	<code>nltk.import word_tokenize.pos_tag</code>
HAC algorithm	<code>sklearn.cluster.AgglomerativeClustering</code>
Dunn index	<code>dunn_index.dunn</code>
Lemmatization	<code>nltk.stem.wordnet.WordNetLemmatizer</code>
English stopwords	<code>nltk.corpus.stopwords.words('english')</code>
Clusters labelling	<code>gensim.summarization.keywords</code>

B.1.2 Execution

Firstly, we have to run the "clustering.py" file in order to generate the semantic clusters of requirements. As shown in Figure B.1, by executing this file for the "CMS Company" case study, we obtain a set of labelled clusters.



```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Oct 6 23:44:20 2019
4
5 @author: takwa
6 """
7
8 import sys
9 import os
10 import re
11 import numpy as np
12 import pandas as pd
13 import nltk
14 import nltk.tokenize
15 import nltk.tokenize.word_tokenize
16 import nltk.tokenize.punkt
17 import nltk.tokenize.trie
18 import nltk.tokenize.lstm
19 import nltk.tokenize.pickle
20 import nltk.tokenize.pickle
21 import nltk.tokenize.pickle
22 import nltk.tokenize.pickle
23 import nltk.tokenize.pickle
24 import nltk.tokenize.pickle
25
26 nltk.download('wordnet')
27
28 lemmas = WordNetLemmatizer()
29 stopwords_set = set(stopwords.words('english') + ['a', 'of', 'at', 's', 'for'])
30
31 df_orig = open('C:/Users/TK257812/Desktop/TK257812/Desktop/docs/25-09-19/user_stories/CMSCom
32 df = open('C:/Users/TK257812/Desktop/TK257812/Desktop/docs/25-09-19/user_stories/preprocess
33
34 dfLen = len(df)
35
36 text = []
37 for line in df:
38     print(line)
39     text.append(line)
40 print(text)
41
42 #TFIDF
43 tfidf_vectorizer = TfidfVectorizer()

```

Clustering execution




```

As an editor I need time zone support for setting publication and expiration dates for my content so I know what time a content item goes 'live' in my time zone.
As an editor I want to search full-text on content in the editor environment so that I can find relevant content faster and therefore work more productive
As an editor I want to maintain FAQs in multiple languages easily and in an intuitive way so that I can work productively
cluster "flash presentation filter"
As a system administrator I want to deploy filters run-time so that I can roll-out filter dependent functionality efficiently throughout DTAP
As a developer I want to check on all released flash versions supported by browsers so I can adjust my flash presentation towards new flash versions in particular.
As a developer I want to pass on variables to flash files directly.
As a developer I want to define presentation variants in my presentation configuration xml file so I can easily and efficiently deploy my presentation throughout DTAP
As a system administrator I want to define presentation variants in my presentation configuration xml file so I can easily and efficiently deploy my presentation throughout DT
cluster "page, url, attribute"
As a marketer I want to set the title attribute of a link so I can improve the SE ranking of the website
As a marketer I want to switch URLs when URL duplication occurs between pages so that I can solve conflicts between pages easily without having to search all pages in the tree.
As a marketer I want to create friendly URLs for my nested product pages so I can improve the SE ranking of the product section of my website
As a marketer I want to set the rel attribute of external links so I can make sure that SE bots do not affect SE rankings for pages with many external links
As a marketer I want to set canonical tags to individual pages so I can avoid duplicate content easily without having to set permanent redirects and thereby will improve the
As a marketer I want to solve URL conflicts immediately so I avoid not-friendly URLs and thereby will positively influence the overall SE ranking of the website
As an editor I want to assign page sections to pages using the current page section structure so I can easily and efficiently manage page sections assigned to pages
cluster "cms sitenanager ipv french"
As an IT manager I want IPv6 support by CMS SiteManager so I'm ensured that CMS SiteManager will work fine once IPv6 is applied into our hosting/network environment

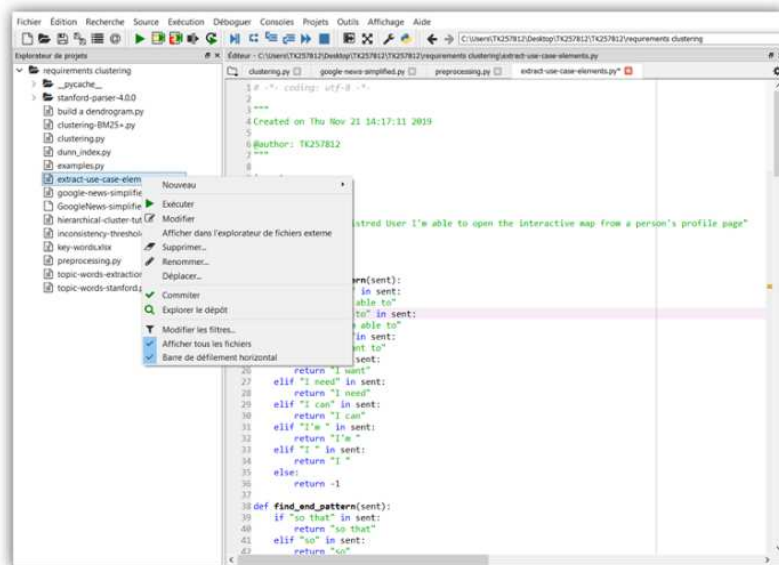
```

The generated clusters

Figure B.1: The generation of semantic clusters of requirements.

Then, we run the "extract-use-case-elements.py" as shown in Figure B.2. This step generates the

CSV file containing the relevant use case model elements as well as the cluster they belong to.

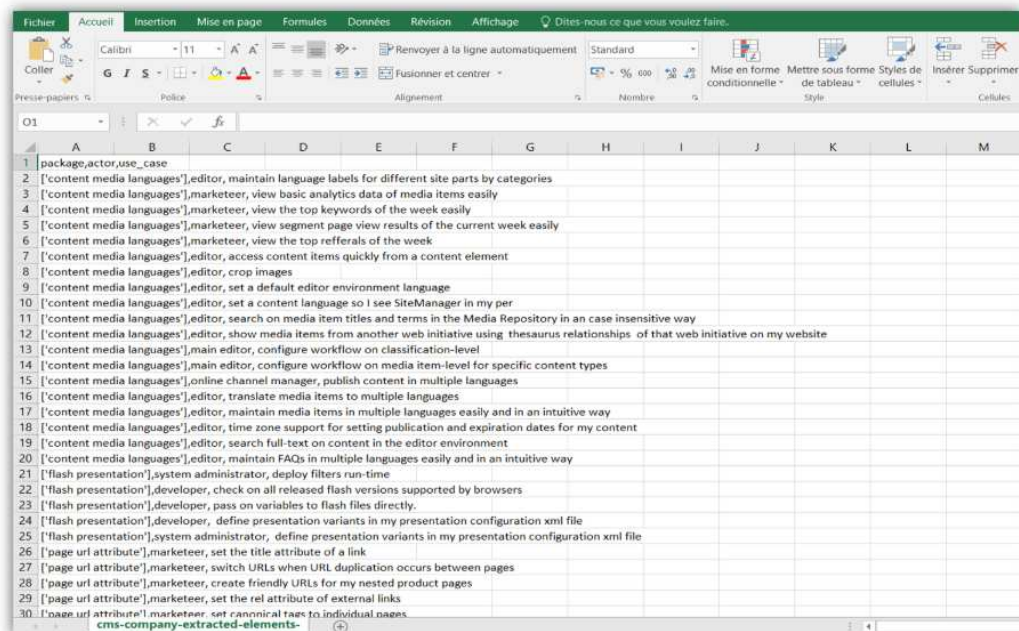


```

1 # -*- coding: utf-8 -*-
2
3
4 Created on Thu Nov 21 14:17:11 2019
5
6 @author: TK257812
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Elements extraction

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		package,actor,use_case											
2		[content media languages],editor, maintain language labels for different site parts by categories											
3		[content media languages],marketeer, view basic analytics data of media items easily											
4		[content media languages],marketeer, view the top keywords of the week easily											
5		[content media languages],marketeer, view segment page view results of the current week easily											
6		[content media languages],marketeer, view the top referrals of the week											
7		[content media languages],editor, access content items quickly from a content element											
8		[content media languages],editor, crop images											
9		[content media languages],editor, set a default editor environment language											
10		[content media languages],editor, set a content language so i see SiteManager in my per											
11		[content media languages],editor, search on media item titles and terms in the Media Repository in an case insensitive way											
12		[content media languages],editor, show media items from another web initiative using thesaurus relationships of that web initiative on my website											
13		[content media languages],main editor, configure workflow on classification-level											
14		[content media languages],main editor, configure workflow on media item-level for specific content types											
15		[content media languages],online channel manager, publish content in multiple languages											
16		[content media languages],editor, translate media items to multiple languages											
17		[content media languages],editor, maintain media items in multiple languages easily and in an intuitive way											
18		[content media languages],editor, time zone support for setting publication and expiration dates for my content											
19		[content media languages],editor, search full-text on content in the editor environment											
20		[content media languages],editor, maintain FAQs in multiple languages easily and in an intuitive way											
21		[flash presentation],system administrator, deploy filters run-time											
22		[flash presentation],developer, check on all released flash versions supported by browsers											
23		[flash presentation],developer, pass on variables to flash files directly.											
24		[flash presentation],developer, define presentation variants in my presentation configuration xml file											
25		[flash presentation],system administrator, define presentation variants in my presentation configuration xml file											
26		[page url attribute],marketeer, set the title attribute of a link											
27		[page url attribute],marketeer, switch URLs when URL duplication occurs between pages											
28		[page url attribute],marketeer, create friendly URLs for my nested product pages											
29		[page url attribute],marketeer, set the rel attribute of external links											
30		[page url attribute],marketeer, set canonical tags to individual pages.											

The CSV file containing the use case model elements

Figure B.2: The use case model elements extraction.

B.2 Automatic generation of the UML use case model

B.2.1 Software and libraries

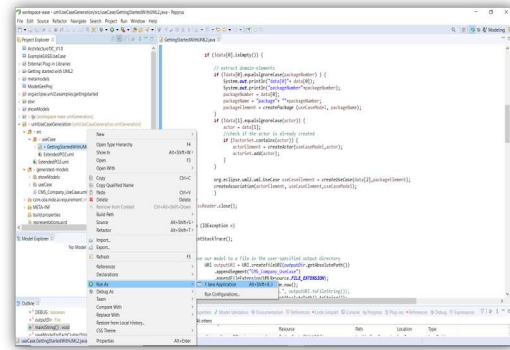
Task	Software / Library
Integrated Development Environment (IDE)	Papyrus: https://www.eclipse.org/papyrus/download.html
UML2 SDK tool	https://www.eclipse.org/modeling/mdt/uml2/docs/articles/Getting_Started_with_UML2/article.html
UML Factory	org.eclipse.uml2.uml.UMLFactory;
UML use case elements	org.eclipse.uml2.uml.Actor org.eclipse.uml2.uml.AggregationKind org.eclipse.uml2.uml.Association org.eclipse.uml2.uml.Classifier org.eclipse.uml2.uml.Component org.eclipse.uml2.uml.Enumeration org.eclipse.uml2.uml.EnumerationLiteral org.eclipse.uml2.uml.Generalization org.eclipse.uml2.uml.LiteralUnlimitedNatural org.eclipse.uml2.uml.Model org.eclipse.uml2.uml.Package org.eclipse.uml2.uml.PrimitiveType org.eclipse.uml2.uml.Property org.eclipse.uml2.uml.Stereotype org.eclipse.uml2.uml.Type

B.2.2 Execution

In order to automatically generate UML use case models, we run the mapping algorithm that takes as input the CSV containing the relevant model elements. The algorithm was developed in *java* using the UML2 SDK plugin as shown in Figure [B.3](#).

OS	A	B	C	D	E	F	G	H	I	J	K	L	M
1	package actor user_editor												
2	Content media language admin	maintain language items for different sites by company											
3	Content media language marketer	view basic analytics data of media items easily											
4	Content media language marketer	view the top keywords of the week easily											
5	Content media language marketer	view segment page view results of the current week easily											
6	Content media language marketer	view the top referrer of the week											
7	Content media language admin	access content items quickly from a content element											
8	Content media language admin	crop images											
9	Content media language admin	set a default editor environment language											
10	Content media language admin	set a content language on a web browser in my job											
11	Content media language admin	search on media item titles and items in the Media Repository in an easy incremental way											
12	Content media language admin	obtain media items from another web instance using the source relationship of that web instance on my website											
13	Content media language admin editor	configure workflow on classification level											
14	Content media language admin editor	configure workflow on media base level for specific content types											
15	Content media language admin	channel manager: publish content in multiple languages											
16	Content media language admin	handle media items in multiple languages											
17	Content media language admin	handle media items in multiple languages easily and in an intuitive way											
18	Content media language admin	take care of setting publication and expiration dates for my content											
19	Content media language admin	search for items in content in the editor environment											
20	Content media language admin	search for items in multiple languages easily and in an intuitive way											
21	Flash presentation system administrator	display items on time											
22	Flash presentation developer	check on all released flash versions supported by browsers											
23	Flash presentation developer	pass on variables to flash files directly											
24	Flash presentation developer	define presentation variants in my presentation configuration xml file											
25	Flash presentation system administrator	define presentation variants in my presentation configuration xml file											
26	Page url attribute marketer	set the title attribute of a link											
27	Page url attribute marketer	switch URLs when URL duplication occurs between pages											
28	Page url attribute marketer	create friendly URLs for my nested product pages											
29	Page url attribute marketer	set the rel attribute of external links											
30	Page url attribute marketer	set the rel attribute of external links											
31	Page url attribute marketer	set canonical tags to individual pages											
32	Page url attribute marketer	set canonical tags to individual pages											
33	Page url attribute marketer	set canonical tags to individual pages											
34	Page url attribute marketer	set canonical tags to individual pages											
35	Page url attribute marketer	set canonical tags to individual pages											
36	Page url attribute marketer	set canonical tags to individual pages											
37	Page url attribute marketer	set canonical tags to individual pages											
38	Page url attribute marketer	set canonical tags to individual pages											
39	Page url attribute marketer	set canonical tags to individual pages											
40	Page url attribute marketer	set canonical tags to individual pages											

The extracted use case model elements



Execution of the mapping algorithm



```

platform/resource/umlUseCaseGeneration/generated-models/CMS_CompanyUseCaseUml
├── <Model> useCaseModel
│   ├── <Package> package/content media languages
│   │   ├── <Actor> editor
│   │   ├── <Actor> marketer
│   │   ├── <Actor> main editor
│   │   └── <Actor> online channel manager
│   ├── <Package> package/flash presentation
│   │   ├── <Use Case> deploy filters run-time
│   │   ├── <Association> A_system administrator_deploy filters run-time
│   │   ├── <Use Case> check on all released flash versions supported by browsers
│   │   ├── <Association> A_developer_check on all released flash versions supported by browsers
│   │   ├── <Use Case> pass on variables to flash files directly
│   │   ├── <Association> A_developer_pass on variables to flash files directly
│   │   ├── <Use Case> define presentation variants in my presentation configuration xml file
│   │   ├── <Association> A_developer_define presentation variants in my presentation configuration xml file
│   │   ├── <Association> A_developer_define presentation variants in my presentation configuration xml file
│   │   ├── <Actor> system administrator
│   │   └── <Actor> developer
│   └── <Package> package/page url attribute
│       ├── <Use Case> set the title attribute of a link
│       ├── <Association> A_developer_set the title attribute of a link
│       ├── <Use Case> switch URLs when URL duplication occurs between pages
│       ├── <Association> A_developer_switch URLs when URL duplication occurs between pages
│       ├── <Use Case> create friendly URLs for my nested product pages
│       ├── <Association> A_developer_create friendly URLs for my nested product pages
│       ├── <Use Case> set the rel attribute of external links
│       ├── <Association> A_developer_set the rel attribute of external links
│       ├── <Use Case> set canonical tags to individual pages
│       ├── <Association> A_developer_set canonical tags to individual pages
│       ├── <Use Case> solve url conflicts immediately
│       ├── <Association> A_developer_solve url conflicts immediately
│       ├── <Use Case> assign page sections to pages using the current page section structure
│       └── <Association> A_developer_assign page sections to pages using the current page section structure
    
```

The generated UML use case model

Figure B.3: The UML use case model generation.

Bibliography

- [1] Standard glossary of software engineering terminology. 1990.
- [2] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider. What works better? a study of classifying requirements. *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 496–501, 2017.
- [3] S. Adolph, A. Cockburn, and P. Bramble. Patterns for effective use cases. 2002.
- [4] E. Agirre, D. M. Cer, M. T. Diab, A. Gonzalez-Agirre, and W. Guo. *sem 2013 shared task: Semantic textual similarity. In **SEM@NAACL-HLT*, 2013.
- [5] A. Albers and C. Zingel. Challenges of model-based systems engineering: A study towards unified term understanding and the state of usage of sysml. 2013.
- [6] A. Alebrahim. Framework for identifying meta-requirements. 2017.
- [7] Y. Amannejad, M. Moshirpour, B. Far, and R. Alhajj. From requirements to software design: An automated solution for packaging software classes. *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*, pages 36–43, 2014.
- [8] M. R. Anderberg. Cluster analysis for applications. 1973.
- [9] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer. Extracting domain models from natural-language requirements: approach and industrial evaluation. *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, 2016.
- [10] O. Badreddin, R. Khandoker, A. Forward, O. Masmali, and T. Lethbridge. A decade of software design and modeling: A survey to uncover trends of the practice. *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2018.

- [11] A. Bahill, B. Bentz, and F. F. Dean. Discovering system requirements. 1996.
- [12] I. S. Bajwa. Object oriented software modeling using nlp based knowledge extraction. 2009.
- [13] S. S. Balram. Perceptions of model-based systems engineering as the foundation for cost estimation and its implications to earned value management. 2012.
- [14] D. Bär, C. Biemann, I. Gurevych, and T. Zesch. Ukp: Computing semantic textual similarity by combining multiple content similarity measures. In *SemEval@NAACL-HLT*, 2012.
- [15] R. Barbosa, D. Janeiro, A. E. Silva, R. L. O. Moraes, and P. Martins. An approach to clustering and sequencing of textual requirements. *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, pages 39–44, 2015.
- [16] M. Baroni, G. Dinu, and G. Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL*, 2014.
- [17] J. Bass. Artefacts and agile method tailoring in large-scale offshore software development programmes. *Inf. Softw. Technol.*, 75:1–16, 2016.
- [18] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. In *J. Mach. Learn. Res.*, 2000.
- [19] S. Bird, E. Klein, and E. Loper. Natural language processing with python. 2009.
- [20] K. Bittner. Use case modeling. 2002.
- [21] G. Booch, J. Rumbaugh, and I. Jacobson. The unified modeling language user guide. *J. Database Manag.*, 10:51–52, 1999.
- [22] F. Brooks. No silver bullet essence and accidents of software engineering. *Computer*, 20:10–19, 1987.
- [23] B. Brügge and A. Dutoit. Object-oriented software engineering using uml, patterns, and java. 2009.
- [24] J. Cabot, R. Clarisó, M. Brambilla, and S. Gérard. Cognifying model-driven software engineering. In *STAF Workshops*, 2017.
- [25] E. Canedo and B. Mendes. Software requirements classification using machine learning algorithms. *Entropy*, 22:1057, 2020.
- [26] A. Casamayor, D. Godoy, and M. Campo. Functional grouping of natural language requirements for assistance in architectural software design. *Knowl. Based Syst.*, 30:78–86, 2012.

- [27] M. Chami and J. Bruel. A survey on mbse adoption challenges. 2018.
- [28] M. Chami, A. Aleksandraviciene, A. Morkevicius, and J. Bruel. Towards solving mbse adoption challenges: The d3 mbse adoption toolbox. 2018.
- [29] G. Chowdhury. Natural language processing. *Annu. Rev. Inf. Sci. Technol.*, 37:51–89, 2003.
- [30] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. Non-functional requirements in software engineering. In *International Series in Software Engineering*, 2000.
- [31] P. Clarkson and C. Eckert. Design process improvement : a review of current practice. 2005.
- [32] M. Cohn. User stories applied: For agile software development. 2004.
- [33] D. K. Deeptimahanti and M. Babar. An automated tool for generating uml models from natural language requirements. *2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 680–682, 2009.
- [34] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the Association for Information Science and Technology*, 41: 391–407, 1990.
- [35] S. Delisle, K. Barker, and I. Biskri. Object-oriented analysis : Getting help from robust computational linguistic tools. 1999.
- [36] D. Dubin. The most influential paper gerard salton never wrote. *Libr. Trends*, 52:748–764, 2004.
- [37] J. C. Dunn. Well-separated clusters and optimal fuzzy partitions. 1974.
- [38] T. Dybå and T. Dingsøy. Empirical studies of agile software development: A systematic review. *Inf. Softw. Technol.*, 50:833–859, 2008.
- [39] J. Eckhardt, A. Vogelsang, and D. Fernández. Are "non-functional" requirements really non-functional? an investigation of non-functional requirements in practice. *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 832–842, 2016.
- [40] M. Elallaoui, K. Nafil, and R. Touahni. Automatic generation of uml sequence diagrams from user stories in scrum process. *2015 10th International Conference on Intelligent Systems: Theories and Applications (SITA)*, pages 1–6, 2015.
- [41] M. Elallaoui, K. Nafil, and R. Touahni. Automatic transformation of user stories into uml use case diagrams using nlp techniques. In *ANT/SEIT*, 2018.

- [42] S. Faulk. Software requirements: A tutorial,. 1995.
- [43] S. Fernando and M. Stevenson. A semantic similarity approach to paraphrase detection. 2008.
- [44] M. Galster, A. Eberlein, and L. Jiang. Structuring software requirements for architecture design. *2013 20th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*, pages 119–128, 2013.
- [45] A. Ghazarian. Characterization of functional software requirements space: The law of requirements taxonomic growth. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 241–250, 2012. doi: 10.1109/RE.2012.6345810.
- [46] A. Ghazarian, M. S. Tehrani, and A. Ghazarian. A software requirements specification framework for objective pattern recognition: A set-theoretic classification approach. In *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, pages 211–220, 2011. doi: 10.1109/ICECCS.2011.28.
- [47] F. Gilson and C. Irwin. From user stories to use case scenarios towards a generative approach. *2018 25th Australasian Software Engineering Conference (ASWEC)*, pages 61–65, 2018.
- [48] V. Gudivada, A. Apon, and J. Ding. Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. 2017.
- [49] J. Hallqvist and J. Larsson. Introducing mbse by using systems engineering principles. 2016.
- [50] L. Han, A. L. Kashyap, T. Finin, J. Mayfield, and J. Weese. UMBC_EBIQUITY-CORE: Semantic textual similarity systems. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 44–52, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics. URL <https://www.aclweb.org/anthology/S13-1005>.
- [51] H. M. Harmain and R. Gaizauskas. Cm-builder: an automated nl-based case tool. *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering*, pages 45–53, 2000.
- [52] H. Herchi and W. B. Abdessalem. From user requirements to uml class diagram. *ArXiv*, abs/1211.0713, 2012.
- [53] A. Hess, P. Diebold, and N. Seyff. Understanding information needs of agile teams to improve requirements communication. *Journal of Industrial Information Integration*, 14:3–15, 2019.

- [54] A. Hoggood. The state of artificial intelligence. *Adv. Comput.*, 65:3–77, 2005.
- [55] A. Hotho, A. Nürnberger, and G. Paass. A brief survey of text mining. *LDV Forum*, 20:19–62, 2005.
- [56] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *NIPS*, 2014.
- [57] INCOSE. A world in motion: Systems engineering vision 2025. 2014.
- [58] A. Islam and D. Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2:1 – 25, 2008.
- [59] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. Object-oriented software engineering - a use case driven approach. In *TOOLS*, 1993.
- [60] H. Jalab and Z. M. Kasirun. Towards requirements reuse: Identifying similar requirements with latent semantic analysis and clustering algorithms. 2014.
- [61] V. Jijkoun and M. Rijke. Recognizing textual entailment using lexical similarity. 2005.
- [62] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 60:493–502, 1988.
- [63] W. B. A. Karaa, Z. B. Azzouz, A. Singh, N. Dey, A. Ashour, and H. B. Ghézala. Automatic builder of class diagram (abcd): an application of uml generation from functional requirements. *Software: Practice and Experience*, 46:1443 – 1458, 2016.
- [64] M. Kassab. The changing landscape of requirements engineering practices over the past decade. *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 1–8, 2015.
- [65] T. Kenter and M. Rijke. Short text similarity with word embeddings. *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015.
- [66] Y. Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.
- [67] D. Kolovos, L. M. Rose, N. Matragkas, R. Paige, E. Guerra, J. S. Cuadrado, J. Lara, I. Ráth, D. Varró, M. Tisi, and J. Cabot. A research roadmap towards achieving scalability in model driven engineering. In *BigMDE '13*, 2013.
- [68] A. Kontostathis and W. Pottenger. A framework for understanding latent semantic indexing (lsi) performance. *Inf. Process. Manag.*, 42:56–73, 2006.

- [69] P. Krutchen. The rational unified process: An introduction. 2000.
- [70] A. V. Lamsweerde. Requirements engineering - from system goals to uml models to software specifications. 2009.
- [71] R. Lebret and R. Collobert. Word embeddings through hellinger pca. *ArXiv*, abs/1312.5542, 2014.
- [72] R. Löffler, B. Güldali, and S. Geisen. Towards model-based acceptance testing for scrum. *Softwaretechnik-Trends*, 30, 2010.
- [73] G. Lucassen, F. Dalpiaz, J. M. V. D. Werf, and S. Brinkkemper. Visualizing user story requirements at multiple granularity levels via semantic relatedness. In *ER*, 2016.
- [74] G. Lucassen, F. Dalpiaz, J. V. D. Werf, and S. Brinkkemper. The use and effectiveness of user stories in practice. In *REFSQ*, 2016.
- [75] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. 1967.
- [76] A. Mahmoud and G. Williams. Detecting, classifying, and tracing non-functional software requirements. *Requirements Engineering*, 21:357–381, 2016.
- [77] C. D. Manning and H. Schütze. Foundations of statistical natural language processing. In *SGMD*, 2002.
- [78] C. D. Manning, P. Raghavan, and H. Schütze. Introduction to information retrieval. 2005.
- [79] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *ACL*, 2014.
- [80] K. Meinke and A. Bennaceur. Machine learning for software engineering: Models, methods, and applications. *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, pages 548–549, 2018.
- [81] L. Mich. NI-oops: from natural language to object oriented requirements using the natural language processing system lolita. *Nat. Lang. Eng.*, 2:161–187, 1996.
- [82] R. Mihalcea and P. Tarau. Textrank: Bringing order into text. In *EMNLP*, 2004.
- [83] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, 2006.

- [84] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *ICLR*, 2013.
- [85] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *ArXiv*, abs/1310.4546, 2013.
- [86] P. R. More and R. Phalnikar. Generating uml diagrams from natural language specifications. *International Journal of Applied Information Systems*, 1:19–23, 2012.
- [87] G. Mussbacher, D. Amyot, R. Breu, J. Bruel, B. Cheng, P. Collet, B. Combemale, R. France, R. Heldal, J. Hill, J. Kienzle, M. Schöttle, F. Steimann, D. R. Stikkolorum, and J. Whittle. The relevance of model-driven engineering thirty years from now. In *MoDELS*, 2014.
- [88] N. R. C. of Italy. Natural language requirements dataset. URL <http://fmt.isti.cnr.it/nlreqdataset/>.
- [89] S. P. Overmyer, B. Lavoie, and O. Rambow. Conceptual modeling through linguistic analysis using lida. *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pages 401–410, 2001.
- [90] H. Pérez-González and J. Kalita. Automatically generating object models from natural language analysis. In *OOPSLA '02*, 2002.
- [91] M. A. Rahman, M. A. Haque, M. H. Tawhid, and M. Siddik. Classifying non-functional requirements using rnn variants for quality software development. *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, 2019.
- [92] A. Rajaraman and J. Ullman. Mining of massive datasets. 2011.
- [93] A. Rashwan, O. Ormandjieva, and R. Witte. Ontology-based classification of non-functional requirements in software specifications: A new corpus and svm-based classifier. *2013 IEEE 37th Annual Computer Software and Applications Conference*, pages 381–386, 2013.
- [94] J. Rech and K. Althoff. Artificial intelligence and software engineering: Status and future trends. *Künstliche Intell.*, 18:5–11, 2004.
- [95] H. Romesburg. Cluster analysis for researchers. 1984.
- [96] H. E. Salman, M. Hammad, A.-D. Seriai, and A. Al-Sbou. Semantic clustering of functional requirements using agglomerative hierarchical clustering. *Inf.*, 9:222, 2018.

- [97] G. Salton. The smart retrieval system—experiments in automatic document processing. 1971.
- [98] G. Salton. A new comparison between conventional indexing (medlars) and automatic text processing (smart). *J. Am. Soc. Inf. Sci.*, 23:75–84, 1972.
- [99] G. Salton and M. McGill. Introduction to modern information retrieval. 1983.
- [100] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18:613–620, 1975.
- [101] J. Sayyad Shirabad and T. Menzies. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada, 2005. URL <http://promise.site.uottawa.ca/SERepository>.
- [102] E.-M. Schön, J. Thomaschewski, and M. J. E. Cuaresma. Agile requirements engineering: A systematic literature review. *Comput. Stand. Interfaces*, 49:79–91, 2017.
- [103] P. Singh, D. Singh, and A. Sharma. Classification of non-functional requirements from srs documents using thematic roles. *2016 IEEE International Symposium on Nanoelectronic and Information Systems (iNIS)*, pages 206–207, 2016.
- [104] P. Singh, D. Singh, and A. K. Sharma. Rule-based system for automated classification of non-functional requirements from requirement specifications. *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 620–626, 2016.
- [105] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, 2013.
- [106] I. Sommerville. Artificial intelligence and systems engineering. 1993.
- [107] I. Sommerville. Software engineering, 10/e. 2020.
- [108] I. Sommerville and P. Sawyer. Requirements engineering: A good practice guide. 1997.
- [109] S. Stavru. A critical examination of recent industrial surveys on agile method usage. *J. Syst. Softw.*, 94:87–97, 2014.
- [110] M. H. C. P. M. System. URL <https://bscs143.files.wordpress.com/2015/11/requirement-mhc-pms.docx>.
- [111] L. Tripp, E. Byrne, P. Croll, P. Deweese, R. Fralick, M. Ginsberg-Finner, J. Harauz, M. Henley, D. Lawrence, D. Maibor, R. Milovanovic, J. Moore, T. Niesen, D. Rilling, T. Rout, R. Schmidt,

N. Schneidewind, D. Schultz, B. Sherlund, P. Voldner, R. Wade, S. Ali, T. K. Atchinson, M. Auguston, R. Barry, L. Beltracchi, H. R. Berlack, R. E. Biehl, M. A. Blackledge, S. Bologna, J. Borzovs, K. L. Briggs, M. Scott, B. M. Caldwell, J. E. Cardow, E. A. Carrara, L. Catchpole, K. Chan, A. Cicu, T. Clarke, S. Clermont, R. Coleman, V. Lee, C. W. W. G. Cozens, G. T. Daich, G. Darnton, T. Daughtrey, B. K. Derganc, J. Do, E. S. Dow, C. Dragstedt, S. Eagles, C. Ebert, L. Egan, R. Fairley, J. Fendrich, J. Forster, K. Fortenberry, E. Freund, R. Fries, R. Fujii, A. N. Ghannam, G. John, J. Glynn, L. Gonzalez-Sanz, D. A. Gunther, J. Gustafson, J. Hagar, R. T. Harauz, H. Harley, W. J. Hecht, M. Heßey, M. Hein, M. Heinrich, D. Henley, J. W. Herrmann, J. Horch, P. L. Huller, G. K. Hung, F. V. Jackelen, W. S. Jorgensen, G. Junk, R. Kambic, R. S. Karcich, J. S. Kenett, R. J. Kerner, D. L. Kierzyk, S. Knirk, T. M. Koenig, J. B. Kurihara, J. Lane, L. Dennis, C. Fang, W. M. Lim, J. J. Lively, D. Longbucco, J. Look, S. Lord, D. J. Magee, H. Maibor, R. A. Mains, T. Martin, M. Matsubara, P. McAndrew, C. Mccray, J. W. McMacken, B. Mersky, A. Michael, C. Miller, J. W. Modell, P. Moore, M. L. Navrat, I. P. Olson, A. Pal, P. T. Polack, L. S. Poon, K. R. Przybylski, A. D. Ptack, D. Reilly, A. P. Rilling, H. Sage, S. R. Sandmayr, H. Schach, N. P. Schaefer, D. J. Schneidewind, L. A. Schultz, R. W. Selmon, D. M. Shillato, C. A. Siefert, J. M. Singer, R. S. Sivak, N. M. Sky, M. Smith, H. M. Smyre, A. R. Sneed, D. W. Sorkowitz, L. Sova, J. Spotorno, F. J. Stesney, C. Strauss, S. Brown, T. Toru, H. Richard, B. Thayer, P. Thomas, T. J. Trelue, G. D. Urbanowicz, U. Venables, D. D. Voges, D. Walden, W. M. Wallace, J. Walsh, C. Walz, S. A. Swhite-Partain, P. Whitmire, P. Wolfgang, N. C. Work, J. Yopconka, G. Zalewski, P. F. Zimmerman, Zoll, R. J. Holleman, D. Heirman, V. Chair, J. Gorman, M. Emeritus, V. E. Zelenty, S. Aggarwal, C. R. Camp, J. T. Carlo, G. R. Engmann, H. E. Epstein, T. Garrity, R. D. Garzón, J. Gurney, J. Isaak, L. G. Johnson, R. A. Kennelly, E. Kiener, J. L. Koepfnger, S. Lambert, J. Logothetis, D. C. Loughry, L. B. McClung, R. C. Petersen, G. H. Peterson, J. Posey, G. S. Robinson, H. Weinrich, and D. Zipse. IEEE recommended practice for software requirements specifications. 1993.

- [112] D. Walden. Systems engineering handbook : a guide for system life cycle processes and activities. 2015.
- [113] Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel. Unifying and extending user story models. In *CAiSE*, 2014.
- [114] T. A. Wiggerts. Using clustering algorithms in legacy systems modularization. *Proceedings of the Fourth Working Conference on Reverse Engineering*, pages 33–43, 1997.
- [115] M. Younas, D. N. A. Jawawi, I. Ghani, and M. A. Shah. Extraction of non-functional requirement using semantic similarity distance. *Neural Computing and Applications*, 32:7383–7397, 2019.

[116] T. Yue, L. Briand, and Y. Labiche. atoucan: An automated framework to derive uml analysis models from use case models. *ACM Trans. Softw. Eng. Methodol.*, 24:13:1–13:52, 2015.

[117] M. L. Zepeda-Mendoza and O. Resendis-Antonio. *Hierarchical Agglomerative Clustering*, pages 886–887. Springer New York, New York, NY, 2013. ISBN 978-1-4419-9863-7. doi: 10.1007/978-1-4419-9863-7_1371. URL https://doi.org/10.1007/978-1-4419-9863-7_1371.

[118] D. Zowghi and C. Coulin. Requirements elicitation: A survey of techniques, approaches, and tools. 2005.

Titre: Combler le fossé entre les exigences du système exprimées en langage naturel et les modèles de conception d'architecture

Mots clés: exigence du système, traitement automatique du langage naturel, apprentissage automatique, conception d'architecture, UML

Résumé : Au cours des dernières années, les contraintes liées à la conception des systèmes évoluent de plus en plus et nécessitent l'intégration d'un plus grand nombre d'intervenants dans les projets. Par conséquent, les systèmes modernes deviennent de plus en plus complexes. L'ingénierie des systèmes basée sur les modèles (MBSE) est reconnue pour favoriser une vision holistique de la conception et permettre une architecture système maintenable et de haute qualité. Cependant, les modèles de conception d'architecture sont toujours extraits manuellement à partir des exigences système, ce qui est devenu une tâche fastidieuse, chronophage et sujette aux erreurs. En particulier, la croissance exponentielle du nombre d'exigences système soulève des difficultés pour gérer ces exigences manuellement et avoir une vision claire et cristalline des attentes et de la portée du système à concevoir. Le manque d'expertise humaine ainsi que des outils d'automatisation puissants sont souvent cités comme les principaux obstacles clés qui ralentissent encore l'adoption de l'approche MBSE et présentent des obstacles importants pour démontrer son retour sur investissement (ROI).

De nos jours, les applications de l'intelligence artificielle (IA) sont de plus en plus présentes dans notre vie quotidienne. En fait, les techniques d'IA associées à une technologie appropriée ont permis aux systèmes de percevoir, de prédire et d'agir pour aider les humains dans un large éventail d'applications. Par conséquent, les progrès de l'IA peuvent apporter une grande valeur pratique pour atténuer les défis soulevés par l'adoption du MBSE au niveau de la transition des exigences systèmes exprimées en langage naturel au

modèles d'architecture exprimés en UML.

Dans cette thèse, nous avons proposé un nouveau flux de composants d'IA, y compris leur paramétrage spécifique, permettant l'automatisation de la transition des exigences exprimées en langage naturel vers un modèle préliminaire de conception d'architecture en UML.

Premièrement, nous avons proposé une solution de clustering qui aide à décomposer le système complexe en sous-systèmes plus petits en fonction de la similarité sémantique des exigences système. La solution de clustering proposée est basée sur un module de calcul de similarité sémantique qui analyse l'information sémantique des mots ainsi que des énoncés d'exigences pour chaque paire d'exigences en utilisant le modèle de prolongements lexicaux neuronaux word2vec. Un ensemble de groupes d'exigences sémantiquement similaires sont ainsi générés désignant les sous-systèmes identifiés et, qui aident à réduire la complexité du système cible. Ensuite, nous avons proposé un extracteur de modèle qui extrait à partir de chaque groupe d'exigences identifié (c.-à-d., sous-système), les éléments pertinents qui sont nécessaires pour construire le modèle de paquets de cas d'utilisation exprimé en UML en utilisant un ensemble d'heuristiques de Traitement Automatique du Langage Naturel (TALN). Enfin, nous avons proposé une opération de mapping qui transforme les éléments pertinents extraits en leurs correspondants dans le modèle de paquet de cas d'utilisation cible exprimé en UML. Notre travail a été prototypé sous Papyrus et évalué sur plusieurs cas d'étude comprenant différents types d'exigences système exprimées en langage naturel.

Title: Bridging the gap between natural language system requirements and architecture design models

Keywords: system requirement, natural language processing, machine learning, architecture design, UML

Abstract: In recent years, system design constraints evolve more and more requiring to embed more stakeholders in the projects. Consequently, modern software projects are becoming many times larger and more complex than in the past.

Model-Based Systems Engineering (MBSE) methods are on their side recognized to foster holistic view of design and empower high quality and maintainable software architecture. However, architecture design models are always extracted manually by engineers, which became a tedious, time-consuming and error prone task. Especially, the exponential growth of the number of system requirements raises difficulties in managing the requirements manually and having a clear crystal view of the expectation and scope of the system to be designed. The lack of human expertise as well as powerful automation tools are often cited as the main key barriers that still slow down the spread of the MBSE approach and present significant hurdles to demonstrate its Return On Investments (ROI).

Recently, Artificial Intelligence (AI) has been receiving intensive attention and its applications have made their way into products in our daily life. In fact, AI techniques together with suitable technology have enabled systems to perceive, predict, and act in assisting humans in a wide range of applications. Hence, it stands to reason that advances in AI can bring great practical value to mitigate some of the challenges raised by the adoption of MBSE.

In this thesis, we contributed a first step towards applying AI for MBSE to optimize the

adoption of MBSE and resolve some of its challenges. Specifically, we proposed a new flow of Machine Learning (ML) and Natural Language Processing (NLP) components, empowering the automation to go from natural language requirements towards a preliminary UML architecture design model including a package breakdown model denoting the system's decomposition.

First, we proposed a clustering solution that helps to decompose the complex system into smallest sub-systems based on the semantic similarity of early requirements. The core of the proposed clustering solution is a semantic similarity computation module that analyzes the semantic information of both the words and requirement statements of each pair of requirements using the neural word embedding model *word2vec*. Accordingly, a set of clusters of similar requirements are generated denoting the identified sub-systems and hence, helping to reduce the complexity of the target system. Then, we proposed a model extractor that extracts from each identified cluster (i.e., sub-system), the relevant elements that are needed to build the UML use-case package break-down model using a set of NLP heuristics. Finally, we proposed a mapping operation that programmatically maps the extracted model elements into their corresponding ones in the target UML use-case package model.

Our proposal was prototyped on *Papyrus* and evaluated on several case studies encompassing different types of natural language software requirements.