Universiteit Leiden

ICT in Business and the Public Sector

A System for Preprocessing Requirements Documents for Automatic UML Modelling

Name: Martijn B.J. Schouten
Student-no: 2670798

Date: 01/07/2022

1st supervisor: Dr. G.J. Ramackers
2nd supervisor: Dr. S. Verberne

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Current approaches to natural language processing of software requirements documents restrict their input to documents that are relevant to specific types of models only, such as domain or process-focused models. Such input texts do not reflect real-world requirements documents. To address this issue, we propose a pipeline for preprocessing such requirements documents at the conceptual level, for subsequent automatic generation of class, activity, and use case models in the Unified Modelling Language (UML) downstream. Our pipeline consists of three steps. Firstly, we implement entity-based extractive summarization of the raw text to enable highlighting certain parts of the requirements that are of interest to the modelling goal. Secondly, we develop a rule-based bucketing method for selecting sentences into a range of 'buckets' for transformation into their corresponding UML models. Finally, in order to demonstrate the effectiveness of supervised machine learning models on requirements texts, a sequence labelling model is applied to generate meta data pertaining to the longstanding problem of distinguishing classes from attributes in the context of domain modelling. In order to enable this step of our pipeline, we address the lack of available annotated data by labelling the widely used PURE requirements dataset on a word level by tagging classes and attributes within the texts. Our three-step solution has been implemented in the LIACS Prose to Prototype SE development environment.

Acknowledgements

First of all, I would like to thank my first supervisor, Dr. Guus Ramackers for supporting me throughout this project, for giving me the inspiration needed for the topic of this research, and for the countless meetings we had to get the content of my research just right. Second of all, I would like to thank my second supervisor, Dr. Suzan Verberne, for providing me with much-needed advice regarding my options for NLP in this project, getting me towards a workable solution, and for the help with proofreading and improving the paper that has been published about this research. Next to that, I would like to thank Pepijn Griffioen en Willem-Pieter van Vlokhoven for their help with the technical implementation of this work in the ngUML application. I also want to extend my thanks to Prof. Dr. Michel Chaudron for his advice and perspectives on the project over the course of the last year in our weekly meetings. Lastly, I would like to thank my girlfriend, Malin Inzinger, for supporting me throughout this whole project, especially at the end where I really needed someone to pull me through the final stretch. This thesis ended with a bet between her and me: she was allowed to decide my next haircut if I did not finish this thesis on time. Luckily, I won the bet. I told you I could do it.

Contents

Parts of this work have been published in two research papers that have been released during this thesis as a direct result of the research findings. The first paper has been released in 2021 as part of the Proceedings of the 23rd ACM/IEEE International Conference on Model-Driven Engineering Languages and Systems and is titled "**From Prose to Prototype: Synthesising Executable UML Models from Natural Language**" [80]. This paper describes the Prose to Prototype project in more detail and gives a high-level overview of the context of this thesis. The second paper encompasses all research findings of this thesis and is published in 2022 as part of the Proceedings of the 27th International Conference on Natural Language & Information Systems. The title is "**Preprocessing Requirements Documents for Automatic UML Modelling**" [89].

1 Introduction

When engineering software systems, the main assumption is that the computational environment is predictable and fully specifiable. However, in the current world, systems are increasingly spread out over parts and layers built by many organizations, in different environments, and require cooperation from human operators. As a result, software engineering is increasingly confronted with uncertainty and complexity [31].

Because of that, the process of defining software requirements, i.e. deciding what to build, has become harder [14]. A good understanding of requirements is the basis of creating systems that satisfy the expectations of stakeholders. Early construction of a software-system architecture is helpful for the discovery of further requirements and constraints, feasibility and determining alternatives for implementation [69]. For this, stakeholder involvement is key. After all, a lack of stakeholder involvement is the number one reason for software projects to run into difficulties [73]. Therefore, achieving a higher level of stakeholder involvement has been a much-researched topic in requirements engineering (RE).

Many possibilities to enhance communication and involvement have been posed, ranging from wikis for stakeholder collaboration to gamification aspects [22, 55], but a definite solution to solidifying early requirements is not offered, nor does current research answer the problem of bridging the gap between stakeholders and system architects in the early stages of requirements engineering. But what if the solution does not lie with the stakeholders but with the process itself?

A new research area uses natural language processing (NLP) to assist software requirements analysis. This can help plan out software projects early on [49, 34, 80]. As requirements documents and domain descriptions are oftentimes provided in natural language, structuring this knowledge can form the basis for the software development process. This early-stage modelling allows individual stakeholders to conceptualize their visions faster, and human error in communicating requirements can be dealt with immediately. With this approach, stakeholders can intervene in the development process at a stage where the detection of errors does not escalate exponentially, as it does in later stages of development.

Several methods have been proposed for generating UML models from requirements texts. However, most of these previous implementations rely on structured input texts, which are not representative of real-world requirements documents. In addition, real world-world requirements texts mix specifications for different kinds of UML models in the same document. Furthermore, distinctions between UML elements, for instance, classes versus attributes in class modelling, are not explicitly identified.

1.1 Research objectives

In this work, we address these limitations by creating an interactive preprocessing pipeline for raw requirements texts that, with the intervention of the human modeller, outputs structured and separated texts that can be used to generate UML models.

To aid downstream model generation, the model suggest metadata alongside the output text of the pipeline. For now, this metadata is limited to suggestions

for distinguishing classes and attributes, which is regarded as a longstanding research problem within automatic class model generation [80]. However, with this approach, we suggest an architecture to generate metadata to target specific downstream NLP transformation modelling issues in the future.

In summary, the main research objective is as follows:

> **Develop a pipeline-like system that is able to preprocess real-world requirements texts describing a system and process it into uniform *class*, *use case* and *activity texts* as input for downstream NLP to UML transformation**.

More specifically, this main objective can be broken up into four sub-objectives:

Sub-objective 1 Process real-world requirements texts describing any system in any context in a wide variety of expressions, specifically by removing textual elements that do not carry any information useful for systems design.

Sub-objective 2 Find patterns or keywords in sentences or paragraphs that are useful for creating activity-, class-, and use case models, and can be used to create buckets of useful sentences for each of these models.

Sub-objective 3 Utilize NLP techniques in combination with machine learning to aid in the generation of metadata useful for downstream modelling.

Sub-objective 4 Allow for a human-in-the-loop approach where users of the pipeline are able to interact and steer decision-making at all steps of the process.

1.2 Overview of our methodology

In this thesis, we apply natural language processing techniques to structure raw requirements texts that are used in real-world modelling settings to use this new structure for class-, activity- and use case modelling "downstream", i.e. in automated modelling activities in steps after the pipeline of this project. Additionally, metadata is generated to assist class modelling.

This thesis follows the *research by design* methodology. According to Roggema (2016), research by design is a type of academic investigation through which design is explored as a method of inquiry, through the development of a project. The results of this project are inherently tangible: a pipeline is the main delivery, which can be used in a modern application landscape to preprocess and structure requirements texts. This thesis project shows the various ways in which design and research are interconnected when new knowledge is produced by its implementation [82].

First, literature research is conducted to sharpen and extend the main research objective and its sub-objectives mentioned in Section 1.1.

To validate the effectiveness of the pipeline, the performance of the methods is tested against 5 examples of real-world requirements texts from several sources that can be found in Appendix A. The validation requirements texts are selected so that they are as close as possible to real-word requirements texts, and span multiple contexts and industries, as laid out in the Sub-objective 1.

1.3 Outline

The structure of this thesis is as follows: in Section 2, a short overview of related work on UML and NLP will be discussed to set the general stage for the context of this research and provide readers with a short background in both disciplines.

Then, in Section 3, a conceptual design of the system will be discussed. In Section 4, the final system design will be discussed, including a technical design and description of the used technologies. After that, in Section 5, the findings of this research are shared through qualitative and quantitative results. To conclude, Section 6 describes the results and how they satisfy the research objectives and directions for future research.

2 Background

In this research, the intersection between natural language processing (NLP) and software design is explored. Therefore, this section is structured as follows: first, the unified modelling language and its application will be discussed. Then, the field of NLP and its recent developments are explored. Finally, research on the intersection of both concepts will be discussed.

2.1 Unified Modelling Language (UML)

Requirements define what stakeholders need from a potential new system and what the system must do in order to satisfy that need [109]. When measuring success in software systems as the degree to which it meets the purpose for which it was intended [70], a system that fully implements all requirements is a successful one. Therefore, a good overview of requirements forms the basis of software analysis and design, which in turn ensures that planned systems will truly reflect the requirements and achieve the quality desired by the stakeholders [60].

These requirements are typically represented in natural language [45]. More specifically, they are gathered in user requirements documents (URD), which oftentimes serve as a contractual agreement between users and developers [87]. According to Schneider et al. (1992), *"early detection and correction of faults in the URD is the key to keeping development costs down and to building correct, reliable software"*. This immediately highlights the main limitation of the URD: requirements specified in natural language can be ambiguous, incomplete and inconsistent. Furthermore, the understanding and interpretation of requirements can potentially be influenced by geographical, psychological and sociological factors [23]. Because of this, the requirements analysis process involves the creation of abstract models for visualization and comprehension of requirements [90]. In this space of visualizing the design of systems, UML has been the de-facto standard language [93].

UML is a graphically based notation developed by the Object Management Group to standardize describing software-oriented designs [78]. The language gives guidelines for modelling several different diagrams that contain a complete representation of a system from a given modelling perspective [66]. These representations can be made with a variety of available tools, such as IBM Rational Software Architect, MagicDraw, and Papyrus [83].

The diagrams are divided into three categories: static structure, behaviour and implementation diagrams. Static structure diagrams describe the structure of a system, while behaviour diagrams describe the dynamic parts. The final category, implementation diagrams, provides a lower level of abstraction by including information about source code for software projects [96].

For this research, the focus is on the two categories of UML models with the highest level of abstraction, namely static structure and behaviour diagrams. Specifically, within these categories, class, activity and use case models will be discussed, because of their favourability in describing systems [81], and because these three models generally cover all aspects of system design [79, 91, 17].

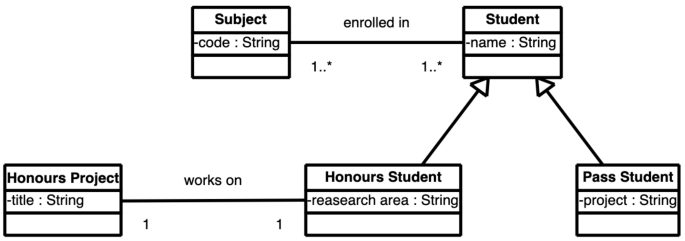


Figure 1: An example of a class diagram [61].

2.1.1 Class modelling

Using class diagrams, a modeller can describe the static structure of a system [96]. Naturally, this puts the class model in the category of static structure diagrams. This static structure is expressed through a collection of classes that are interconnected by a relationship. The relationships are also categorized by their multiplicity, which denotes how many objects are involved in the relationship [63]. Additionally, a diagram can display attributes and, when modelling software systems, operations of classes [96].

An example is shown in Figure 1. Classes are shown as rectangles, with relationships as lines between classes. These lines hold numbers on both lines that describe the multiplicity of the relationship.

This example describes a simple context of a university: the main class "Student" has an inheritance relationship with the classes "Honours Student" and "Pass Student", describing the types of students that are available [61].

2.1.2 Activity modelling

Activity diagrams are intended for modelling computational and business or organisational processes [104], and describe dynamic system behaviour. Therefore, activity diagrams are considered as belonging to the category of behaviour diagrams. In essence, they are flowcharts that give an overview of the flow of control between sequential or concurrent activities of a process [13].

This sequential nature can be observed in the example shown in Figure 2. This is the activity diagram associated with an ordering process. Activities that must be conducted by specific actors are modelled via action states in their swimming lanes. The complete process models all actions between the start state and the stop state. In this example, a focus is laid on controlling the flow through forks and joins: the synchronization of two or more concurrent flows of control, modelled using a thick black line [12].

When creating an activity model from requirements, the modeller should look for activities that take place over time, and operations that are passed among objects. These activities result in some action that results in a change in the state of a system or return of a value [12].

2.1.3 Use case modelling

Use case diagrams are useful for modelling and describing interactions between certain actors and a system from the actor's point of view. It gives a visual overview of what the actor does or gives the system and what it needs or gets

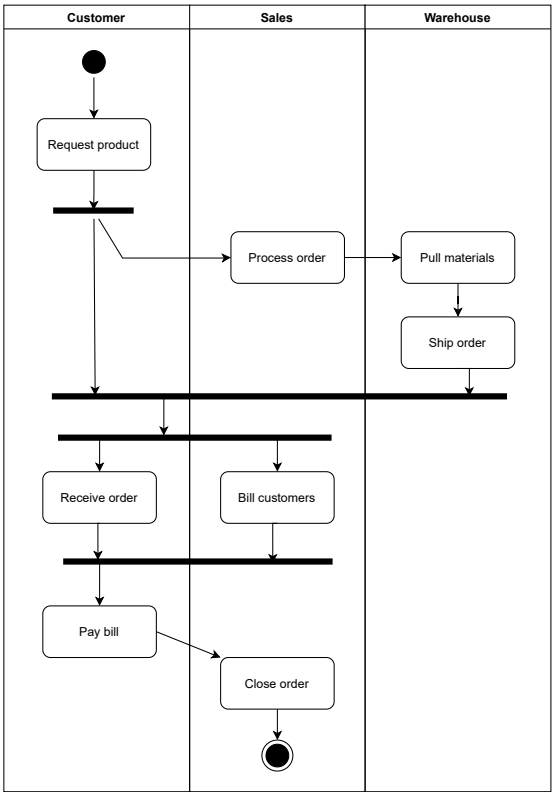


Figure 2: An example of an activity diagram [12].

from the system [60]. In essence, a use case is a sequence of actions a system can perform that creates an observable result with regard to a specific user [43]. Because use case diagrams capture and describe the dynamics of systems, they belong to the category of behaviour diagrams.

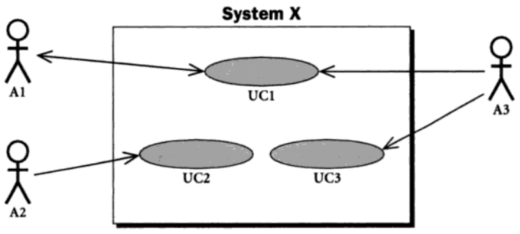


Figure 3: An example of a use case diagram [42].

In Figure 3, a very simple example of a use case diagram can be seen. Here, the modeller has identified three actors, A1, A2, and A3, and they each are able to perform certain use cases in System X, namely UC1, UC2 and UC3.

2.2 Natural Language Processing (NLP)

According to Liddy (2001), *"Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis to achieve human-like language processing for a range of tasks or applications"*. NLP has been through major stages, starting with rule-based systems in the 80s, to statistical methods, and currently, the extensive usage of neural networks and the dominance of Transformer architectures [19]. Because of the sheer size of the research field and the diversity of NLP applications, only techniques and developments that are deemed useful for answering the research questions of this thesis are described in the following sections.

2.2.1 Syntactic dependency trees

When processing text on a syntactic level using NLP methods, a representation is outputted that revels the structural dependency relationships between words [52]. One way of displaying these relationships is through the use of dependency trees. An example of a dependency tree is given in Figure 4. This example has been generated using the spaCy Python library [39].

These trees are oftentimes useful as part of a bigger solution to tackle NLP problems. For example, they have been used for word reorderings in machine translation [99, 75], to aid and replace the role of human labelling in the identification of domain terminology [53] and for more general tasks like opinion mining and sentiment analysis [92].

For this project, the dependencies between words are used to formulate rules to divide sentences into groups, together with lexico-semantic analysis.

2.2.2 Lexico-semantic analysis

Where syntax dependency trees only show grammatical structure, there are also techniques to extract semantics from sentences. Semantics describes the possible meanings of a sentence [52]. However, tackling meaning in sentences has been one of the tougher challenges in NLP. WordNet is an online lexical database that contains English nouns, verbs, adjectives and adverbs organized into hierarchical sets of synonyms that are then linked through lexico-semantic relations. These relations then determine the meaning of words [64]. At the top of this taxonomy, supersenses (or semantic fields) are defined, which are a finite set of top-level hypernyms. They are designed to be broad enough to encompass all nouns and verbs [88].

An overview of all supersenses is shown in Figure 5. Supersenses can be seen as classes, which are useful for capturing generalizations: one can abstract

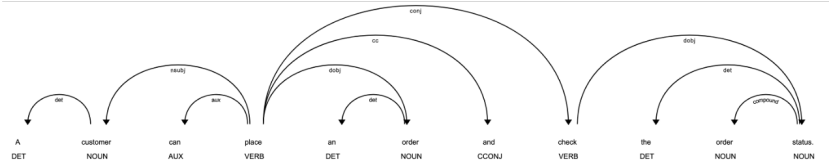
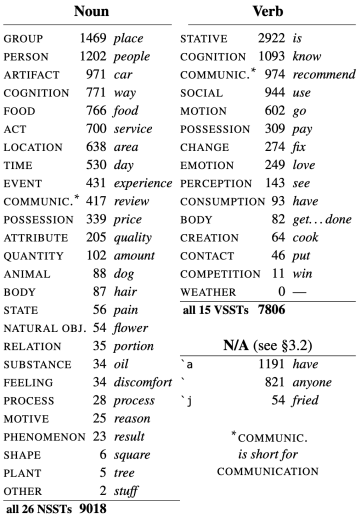


Figure 4: An example of a syntax dependency tree structure.



Noun			Verb		
GROUP	1469	*place*	STATIVE	2922	*is*
PERSON	1202	*people*	COGNITION	1093	*know*
ARTIFACT	971	*car*	COMMUNIC.*	974	*recommend*
COGNITION	771	*way*	SOCIAL	944	*use*
FOOD	766	*food*	MOTION	602	*go*
ACT	700	*service*	POSSESSION	309	*pay*
LOCATION	638	*area*	CHANGE	274	*fix*
TIME	530	*day*	EMOTION	249	*love*
EVENT	431	*experience*	PERCEPTION	143	*see*
COMMUNIC.*	417	*review*	CONSUMPTION	93	*have*
POSSESSION	339	*price*	BODY	82	*get…done*
ATTRIBUTE	205	*quality*	CREATION	64	*cook*
QUANTITY	102	*amount*	CONTACT	46	*put*
ANIMAL	88	*dog*	COMPETITION	11	*win*
BODY	87	*hair*	WEATHER	0	—
STATE	56	*pain*	**all 15 VSSTs**	**7806**	
NATURAL OBJ.	54	*flower*			
RELATION	35	*portion*	**N/A** (see §3.2)		
SUBSTANCE	34	*oil*	`a	1191	*have*
FEELING	34	*discomfort*	`	821	*anyone*
PROCESS	28	*process*	`j	54	*fried*
MOTIVE	25	*reason*			
PHENOMENON	23	*result*			
SHAPE	6	*square*			
PLANT	5	*tree*			
OTHER	2	*stuff*			
all 26 NSSTs	**9018**				

*COMMUNIC. *is short for* COMMUNICATION

Figure 5: A table containing all the supersenses for nouns and verbs [88].

away from individual words because words or verbs with similar meanings will be grouped in the same classes [46].

Supersenses have a broad application, specifically in aiding other NLP tasks: they have been used to assist the training process of neural machine translation [97], but also have shown to help in defining rules in a rule-based system for coreference resolution [71] and generating document keywords [41].

2.2.3 Summarization

According to Paulus et al. (2017), *"text summarization is the process of automatically generating natural language summaries from an input document while retaining the important points"*. Automatic summarization is often used for aiding legal research [62, 9], making patent filing more efficient [16] and in other sectors like healthcare [58, 27, 86, 32] and media [21, 94, 18]. Generally, these algorithms are divided into two types: extractive summarization, where parts of the input are copied, and abstractive summarization, where new phrases are generated [74].

Currently, most state-of-the-art abstractive summarization models are based on Transformer architectures [107, 51, 108]. Transformer architectures focus on attention mechanisms to create representations [98], making it incredibly useful for sequence-to-sequence predictions such as summaries.

Traditionally, adjusting the output of these models by an end-user has been largely impossible. However, there have been tries in research to allow for more customization of outputs of these models [103, 6, 105]. One of these models is the Customizable Abstractive Topic-based sequence-to-sequence Summarization (CATS) model. The model allows for summarizating in an abstractive way, while selectively focusing on a range of desired topics, chosen by the user. In essence, this means that summarization happens in two stages: first, the model retrieves a set of topics that are present in the text and presents those in the user. Second, the user returns topics of interest to the model and the model uses

these topics to generate a summary. The model is based on the encoder-decoder architecture combined with the concept of pointer networks, which enables a combination of copying words from the source text and generate words from a fixed vocabulary. Summarizations are produced using a novel concept called *topical attention*: over encoder-decoder training steps, parameters adapt to learn the topics of each document [6].

2.2.4 fastText

The representation of words has been a central topic in NLP. A word embedding is a dense, distributed, continuous vector representation of a word, where more similar words are closer to each other in the vector space than less similar words. Preferably, this representation and its location in the vector space captures semantic and syntactic similarities and differences between words [50]. Representing words on a continuous scale allows for methods to measure word similarities, which are widely used in various information retrieval and NLP tasks [54]. Next to that, word embeddings are used for classification tasks, ranging from sentiment classification [106, 20, 85, 101] to cancer diagnosis [59], fake news detection [102] and emotion detection [38].

One method of creating embeddings is fastText. fastText is an efficient method to learn word representations useful for downstream tasks such as text classification. The main idea behind the approach is to represent the internal structure of the words through n-grams instead of learning a full word representation. Therefore, the model represents each word as a sum of vectors, where each vector represents an n-gram [5]. This approach has two benefits: first, morphological forms of a word are learned differently but will be very similar in the vector space. Second, embeddings for unknown words will generally still be informative, meaning that in all cases, we can use fastText to calculate semantic similarity.

Under the hood, it works as follows: a window is slid over the input text and the model either learns the center word from the remaining context, or all the context words from the center one. Essentially, a neural network with two layers of weights and three layers of neurons is trained, in which the two outer layers have a neuron for each word or sub-part of words (character n-grams) in the vocabulary, and the middle layer has as many neurons as dimensions in the embedding space [84, 11].

The open-sourced fastText package also includes a language identification model, which is a classification algorithm trained using fastText embeddings, to recognize more than 170 languages. This language detection functionality makes use of a simple linear model with rank constraint. Weights are set as a look-up table over the words, which are then averaged into a text representation. This representation is fed to a linear classifier. A softmax function computes the probability distribution over the predefined classes, which are the languages in this case [44]. The language with the highest probability is selected as the output language.

2.3 NLP for UML modelling

Several previous research approaches have been made to assist UML modellers by applying NLP techniques to requirements texts. For **class modelling**, multiple approaches have been proposed since the early 2000s to partly automate the process of creating class models. Using hand-coded rules based on Part-of-Speech (POS) tagging, the Linguistic assistant for Domain Analysis (LIDA) by Overmeyer et al. [72] identifies objects, their attributes and methods. The output can then be used for manually creating associations between the classes and to refine the identified objects. The Graphic Object-Oriented Analysis Laboratory (GOOAL) developed by Perez-Gonzalez and Kalita [76] takes a similar rule-based approach but regulates the requirements document before constructing a model. However, it can only handle simple problem domains and the texts still need to be structured before being able to be processed by the system. The Class Model Builder (CM-Builder) by Harmain and Gaizauskas [36] and the Requirements Analysis to Provide Instant Diagrams (RAPID) by More and Phalnikar [65] also suffer from this same limitation on input text, although they make use of more sophisticated NLP techniques.

More recent developments in class modelling still make use of rules to recognize parts of class diagrams. In their Automatic Builder of Class Diagram (ABCD), Azzouz et al. [8] use over a thousand hand-written patterns. Narawita and Vidanage [67] extend this rule-based approach with Weka software for recognizing relationship types and multiplicity with their UML Generator, being the first implementation to utilize machine learning in the effort of generating a class diagram from text. Contrasting previous approaches, Tang [95] combined parts of previous approaches to create a semi-automatic class modeller that supports extensive interactivity with the end-user for refining and finalizing diagrams, but similar problems are still apparent: because the tool uses a list of keywords to distinguish and link classes and attributes, the input text needs to be structured before being used.

The research field for automatic **activity and use case modelling** is considerably less extensive but has the same challenges as automatic class modelling. Iqbal and Bajwa [40] rely on the usage of natural language requirements in a formal structure in order to extract basic UML elements of an activity model. This approach is very similar to Nassar and Khamayseh [68], who specify clear guidelines for the requirements texts to follow before being able to generate activity models, and the method proposed by Maatuk and Abdelnabi [57], which requires input texts to follow a set of sixteen syntactic rules before facilitating UML element extraction for activity and use case models. Most of these implementations rely on POS tagging to identify usable sentences in requirements texts, such as work by Gulia and Choudhury [33], which again limits the input requirements text in terms of its structure.

For use case models, structuring raw text continues to be an apparent issue as well. Three implementations by Deeptimahanti et al. [23, 48, 24] normalize incoming texts using NLP tools before automatically forming class and use case models. The approach by Elallaoui et al. [26] relies on user stories, which inherently provide a strict structure already. Finally, the process of going from requirements texts to use case diagrams in the method developed by Hamza and Hammad [35] involves an intricate preprocessing step, including spell checking, and an approach is taken depending on structures in the text that indicate

whether the sentence is written in an active or passive voice.

To conclude, existing implementations for automatic generation of class, activity and use case models provide promising results, but share the same problem: the input text is limited to following a certain, predefined structure, and the implementations do not account for mixed model documents [80]. This underlines the need for a unified approach for processing raw, real-world requirements texts that can be used as a preprocessing step for (semi-)automatic UML modelling downstream.

3 Methods

3.1 Architecture overview

Our three-step pipeline architecture makes use of the following NLP techniques: i) entity-based summarization, ii) structural filtering, and iii) metadata tagging. These three steps are discussed in detail below. An overview of the complete pipeline is shown in Figure 6.

For the implementation of our preprocessing modules, we make use of BookNLP, an NLP pipeline in Python 3 specifically developed for operations on long texts [25]. BookNLP uses spaCy for POS tagging and dependency parsing. For the more complex tasks, it uses BERT models fine-tuned on different datasets, depending on the task at hand [7].

3.2 Datasets

3.2.1 PURE: a Dataset of Public Requirements Documents

The Public Requirements dataset is a dataset of 79 publicly available requirements documents covering a variety of domains and topics. This dataset only contains text documents without attached UML models. A subset of these requirements documents is ported to a common XML format with the goal of facilitating replication of NLP experiments [28], resulting in 18 requirements documents used in this project. Details on the amount of tokens, vocabulary and lexical diversity can be found in Table 2.

Because this subset is unlabelled, it cannot be used for supervised training. To be able to train the sequence labelling model introduced in Section 3.5, we analyzed the requirements texts in this dataset with the BookNLP modules and used coarse and fine-grained POS tags, lemmas, dependency relations, supersense categories, entity numbers, entity types and ACE 2005 entity categories as features. In the running pipeline, these features used for predictions are by-products of the previous two steps of the pipeline, and therefore require no additional computing power. We manually added labels to each word in the dataset, indicating the following:

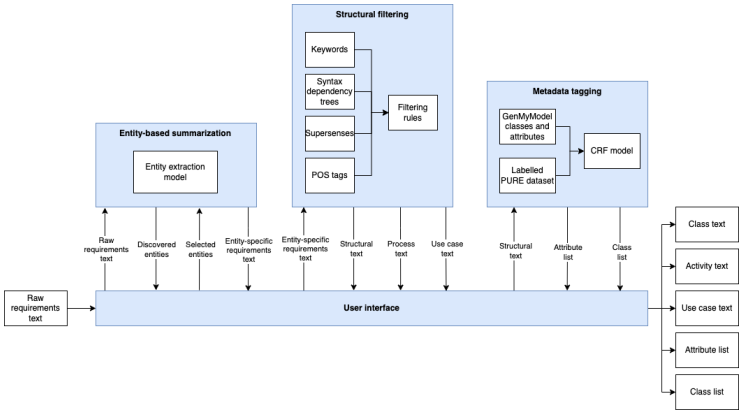


Figure 6: Architecture pipeline for going from input to output.

sentence_ ID	token_ ID_ within_ sentence	token_ ID_ within_ document	word	lemma	POS_ tag	fine_ POS_ tag	dependency_ relation	supersense_ category	entity	entity_ type	entity_ category	IOB_ tag
28	21	908	Distributors	distributor	NOUN	NNS	nsubj	noun.person	51.0	NOM	PER	B-class
28	22	909	keep	keep	VERB	VBP	ROOT	verb.stative				O
28	23	910	a	a	DET	DT	det					O
28	24	911	distribution	distribution	NOUN	NN	compound	noun.communication				B-class
28	25	912	list	list	NOUN	NN	nmod	noun.communication				I-class
28	26	913	documenting	document	VERB	VBG	amod	verb.communication				O
28	27	914	distributor	distributor	NOUN	NN	dobj	noun.person	52.0	NOM	PER	B-attr
28	28	915	,	,	PUNCT	,	punct					O
28	29	916	voucher	voucher	ADJ	JJ	amod	noun.possession				B-attr
28	30	917	number	number	NOUN	NN	conj	noun.quantity				I-attr
28	31	918	,	,	PUNCT	,	punct					O
28	32	919	date	date	NOUN	NN	conj	noun.time				B-attr
28	33	920	,	,	PUNCT	,	punct					O
28	34	921	place	place	NOUN	NN	appos	noun.location				B-attr
28	35	922	of	of	ADP	IN	prep					I-attr
28	36	923	sale	sale	NOUN	NN	pobj	noun.act				I-attr
28	37	924	,	,	PUNCT	,	punct					O
28	38	925	and	and	CCONJ	CC	cc					O
28	39	926	name	name	NOUN	NN	conj	noun.communication				B-attr
28	40	927	and	and	CCONJ	CC	cc					O
28	41	928	place	place	NOUN	NN	conj	noun.location				B-attr
28	42	929	of	of	ADP	IN	prep	noun.location				I-attr
28	43	930	living	living	NOUN	NN	pobj	noun.cognition				I-attr
28	44	931	of	of	ADP	IN	prep					O
28	45	932	the	the	DET	DT	det		53.0	NOM	PER	O
28	46	933	customer	customer	NOUN	NN	pobj	noun.person	53.0	NOM	PER	B-class
28	47	934	.	.	PUNCT	.	punct					O

Table 1: An example of a fully labelled sentence from the PURE dataset. Everything from the "lemma" column to the "dependency_relation" column are generated through spaCy, the "supersense_category" column and all entity columns are created through the BERT models of BookNLP. The final column, the "IOB_tag", is manually added.

- **B-class** indicates the beginning of a word group that represents a class in a class diagram.
- **I-class** indicates the continuation of a word group that represents a class in a class diagram.
- **B-attr** indicates the beginning of a word group that represents an attribute in a class diagram.
- **I-attr** indicates the continuation of a word group that represents an attribute in a class diagram.
- **O** indicates that the word does not belong to any word group and therefore does not represent either (a part of) a class group or attribute group.

An example of a fully labelled sentence from the dataset is displayed in Table 1. As part of this thesis, the PURE dataset with these labels in IOB format is publicly accessible in the GitHub repository linked in the Section 4 and can be used in machine learning tasks for the identification of classes and attributes in requirements texts.

3.2.2 Validation set

To evaluate our approach on unseen data, we gathered a selection of five requirements documents (without associated UML models) which are currently being used as training material by a large American software company for their software consultants and architects. All five texts are listed in Appendix A. An overview of the metrics of this validation set and how it relates to the training

Metric	PURE (training set)	Validation
Tokens	187,649	2,722
Vocabulary size (original tokens)	10,977	805
Vocabulary size (stems)	8,688	664
Number of sentences	7,928	147
Average sentence length (tokens)	24	18
Lexical diversity	0.046	0.244
Number of class (groups)	4478	215
Number of attribute (groups)	2182	140

Table 2: Characteristics of the PURE dataset and the validation set.

set can be found in Table 2. To validate the performance of the sequence labelling model introduced in Section 3.5, we labelled this dataset with the same IOB tags as the PURE dataset.

3.3 Entity-based summarization

The first step of the preprocessing pipeline is a summarization step to condense the incoming requirements into a more focused text. We propose a method for performing extractive summarization where a user can select discovered entities in the text and the pipeline only returns sentences that relate to these entities of interest.

The benefit of the interactive, entity-based summarization step is two-fold: firstly, by extracting sentences we omit the processing of the whole document which often includes irrelevant parts such as tables of content, management summaries, reasons for development, appendices etc. that are not useful for modelling. Secondly, enabling the user to focus on specific parts of the software allows for compartmentalized and incremental development, where big software systems can be split into smaller parts.

We first extract all entities from the raw requirements texts using the entity annotation module of BookNLP, which has been trained on an annotated dataset of 968K tokens, combining public domain materials in LitBank with a dataset of approximately 500 contemporary books.

From all discovered entities, two categories of entities are excluded: Geopolitical entities (GPE) and Organizations (ORG), together with all pronouns [100]. These categories are excluded because they often refer to named entities, which are typically not modelled in UML diagrams, leaving us with concepts that are more likely to refer to UML objects.

The entity extraction step often results in duplicate entities: for example, BookNLP classifies 'customer' and 'each customer' as separate entities. To combine these into a set of unique entities, we rely on the POS tags of the detected entities: we remove all words of the entity groups that are not a noun or an adjective according to the POS tagging of spaCy and remove duplicates.

After performing these transformations, the user is presented with all the extracted entities and makes a selection of entities that the modeller wants to use in modelling downstream.

After this selection, the next substep is the filtering of relevant sentences based on the relevant entities. Only sentences that contain (references to) the

selected entities of interest are returned to the user for further preprocessing in the pipeline. Thus, sentences that do not contain entities of interest directly, or have indirect links to the entities of interest via words such as "their" and "this" will be removed from the running text before continuing to the next step in the pipeline.

Alternatively, an abstractive approach could be feasible here. However, research on abstractive summarization methods with human interaction is limited. One of these methods, Customizable Abstractive Topic-based sequence-to-sequence Summarization (CATS), is an abstractive summarization method with which text documents can be summarized while selectively focusing on a range of desired topics. The model is based on the encoder-decoder architecture combined with the concept of pointer networks, which enables a combination of copying words from the source text and generating words from a fixed vocabulary. Summarizations are produced using a novel concept called *topical attention*: over encoder-decoder training steps, parameters adapt to learn the topics of each document [6]. However, using this model for creating summaries is not feasible in a production environment. The codebase is written in Python 2.7, and even after migrating the solution to Python 3, the model does not allow for extension to unknown topics due to its pretrained topic model. Next to that, selecting and deselecting topics of interest are only showcased for research purposes and not usable on scale, making our extractive summarization method based on entity extraction the best choice for this thesis.

3.4 Structural filtering

The next step in the pipeline is forming three 'buckets' to put sentences in: one for class modelling, one for activity modelling and one for use case modelling. The output of these steps is three texts that are concatenations of the sentences that belong in these buckets: a text for class modelling, a text for activity modelling and a text for use case modelling.

One sentence can appear in multiple buckets, as long as it conforms to the filtering rules that are defined for each of the buckets. These rules are based on four characteristics of the sentences, or a combination of multiple characteristics: keywords, syntax dependencies, supersenses and POS tags. For syntactic dependency parsing, we use word-level information of spaCy. Supersenses are a classification scheme for nouns and verbs that groups them based on the semantic meaning of the words [46]. For supersense tagging, we use the supersenses module from BookNLP, which was trained on SemCor. SemCor is a subset of the Brown Corpus (360K tokens) that is annotated with supersenses [77]. The combination of syntax dependencies and POS tags with supersenses allows us to create rules that both target syntactic and semantic structures within sentences.

The filtering rules for the class and use case texts are based on this combination. Combinations are made based on manual observations of the requirements texts in the PURE dataset, which is introduced in Section 3.2. We manually labelled all sentences of this dataset with whether they had indicators for class and use case modelling, gathered all sentences for class and use case modelling, generated their characteristics, and created rules that were as abstract as possible in order to keep the rules as high-level as possible, not focusing on edge cases. This process resulted in 16 manually defined structural rules for class texts and 4 rules for use case texts. Contrasting this approach, the filtering

rules for activity texts are based on keywords, extracted from previous research conducted by Friedrich et al. [30] and Ferreira [29] in the Business Process Model and Notation (BPMN) domain. All filtering rules are listed in Appendix C.

3.5 Metadata tagging

The last step of the process takes as input the class text gathered from the previous step and identifies and labels classes and attributes that are present in the text. For this purpose, we use a supervised sequence labelling model, conditional random fields (CRF). As stated previously, the relatively limited availability of publicly available training data for requirements engineering is a limitation of this research field. By using a CRF model, we can classify elements in a running text with only a small amount of training data.

For training the model, we use the labelled PURE dataset with the filtering rules of the second step of the pipeline applied to it. This ensures that we limit the training to patterns that we will be able to observe in the final implementation. Additionally, we experiment with extending this dataset in two ways. First, we add fastText embeddings for each token [10]. Second, we extended the dataset with class and attribute frequencies from the GenMyModel dataset, which is a repository of UML model files.

The frequencies work as follows. For every token that belongs to a class or attribute group, we retrieve how many times the word is used as a class and as an attribute in the repository. This results in three additional values used in the training process: the total amount of times the word is observed in classes and attributes in the repository and two values that indicate the absolute counts of how many times the word appears in class names and attribute names.

The GenMyModel dataset is a dataset of 352,216 XML files containing UML diagrams from GenMyModel, an online modelling tool through MAR, a search engine for model files [56]. GenMyModel is a UML modelling tool, without any specific focus on modelling areas, making it more suitable for this thesis. The files make use of a shared UML namespace, making it relatively easy to extract classes and attributes by extracting elements with the xsi:type attribute uml:Class, which mark the class objects, and then searching for owned attributes within this object, which are the attributes of the object. After stripping the names of the extracted classes and attributes, this resulted in 344,981 unique classes and 455,730 unique attributes.

Because we focus on the larger applications of UML within this project, we apply some additional syntactic preprocessing steps to transform programmatic class and attribute names:

1. Remove function calls, getters and setters, dot-separated widgets and filenames, comma-separated attributes, HTML and XML tags, dollar signs at the start of strings, digits attached to the end of words and all notions of "my" before another word, left-over parentheses, dashes, square brackets, hashtags, stars and slashes.

2. Transform all programmatic cases (snake case, camel case etc.) into space-separated text in order to reflect running texts as best as possible.

3. Replace abbreviations for implementation, reference, and the ampersand for their written-out version.

4. Remove duplicate spaces.

5. Remove all entries that contain non-Latin characters.

Because the collection of UML models are multi-lingual, the last cleaning step is to remove all non-English files from the dataset. For this, we use fastText's language identification model [44].

As an alternative way to get frequencies from model files, the Lindholmen Dataset of UML Models was considered. This dataset contains files extracted from GitHub, with a focus on the development of software projects [37]. However, even though this is a valuable resource for this field, this particular focus is not fitting for this thesis because it does not satisfy Sub-objective 1: requirements in any context and any system should be supported. Therefore, we only use the GenMyModel dataset, but a transformed version of this dataset with exactly the same preprocessing steps as described in this section can be found in the repository of this project on GitHub.

The end result is a list of 908,946 English-language classes, of which 180,429 are unique, and 1,232,355 attributes, of which 203,154 are unique.

For example, the word 'address' appears 21,845 times as an attribute, and 1,287 times as a class. This results in a total amount of occurences of 23,132. When extending the training data with this data, every time we use the token 'address', it will have the triplet (21845, 1287, 23132) added to it.

4 System implementation

The preprocessing pipeline as described in this thesis is implemented as part of the Prose to Prototype project to prove its effectiveness in real-world scenarios. The Prose to Prototype project aims to investigate how textual requirements documents can be processed with NLP techniques to map them to UML models. The main focus of the project is to allow for all types of textual expressions of requirements to be modelled automatically, or with a human-in-the-loop approach, resulting in sets of UML diagrams. Additionally, the project aims to synthesise these models with the goal of executing them as prototypes immediately [80]. Currently, the project is still in its prototype phase and therefore not yet publicly available.

The pipeline is the first step of the journey of going from requirements text to executable UML models within the project and aids the use case, class and activity modelling modules of the project, which make use of the output of the preprocessing in this thesis.

In addition to this integration, the code that powers the complete pipeline, including the produced datasets, models and experiments is on GitHub for reference.

4.1 Implementation details

The Prose to Prototype project consists of two parts: the Backend and the Editor. The Backend handles data storage and machine learning operations, while the Editor handles user interactions and provides the user with information about the project. The Backend and the Editor communicate through a RESTful API, allowing the technological stacks to be different for each part.

Both the Editor and Backend run in Docker containers, isolated environments that package the applications and manage their lifecycles, allowing for quick and easy deployment [2].

4.1.1 The Backend

The Backend consists of a Django application in Python 3, exposing API routes to interact with the data model of the preprocessing pipeline, but also powering the generation of activity, use case and class diagrams, and creating running prototypes.

In this project, the existing API was extended by introducing a new architecture to manage the creation of diagrams. A class model showing this new architecture is shown in Figure 7.

To allow for compartmentalization of the requirements engineering process, a user can create a project, which holds one or more requirements documents to store the general requirements. In practice, requirements documents are often documents with more than 20 pages, describing software in general. Therefore, we introduce the notion of a system: a certain subselection of requirements from the main requirements document that allows for compartmentalization of the modelling process. A project can have many systems, each describing a certain part of the document. A user can define this subset by selecting entities in the main requirements document that are of interest for the system, together with a selection of UML models that the user wants to create for the system. As a

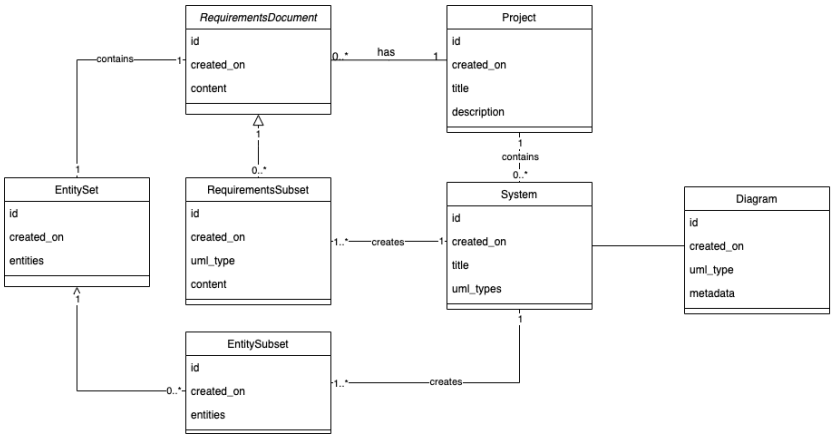


Figure 7: A class diagram showing a high-level structure of the Prose to Prototype system.

result, up to three requirements subsets are generated per system: one for each of the UML types that are currently supported, holding the content describing only the entities that the user has defined.

These requirements subsets are then used in the UML generation and extraction processes for each of the different UML types, which are already present in the Prose to Prototype project. In this research, these extraction and generation processes have remained largely untouched, except for the implementation of the metadata extraction based on the CRF models introduced in this thesis in the class model extractor.

4.1.2 The Editor

The Editor is created with React.js, a JavaScript library for building user interfaces [3], in combination with TypeScript, an extension of JavaScript for creating a more stable environment due to user-defined type definitions [4]. For the purpose of reusability, the project makes use of IBM's Carbon Components library, which contains common components that keep the amount of code that has to be written to create new functionalities to a minimum [1]. This technological stack was chosen so that there is a stable and modern development workflow, allowing the project to grow and mature further in the future, even when new research projects add to it.

During this research, several web pages have been added to the Editor to facilitate interaction with the preprocessing pipeline. Screenshots of these web pages can be found in Appendix B. The general user flow is as follows: a user finds the landing page (Figure 11) and clicks the button to get the process started. Then, the user is taken into a three-part process.

First, they can decide to choose an existing Project if the user has created one before (Figure 12), or the user can click a button to create a new Project. For creating a new Project, the user has to fill in a project name, description, and requirements have to be provided. These requirements can either be written in a textbox, or the user can upload a plaintext file, or an audio file, which is converted into plaintext. Once the user is satisfied with the content of the

Project, the user can continue to step 2: creating a new System (Figure 14).

As stated earlier, a System contains a subselection of the requirements, focused on specific entities of interest and UML types. From the requirements text that has been filled in by the user in the previous text, the preprocessing pipeline extracts entities. Here, the user can select some of the detected entities that they are interested in to proceed with automatic modelling. After choosing a name for the System and picking entities of interest and UML types, the user proceeds to the final step: reviewing the extraction (Figure 15).

In this last step, the subsets of the requirements are shown to the user for each of the selected UML types, together with the generated metadata from the pipelines for the generation of each of the diagrams. Dedicating a step to reviewing the extraction allows the user to go back in the process via the navigation elements at the bottom of each of the preprocessing steps and change certain inputs to tune the end result.

After reviewing the extraction, the visual diagrams are generated, and the user is redirected to a built-in UML editor, where they can change, extend and improve the generated models. Next to that, the system has the capability of generating a runnable prototype of a system. This is done by taking the created class model and use it as a database model for a sample web application. In this generated web application, the user can read and write data to the system so that they can test out a system and validate it immediately.

5 Results

To evaluate the results of our pipeline, we run one text out of our validation set through the steps outlined above. Where possible, we provide quantitative results to accompany our qualitative example.

5.1 Entity-based summarization

The contents of the validation text are shown in Figure 8. The named entity extraction model discovers the following entities in this text: *small truck trailer, rental office, vehicle various type truck, customer, customer ready possession, vehicle, rented vehicle, central office, office, single home office, different type vehicle, truck, individual customer, individual, company, home, driver, single individual company, more vehicle, truck trailer* and *open trailer*.

The results of selecting the entities *customer*, *vehicle* and *truck* for the entity-based summarization are also shown in Figure 8. The selected text is displayed in green, the discarded text is not highlighted. Because of our efforts of grouping (semi-)duplicate entities together, we achieve good qualitative results in this step for this sample text, allowing the user to effectively select parts of the system for further modelling while keeping in mind the user experience by not presenting the complete list of detected entities in the text.

5.2 Structural filtering

In Figure 9, the filtered output texts of the first step of the pipeline are shown. The usage of a broad range of rules for the class text results in the longest text of the three (a). Due to the limitations of the use of keywords to find sentences related to processes, the activity text only consists of two sentences. Finally, our rules result in a use case text that contains sentences that always involve an actor in an active way, giving us a satisfactory result overall.

To provide a benchmark for future research, we have labelled all sentences in our validation set with the type of modelling the sentence is useful for. This validation set comprises 5 texts, which can be found in Appendix A. As displayed in Table 3, each text has a main focus, but we also labelled each sentence for

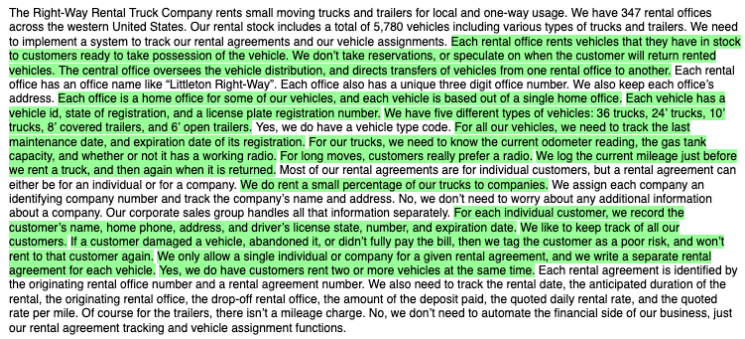
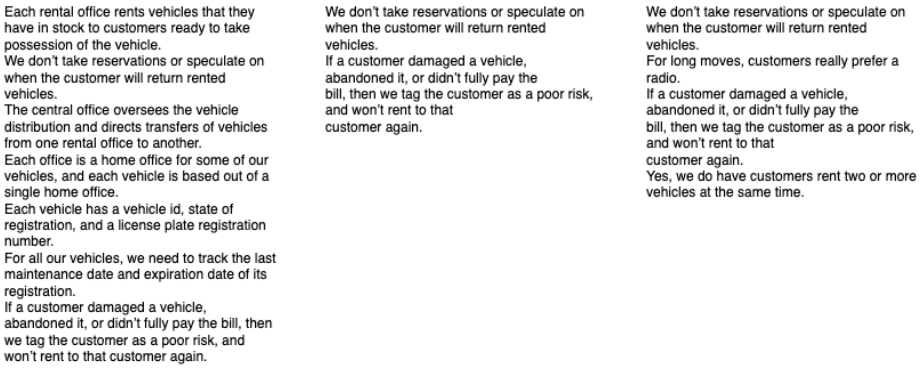

The Right-Way Rental Truck Company rents small moving trucks and trailers for local and one-way usage. We have 347 rental offices across the western United States. Our rental stock includes a total of 5,780 vehicles including various types of trucks and trailers. We need to implement a system to track our rental agreements and our vehicle assignments. Each rental office rents vehicles that they have in stock to customers ready to take possession of the vehicle. We don't take reservations, or speculate on when the customer will return rented vehicles. The central office oversees the vehicle distribution, and directs transfers of vehicles from one rental office to another. Each rental office has an office name like "Littleton Right-Way". Each office also has a unique three digit office number. We also keep each office's address. Each office is a home office for some of our vehicles, and each vehicle is based out of a single home office. Each vehicle has a vehicle id, state of registration, and a license plate registration number. We have five different types of vehicles: 36 trucks, 24' trucks, 10' trucks, 8' covered trailers, and 6' open trailers. Yes, we do have a vehicle type code. For all our vehicles, we need to track the last maintenance date, and expiration date of its registration. For our trucks, we need to know the current odometer reading, the gas tank capacity, and whether or not it has a working radio. For long moves, customers really prefer a radio. We log the current mileage just before we rent a truck, and then again when it is returned. Most of our rental agreements are for individual customers, but a rental agreement can either be for an individual or for a company. We do rent a small percentage of our trucks to companies. We assign each company an identifying company number and track the company's name and address. No, we don't need to worry about any additional information about a company. Our corporate sales group handles all that information separately. For each individual customer, we record the customer's name, home phone, address, and driver's license state, number, and expiration date. We like to keep track of all our customers. If a customer damaged a vehicle, abandoned it, or didn't fully pay the bill, then we tag the customer as a poor risk, and won't rent to that customer again. We only allow a single individual or company for a given rental agreement, and we write a separate rental agreement for each vehicle. Yes, we do have customers rent two or more vehicles at the same time. Each rental agreement is identified by the originating rental office number and a rental agreement number. We also need to track the rental date, the anticipated duration of the rental, the originating rental office, the drop-off rental office, the amount of the deposit paid, the quoted daily rental rate, and the quoted rate per mile. Of course for the trailers, there isn't a mileage charge. No, we don't need to automate the financial side of our business, just our rental agreement tracking and vehicle assignment functions.

Figure 8: Applying the entity-based extractive summarization step on the validation text.



(a) Class text. (b) Activity text. (c) Use case text.

Figure 9: Applying the structural filtering step on the summarized text to create 3 'buckets'.

which type of modelling it is useful. The validation set contains 145 sentences, and each sentence can be labelled as useful for more than one type of modelling. The length of the texts is between 300 and 549 words, and they span a range of industries in order to make the findings of this research applicable to a large number of scenarios.

Even though this is only a small collection, we show the classification result in Table 4. Because we want to be able to effectively use the filtering rules to take out positive cases, we look at the recall on the "useful" classes. Unfortunately, the recall here is relatively low. For the rules for class models and use case models, this can be explained by the difference in sparsity of useful information between the training set and the validation set: in the validation set, very focused texts with lots of useful information is present, while the training data on which the rules are based has a relatively low amount of useful information, resulting in highly specific rules to catch sparse patterns. For the rules for activity models, we can observe that only using the keywords defined by Friedrich et al. [30] and Ferreira [29] do not result in a satisfying performance.

5.3 Metadata tagging

Table 5 displays all gathered classification results of the trained CRF model on the validation set. The scores are displayed in four stages in order to approxi-

Title	**Word count**	**Modelling focus**
Dental clinic	300	Class, activity
Restaurant	549	Class, activity, use case
Observations on geological samples	413	Class, activity
Law firm	494	Class
Rental truck company	520	Class, activity

Table 3: An overview of characteristics of the used requirements texts in the validation set.

	Precision	Recall	F1-score	Support	Accuracy
Useful for class modelling	0.79	0.56	0.66	110	0.56
Not useful for class modelling	0.28	0.54	0.37	35	
Useful for activity modelling	0.60	0.41	0.49	22	0.87
Not useful for activity modelling	0.90	0.95	0.92	123	
Useful for use case modelling	0.38	0.44	0.41	27	0.76
Not useful for use case modelling	0.87	0.83	0.85	118	

Table 4: Classification scores of the structural filtering step.

mate the influence of each group of features on the end result. For each training stage, we tuned the hyperparameters c1 and c2 using randomized search. The first stage entailed a minimal training setup: we trained the CRF model using our base features, which only included the base information for each token (POS tag, dependency relation, supersense, entity type, entity category, surrounding words and POS tags, etc.). In the second stage, we added information from the GenMyModel dataset: training was conducted on the base information in combination with the frequency of occurrences of the token in the GenMyModel dataset, and how many of those occurrences were labelled as classes or attributes. The third stage combined the base information with fastText embeddings for each token. Finally, in the last stage, all of the additional features were combined.

By default, the training materials were sparse in terms of classes and attributes as compared to the validation materials. As a result, the related F1 scores are generally not very high and differ only slightly between the four different test scenarios. As is often the case in machine learning, we can observe that having more data results in the best performance: all features combined gives the best scores across the board. Only frequencies alone score slightly higher, and only on the B-class tag. Frequencies and embeddings separately hardly pass the default experimental set-up in performance, indicating that a historical lookup still requires context-level information for disambiguating aspects of class models in requirements texts.

To finalize the qualitative evaluation of the validation text, we display the results of running the class text from the previous step through the model in

		B-class	I-class	B-attr	I-attr
Default	**Precision**	0.622	0.382	0.681	0.722
	Recall	0.473	0.310	0.320	0.361
	F1-score	0.537	0.342	0.435	0.481
With frequencies	**Precision**	0.627	0.382	0.721	0.792
	Recall	0.496	0.310	0.310	0.352
	F1-score	0.554	0.342	0.434	0.487
With embeddings	**Precision**	0.608	0.353	0.681	0.750
	Recall	0.457	0.286	0.320	0.361
	F1-score	0.522	0.316	0.435	0.487
Everything combined	**Precision**	0.633	0.394	0.744	0.750
	Recall	0.481	0.310	0.320	0.361
	F1-score	0.546	0.347	0.448	0.487

Table 5: Classification scores for the CRF model in the final step of the pipeline.

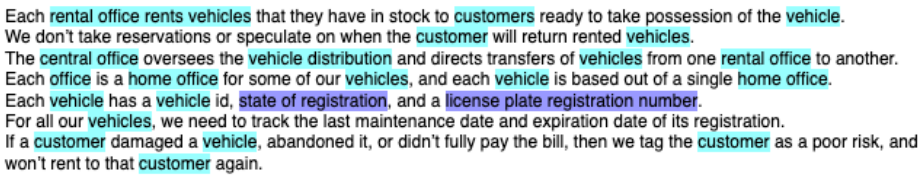


Figure 10: Applying the metadata tagging step on the validation text.

Figure 10. This is the result of using the full feature set, so with frequencies and embeddings. The blue highlighted groups are classified as classes and the purple highlighted groups are classified as attributes.

The most important classes, such as the customer class and the vehicle class, are consistently classified the right way throughout the text. However, the model does not accommodate for less common classes. For example, in the second sentence, we would expect "reservations" to also be a class. Next to that, it seems that attributes listed in a sentence structure that deviates from the standard format "*each <class> has a <enumeration of attributes>*" are not picked up by the model.

To conclude, from a quantitative point of view, the model seems to perform best when there is information about historical occurrences and word embedding, but the F1 scores are not satisfactory. From a qualitative point of view, it seems like the model performs its basic functions well and offers a good basis for adjusting the outcomes of the classification downstream. Especially considering the user has plenty of options to adjust the model outcome downstream, we can safely say the system is usable even with the relatively low F1 scores.

6 Conclusion

6.1 Main findings

In previous sections, this paper laid out a pipeline for preprocessing real-world requirements texts into structured texts for the purpose of generating class, activity, and use case models, including metadata for class modelling specifically.

We defined four sub-objectives for the development of the pipeline. Sub-objective 1 focused on the ability to preprocess systems in any context. The entity-based extractive summarization method developed in this thesis allows a user to select entities of interest. These entities are extracted in a context-agnostic way, and can therefore be applied in any circumstance. In the end, with this pipeline, the user is responsible for selecting entities that are of interest, and we can assume that the user selects entities that are useful for the specific context of the UML model. As an alternative to the CATS model that we experimented with in the beginning, but which was unable to function in a production environment, we used the entity extraction model of the BookNLP package to replicate the same effect in an extractive approach. We ended up with a method that is predictable and easily tunable by the end user, which shows good qualitative results.

Sub-objective 2 focused on the ability to find and recognize patterns for bucketing texts for specific UML models. Our approach combines findings in research with linguistic techniques such as supersenses and syntax dependencies to define patterns that are useful for determining for which type of UML model a sentence is useful. The main challenge of this step was bridging the gap between the sparse PURE dataset, which formed the basis for our rules, and the dense validation set. This gap resulted in subpar recall scores. However, because of the interpretable nature of the rules, we ended up with a method that allows for easy extension of the rules in the future.

Sub-objective 3 focused on the application of novel machine learning and NLP techniques to generate metadata. To develop our class model metadata extractor, we created a novel dataset of classes and attributes, and the classification results provide both a model for future development of metadata models, but also put out a benchmark for the classification task of distinguishing classes and attributes in running texts.

Lastly, sub-objective 4 focused on the option of human intervention in the output of the preprocessing pipeline. Through the developed interface and the summarization where user interaction lies at its core, the user can freely control both the input and output of the preprocessing pipeline and tweak results before even generating a visual diagram.

To conclude, our experimental results set a benchmark for future work, provide new training material, and provide a new direction of methods for the analysis of requirements texts, including the novel use of entity extraction to gather entities of interest for UML modelling. Next to that, this paper forms a basis for a more uniform approach to preprocessing requirements texts, with the goal of advancing research in this area.

6.2 Future work

Looking at the limitations of our research, future work is needed on automatically locating parts of requirements texts that are useful for systems design. We especially see opportunities in a more context-aware method of distinguishing classes and attributes from each other, but more research on the difference between classes and attributes on the word level is also welcomed. Furthermore, the preprocessing pipeline developed in this thesis is focused on "traditional" requirements engineering. However, the business environment is rapidly changing, challenging these approaches, and favouring agile methods in certain industries, such as software development [15]. With agile methods, user stories are oftentimes used to communicate system requirements. These user stories are written on a value level, as opposed to a type level which is currently supported in the pipeline. Support for user stories in this pipeline should be explored, allowing for example to go from "as a user, I want my aeroplane to be pink" to "the aeroplane has a colour". To conclude, the lack of datasets on this topic remains a limitation for future research. Even though considerable work went into creating labelled datasets to support this thesis, independent validation of our data, or extension of this work into, for example, attributes and methods versus classes and subclasses would be a valuable addition to this research.

References

[1] Carbon Design System. https://carbondesignsystem.com/. Accessed: 2022-05-29.

[2] Docker overview. https://docs.docker.com/get-started/overview/. Accessed: 2022-05-29.

[3] React.js: A JavaScript library for building user interfaces. https://reactjs.org/. Accessed: 2022-05-29.

[4] TypeScript: JavaScript with syntax for types. https://www.typescriptlang.org/. Accessed: 2022-05-29.

[5] Ben Athiwaratkun, Andrew Gordon Wilson, and Anima Anandkumar. Probabilistic fasttext for multi-sense word embeddings. *arXiv preprint arXiv:1806.02901*, 2018.

[6] Seyed Ali Bahrainian, George Zerveas, Fabio Crestani, and Carsten Eickhoff. CATS: Customizable Abstractive Topic-based Summarization. *ACM Transactions on Information Systems*, 2021.

[7] David Bamman. Booknlp, a natural language processing pipeline for books. https://github.com/booknlp/booknlp, 2021.

[8] Wahiba Ben Abdessalem Karaa, Zeineb Ben Azzouz, Aarti Singh, Nilanjan Dey, Amira S. Ashour, and Henda Ben Ghazala. Automatic builder of class diagram (abcd): an application of uml generation from functional requirements. *Software: Practice and Experience*, 46(11):1443–1458, 2016.

[9] Paheli Bhattacharya, Kaustubh Hiware, Subham Rajgaria, Nilay Pochhi, Kripabandhu Ghosh, and Saptarshi Ghosh. A comparative study of summarization algorithms applied to legal case judgments. In *European Conference on Information Retrieval*, pages 413–428. Springer, 2019.

[10] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[11] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[12] Grady Booch. *The unified modeling language user guide*. Pearson Education India, 2005.

[13] Egon Börger, Alessandra Cavarra, and Elvinia Riccobene. An asm semantics for uml activity diagrams. In *International Conference on Algebraic Methodology and Software Technology*, pages 293–308. Springer, 2000.

[14] John W Brackett. Software requirements. Technical report, Carnegie-Mellon University Software Engineering Institute, 1990.

[15] Lan Cao and Balasubramaniam Ramesh. Agile requirements engineering practices: An empirical study. *IEEE software*, 25(1):60–67, 2008.

[16] Silvia Casola and Alberto Lavelli. Summarization, simplification, and generation: The case of patents. *arXiv preprint arXiv:2104.14860*, 2021.

[17] Jayeeta Chanda, Ananya Kanjilal, Sabnam Sengupta, and Swapan Bhattacharya. Traceability of requirements and consistency verification of uml use case, activity and class diagram: A formal approach. In *2009 Proceeding of International Conference on Methods and Models in Computer Science (ICM2CS)*, pages 1–4. IEEE, 2009.

[18] Fan Chen, Christophe De Vleeschouwer, and Andrea Cavallaro. Resource allocation for personalized video summarization. *IEEE Transactions on Multimedia*, 16(2):455–469, 2013.

[19] Anton Chernyavskiy, Dmitry Ilvovsky, and Preslav Nakov. Transformers:" the end of history" for nlp? *arXiv preprint arXiv:2105.00813*, 2021.

[20] Abdelghani Dahou, Shengwu Xiong, Junwei Zhou, Mohamed Houcine Haddoud, and Pengfei Duan. Word embeddings and convolutional neural network for arabic sentiment classification. In *Proceedings of coling 2016, the 26th international conference on computational linguistics: Technical papers*, pages 2418–2427, 2016.

[21] B Davis et al. The role of cnl and amr in scalable abstractive summarization for multilingual media monitoring. In *Controlled Natural Language: 5th International Workshop, CNL 2016, Aberdeen, UK, July 25-27, 2016, Proceedings*, volume 9767, page 127. Springer, 2016.

[22] Bjorn Decker, Eric Ras, Jorg Rech, Pascal Jaubert, and Marco Rieth. Wiki-based stakeholder participation in requirements engineering. *IEEE software*, 24(2):28–35, 2007.

[23] Deva Kumar Deeptimahanti and Muhammad Ali Babar. An automated tool for generating uml models from natural language requirements. In *2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 680–682. IEEE, 2009.

[24] Deva Kumar Deeptimahanti and Ratna Sanyal. Semi-automatic generation of uml models from natural language requirements. In *Proceedings of the 4th India Software Engineering Conference*, pages 165–174, 2011.

[25] Ryan Dubnicek, Ted Underwood, and J Stephen Downie. Creating a disability corpus for literary analysis: Pilot classification experiments. *iConference 2018 Proceedings*, 2018.

[26] Meryem Elallaoui, Khalid Nafil, and Raja Touahni. Automatic transformation of user stories into uml use case diagrams using nlp techniques. *Procedia computer science*, 130:42–49, 2018.

[27] Andre Esteva, Anuprit Kale, Romain Paulus, Kazuma Hashimoto, Wenpeng Yin, Dragomir Radev, and Richard Socher. Covid-19 information retrieval with deep-learning based semantic search, question answering, and abstractive summarization. *NPJ digital medicine*, 4(1):1–9, 2021.

[28] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. Pure: A dataset of public requirements documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 502–505, 2017.

[29] Renato César Borges Ferreira, Lucinéia Heloisa Thom, and Marcelo Fantinato. A semi-automatic approach to identify business process elements in natural language texts. In *ICEIS (3)*, pages 250–261, 2017.

[30] Fabian Friedrich, Jan Mendling, and Frank Puhlmann. Process model generation from natural language text. In *International Conference on Advanced Information Systems Engineering*, pages 482–496. Springer, 2011.

[31] David Garlan. Software engineering in an uncertain world. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 125–128, 2010.

[32] Christian Gulden, Melanie Kirchner, Christina Schüttler, Marc Hinderer, Marvin Kampf, Hans-Ulrich Prokosch, and Dennis Toddenroth. Extractive summarization of clinical trial descriptions. *International journal of medical informatics*, 129:114–121, 2019.

[33] Sarita Gulia and Tanupriya Choudhury. An efficient automated design to generate uml diagram from natural language specifications. In *2016 6th international conference-cloud system and big data engineering (Confluence)*, pages 641–648. IEEE, 2016.

[34] Ashok Kumar Gupta Gupta and Aziz Deraman. A framework for software requirement ambiguity avoidance. *International Journal of Electrical and Computer Engineering*, 9(6):5436, 2019.

[35] Zahra Abdulkarim Hamza and Mustafa Hammad. Generating uml use case models from software requirements using natural language processing. In *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*, pages 1–6. IEEE, 2019.

[36] Harmain Mohamed Harmain and R Gaizauskas. Cm-builder: an automated nl-based case tool. In *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering*, pages 45–53. IEEE, 2000.

[37] Regina Hebig, Truong Ho Quang, Michel RV Chaudron, Gregorio Robles, and Miguel Angel Fernandez. The quest for open source projects that use uml: mining github. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 173–183, 2016.

[38] Jonathan Herzig, Michal Shmueli-Scheuer, and David Konopnicki. Emotion detection from text via ensemble classification using word embeddings. In *Proceedings of the ACM SIGIR international conference on theory of information retrieval*, pages 269–272, 2017.

[39] Matthew Honnibal and Mark Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[40] Usama Iqbal and Imran Sarwar Bajwa. Generating uml activity diagram from sbvr rules. In *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, pages 216–219. IEEE, 2016.

[41] Rubén Izquierdo, Armando Suárez, and German Rigau. Using semantic classes as document keywords. In *International Conference on Application of Natural Language to Information Systems*, pages 225–229. Springer, 2011.

[42] Ivar Jacobson. Basic use-case modeling (continued). *The Road to the Unified Software Development Process*, 18:183, 2000.

[43] Ivar Jacobson, Ian Spence, and Brian Kerr. Use-case 2.0. *Communications of the ACM*, 59(5):61–69, 2016.

[44] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[45] Mohamad Kassab, Colin Neill, and Phillip Laplante. State of practice in requirements engineering: contemporary data. *Innovations in Systems and Software Engineering*, 10(4):235–241, 2014.

[46] Karin Kipper, Anna Korhonen, Neville Ryant, and Martha Palmer. Extending verbnet with novel verb classes. In *LREC*, pages 1027–1032, 2006.

[47] Frans Knibbe and Alejandro Llaves. Spatial Data on the Web Use Cases & Requirements: W3C Working Group Note 25 October 2016. `https://www.w3.org/TR/sdw-ucr/#ObservationsOnGeologicalSamples`. Accessed: 2021-11-08.

[48] Deeptimahanti Deva Kumar and Ratna Sanyal. Static uml model generator from analysis of requirements (sugar). In *2008 Advanced Software Engineering and Its Applications*, pages 77–84. IEEE, 2008.

[49] Mathias Landhäußer, Sven J Körner, and Walter F Tichy. From requirements to uml models and back: how automatic processing of text can support requirements engineering. *Software Quality Journal*, 22(1):121–149, 2014.

[50] Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, 2014.

[51] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019.

[52] Elizabeth D Liddy. Natural language processing. In *Encyclopedia of Library and Information Science, 2nd Ed.* Marcel Decker, Inc., 2001.

[53] Yanyan Lin and Jieping Lu. Research on domain terminology recognition based on dependency tree-conditional random field. In *Journal of Physics: Conference Series*, volume 1213, page 052076. IOP Publishing, 2019.

[54] Yang Liu, Zhiyuan Liu, Tat-Seng Chua, and Maosong Sun. Topical word embeddings. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[55] Philipp Lombriser, Fabiano Dalpiaz, Garm Lucassen, and Sjaak Brinkkemper. Gamified requirements engineering: model and experimentation. In *International Working conference on requirements engineering: foundation for software quality*, pages 171–187. Springer, 2016.

[56] José Antonio Hernández López and Jesús Sánchez Cuadrado. Mar: a structure-based search engine for models. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 57–67, 2020.

[57] Abdelsalam M. Maatuk and Esra A. Abdelnabi. Generating uml use case and activity diagrams using nlp techniques and heuristics rules. In *International Conference on Data Science, E-learning and Information Systems 2021*, pages 271–277, 2021.

[58] Sean MacAvaney, Sajad Sotudeh, Arman Cohan, Nazli Goharian, Ish Talati, and Ross W Filice. Ontology-aware clinical abstractive summarization. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1013–1016, 2019.

[59] Andrés Alejandro Ramos Magna, Héctor Allende-Cid, Carla Taramasco, Carlos Becerra, and Rosa L Figueroa. Application of machine learning and word embeddings in the classification of cancer diagnosis using patient anamnesis. *Ieee Access*, 8:106198–106213, 2020.

[60] Dian Sa'adillah Maylawati, Muhammad Ali Ramdhani, and Abdusy Syakur Amin. Tracing the linkage of several unified modelling language diagrams in software modelling based on best practices. *International Journal of Engineering & Technology (UEA)*, 7(2.19):776–780, 2018.

[61] Matthew John McGill. Uml class diagram syntax: An empirical study of comprehension. 2001.

[62] Kaiz Merchant and Yash Pande. NLP based latent semantic analysis for legal text summarization. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1803–1807. IEEE, 2018.

[63] Farid Meziane, Nikos Athanasakis, and Sophia Ananiadou. Generating natural language specifications from uml class diagrams. *Requirements Engineering*, 13(1):1–18, 2008.

[64] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[65] Priyanka More and Rashmi Phalnikar. Generating uml diagrams from natural language specifications. *International Journal of Applied Information Systems*, 1(8):19–23, 2012.

[66] Robert J Muller. *Database design for smarties: using UML for data modeling*. Morgan Kaufmann, 1999.

[67] Chamitha Ramal Narawita and Kaneeka Vidanage. UMl generator –use case and class diagram generation from text requirements. *International Journal on Advances in ICT for Emerging Regions (ICTer)*, 10:1, 1 2018.

[68] Ibrahim N Nassar and Faisal T Khamayseh. Constructing activity diagrams from arabic user requirements using natural language processing tool. In *2015 6th International Conference on Information and Communication Systems (ICICS)*, pages 50–54. IEEE, 2015.

[69] Bashar Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–119, 2001.

[70] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 35–46, 2000.

[71] Brendan O'Connor and Michael Heilman. Arkref: A rule-based coreference resolution system. *arXiv preprint arXiv:1310.1975*, 2013.

[72] Scott P Overmyer, L Benoit, and R Owen. Conceptual modeling through linguistic analysis using lida. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pages 401–410. IEEE, 2001.

[73] Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements engineering and agile software development. In *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.*, pages 308–313. IEEE, 2003.

[74] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[75] Ru Peng, Zhitao Chen, Tianyong Hao, and Yi Fang. Neural machine translation with attention based on a new syntactic branch distance. In *China Conference on Machine Translation*, pages 47–57. Springer, 2019.

[76] Hector G Perez-Gonzalez and Jugal K Kalita. Gooal: a graphic object oriented analysis laboratory. In *Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 38–39, 2002.

[77] Tommaso Petrolito and Francis Bond. A survey of wordnet annotated corpora. In *Proceedings of the Seventh Global WordNet Conference*, pages 236–245, 2014.

[78] Rob Pooley and Peter King. The unified modelling language and performance engineering. *IEE Proceedings-Software*, 146(1):2–10, 1999.

[79] Guus Ramackers. Personal communication.

[80] Guus Ramackers, Pepijn Griffioen, Martijn Schouten, and Michel Chaudron. From Prose to Prototype: Synthesising Executable UML Models from Natural Language. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 380–389, 2021.

[81] Gianna Reggio, Maurizio Leotta, Filippo Ricca, and Diego Clerissi. What are the used uml diagrams? a preliminary survey. *EESSMOD@ MoDELS*, 1078(10), 2013.

[82] Rob Roggema. Research by design: Proposition for a methodological approach. *Urban science*, 1(1):2, 2016.

[83] Safdar Aqeel Safdar, Muhammad Zohaib Iqbal, and Muhammad Uzair Khan. Empirical evaluation of uml modeling tools–a controlled experiment. In *European Conference on Modelling Foundations and Applications*, pages 33–44. Springer, 2015.

[84] Igor Santos, Nadia Nedjah, and Luiza de Macedo Mourelle. Sentiment analysis using convolutional neural network with fasttext embeddings. In *2017 IEEE Latin American conference on computational intelligence (LA-CCI)*, pages 1–5. IEEE, 2017.

[85] Prathusha K Sarma, Yingyu Liang, and William A Sethares. Domain adapted word embeddings for improved sentiment classification. *arXiv preprint arXiv:1805.04576*, 2018.

[86] Max Savery, Asma Ben Abacha, Soumya Gayen, and Dina Demner-Fushman. Question-driven summarization of answers to consumer health questions. *Scientific Data*, 7(1):1–9, 2020.

[87] G Michael Schneider, Johnny Martin, and Wei-Tek Tsai. An experimental study of fault detection in user requirements documents. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 1(2):188–204, 1992.

[88] Nathan Schneider and Noah A Smith. A corpus and model integrating multiword expressions and supersenses. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1537–1547, 2015.

[89] Martijn B.J. Schouten, Guus J. Ramackers, and Suzan Verberne. Preprocessing Requirements Documents for Automatic UML Modelling. In *Proceedings of the 27th International Conference on Natural Language & Information Systems*. Springer Nature, 2022.

[90] Richa Sharma, Pratyoush K Srivastava, and Kanad K Biswas. From natural language requirements to uml class diagrams. In *2015 IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, pages 1–8. IEEE, 2015.

[91] Keng Siau and Lihyunn Lee. Are use case and class diagrams complementary in requirements analysis? an experimental study on use case and class diagrams in uml. *Requirements engineering*, 9(4):229–237, 2004.

[92] Shafaq Siddiqui, M Abdul Rehman, Sher M Daudpota, and Ahmad Waqas. Opinion mining: An approach to feature engineering. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 10(3), 2019.

[93] Perdita Stevens. On use cases and their relationships in the unified modelling language. In *International Conference on Fundamental Approaches to Software Engineering*, pages 140–155. Springer, 2001.

[94] Milan Straka, Nikita Mediankin, Tom Kocmi, Zdeněk Žabokrtskỳ, Vojtěch Hudeček, and Jan Hajic. Sumeczech: Large czech news-based summarization dataset. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[95] Tiantian Tang. From natural language to uml class models: An automated solution using nlp to assist requirements analysis. Master's thesis, Leiden University, 2020.

[96] Takuya Uemura, Shinji Kusumoto, and Katsuro Inoue. Function-point analysis using design specifications based on the unified modelling language. *Journal of software maintenance and evolution: Research and practice*, 13(4):223–243, 2001.

[97] Eva Vanmassenhove and Andy Way. Supernmt: neural machine translation with semantic supersenses and syntactic supertags. In *Proceedings of ACL 2018, Student Research Workshop*, pages 67–73. Association for Computational Linguistics (ACL), 2018.

[98] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[99] Sriram Venkatapathy, Rajeev Sangal, Aravind Joshi, and Karthik Gali. A discriminative approach for dependency based statistical machine translation. In *Proceedings of the 4th Workshop on Syntax and Structure in Statistical Translation*, pages 66–74, 2010.

[100] Christopher Walker, Stephanie Strassel, Julie Medero, and Kazuaki Maeda. Ace 2005 multilingual training corpus. *Linguistic Data Consortium, Philadelphia*, 57:45, 2006.

[101] Jenq-Haur Wang, Ting-Wei Liu, Xiong Luo, and Long Wang. An lstm approach to short text sentiment classification with word embeddings. In *Proceedings of the 30th conference on computational linguistics and speech processing (ROCLING 2018)*, pages 214–223, 2018.

[102] William Yang Wang. "liar, liar pants on fire": A new benchmark dataset for fake news detection. *arXiv preprint arXiv:1705.00648*, 2017.

[103] Zhengjue Wang, Zhibin Duan, Hao Zhang, Chaojie Wang, Long Tian, Bo Chen, and Mingyuan Zhou. Friendly topic assistant for transformer based abstractive summarization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 485–497, 2020.

[104] Petia Wohed, Wil MP van der Aalst, Marlon Dumas, Arthur HM ter Hofstede, and Nick Russell. Pattern-based analysis of the control-flow perspective of uml activity diagrams. In *International Conference on Conceptual Modeling*, pages 63–78. Springer, 2005.

[105] Shweta Yadav, Deepak Gupta, Asma Ben Abacha, and Dina Demner-Fushman. Question-aware transformer models for consumer health question summarization. *Journal of Biomedical Informatics*, 128:104040, 2022.

[106] Xiao Yang, Craig Macdonald, and Iadh Ounis. Using word embeddings in twitter election classification. *Information Retrieval Journal*, 21(2):183–207, 2018.

[107] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences, 2021.

[108] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2019.

[109] Didar Zowghi and Zhi Jin. *Requirements engineering*. Springer, 2011.

Appendices

A Validation set of requirements texts

A.1 Dental clinic

The clinic basically schedules patients, provides services for them, and bills them for those services.

New patients fill out a form listing their name, address, telephone numbers, allergies, and state of mind prior to scheduling their first appointment. Existing patients are normally scheduled for their next appointment as they depart from their current appointment. When the office staff forget to do this, a desk worker has to call the patient to set up a date. Schedules are entered into a central appointment book; patient records (including contact information) are kept in paper files.

Appointments are for one of three procedures: dental hygiene, cavities and fillings, and oral surgery (including root canals and tooth extractions). For each procedure the patient needs to be prepared and supplies need to be collected (e.g., probes, drill bits, cements, resins, etc.). For a hygienist's appointment, preparation could be as simple as seating the patient in dental chair and putting a bib around his or her neck. For oral surgery, anesthesia of various strengths are normally administered prior to operation. Only for oral surgery procedures is it necessary to ask the patient to wait for up to twenty minutes before performing a post-operative check.

Billing is always done by the month, and bills are always sent by mail to patients' contact addresses. Checks are received by mail. HMO-funded patients are asked to make a copayment at the time that they leave the office. Each patient also generates a reimbursement request to an insurance company. Insurance companies and HMOs send their checks by mail three months after receiving a reimbursement request.

The clinic maintains a supplies inventory file that a worker fills out once a week by physically inspecting each of the three procedures rooms. Supplies and tools are stored in a standard layout in each room.

A.2 Restaurant

Romano's is the finest Italian restaurant in the city. Unless you are a celebrity or a good friend of Romano you will need a reservation. A reservation is made for a specific time, date and number of people. The reservation also captures the name and phone number of the person making the reservation. Each reservation is assigned a unique reservation number. There are two categories of reservations at Romano's: individual reservations and banquet reservations. Additional reservation information captured when an individual makes a reservation includes seating preference (inside or patio) and smoking preference (smoking or nonsmoking). Additional reservation information captured for banquet reservations includes the group name and the method of payment. Seating at Romano's is limited. Romano's has a fixed number of tables. Each table is identified by a unique table number. Each of the tables is further described by a unique free form description such as "located by the North window", "located in front of the

fountain", "by the kitchen door". Each table is classified as a 2-person, 4-person or 6-person table. When a reservation is made, Romano associates a specific tal4 number(s) to the reservation. A table can be utilized many times over the evening by many reservations. Romano tends to overbook tables. Therefore, there can be overlapping table reservations. The management structure at Romano's is hierarchical. There are several restaurant managers who report to Romano. The managers are responsible for managing the Maitre'd and the chefs as well as ensuring that the guests have a pleasant dining experience. The Maitre'd is responsible for managing the waiters, bartenders and bus personnel. The Chefs are responsible for managing the cooks and dishwashers. Each person working for Romano's must be classified as either a manager, Maitre'd, waiter, bartender, chef, cook, bus person or dishwasher. Additional information maintained by Romano's for each person includes the persons name, date of birth and drivers license number. When the reservation party arrives at Romano's the reservation is assigned to one waiter. A waiter can be assigned to many reservations during the course of the evening". The menu at Romano's is exquisite. There are many exciting and exotic items. Each menu item is identified by a unique menu item number. Information maintained by Romano's for each menu item includes an item description of (e.g. "chicken marsala", "fish soup", "endive salad","1988 Merlot wine", etc.), and item prep time. Each menu item is classified by Romano's as "appetizer", "entree", "dessert" or "beverage". The price of each menu item can vary based on the time of day. For example, some of the menu items have different lunch and dinner prices. Some of the menu items change prices for happy hour. In order to calculate the check at the end of the dinner, the waiter maintains a list, by reservation number, of the menu items ordered and the time that the menu item was ordered. In other words, each reservation can be associated with many menu items and a menu item can be associated with many reservations. In addition to menu items, Romano's maintains a list of the food items that are utilized by the restaurant such as chicken, mushrooms, bread sticks, red sauce, cream sauce, etc. Food items are utilized in the preparation of menu items. Each food item-is identified by a unique food item number.

A.3 Observations on geological samples

Retrieved from [47].

Geological samples are retrieved from the field and then processed in the laboratory to determine various properties, including chemistry, mineralogy, age, and petrophysical properties like density, porosity, permeability.

Samples obtained as part of economic activities, such as mineral exploration, are usually processed in commercial assay and chemistry labs. For QA/QC purposes, each batch of samples will have a number of control samples inserted, for which the concentration of particular chemical species are already known. For confidentiality reasons the location information associated with each sample is not provided to the lab, but must be re-attached during the interpretation phase. During processing, many derived samples will be generated by various physical and chemical procedures. In some cases the derived samples are strict sub-samples, whose intensive properties are intended to be the same as the parent. In other cases, the split is 'biased', with the derived sample intended

to select a specific sub-sample, defined by a particular particle size, density, magnetic properties, etc. The link from the derived sample to the parent sample must be preserved, and the link from the parent to the location from which it was obtained also. In some cases the location is associated with another sampling artifact, such as a drill-hole or traverse or cruise, with the latter carrying the detailed location information.

In a research context some samples have a particularly high-value, having been obtained by an expensive process (involving drilling or ships or spacecraft) or from a location that is hard to visit (remote, offshore, in space). These samples are sometimes sub-divided and distributed to multiple research teams or labs for different specialized observations. Each lab will run its own LIMS system, which will usually assign a local identifier for the sample. When the results of these observations are reported, it is necessary that observations from different labs can be correlated with each other, so that the complete picture around each sample can be assembled.

These stories focus on sensing applications involving ex-situ sampling, where a location is visited and a specimen obtained using some sampling process, then transported to one or more laboratories where it is processed into one or more sub-samples and various observations made. Sample identity is usually key, and the relationships between samples, between samples and other artifacts of the sampling process, and also with other geographic features or locations. The sampling time and analysis and reporting time are all different.

Similar process apply to botanical sampling, and to environmental sampling (water, air, dust).

A.4 Law firm

Analyst: Do you run into any challenges with international addresses, given the wide variation in address formats?

AP: Actually, that is a constant source of confusion and pain. We have had several situations where, due to an incorrect address format, the client did not receive correspondence.

Analyst: Can you give me some examples of this?

AP: Yes, one happened recently. Sometimes, addresses start with a house name or number, but some overseas clients put the city or town first. Imagine the confusion of sending something to Paris which we thought was the city but was in fact the house name!

The other issue is that overseas zip codes or postal codes, as they are sometimes called, are not always numeric—they can be a combination of numbers and letters.

Analyst: Tell me about the process when you take on a new client. What information do you initially record about that person?

AP: Remember, it's not always an individual; it could be an organization.

Analyst: When it's an organization, do you always need to know the contact?

AP: Yes, and sometimes there are more than one. For each client or representative of an organization, I need to write down their full name and how they prefer their honorific: Mr., Ms., Mrs., Dr., etc. And of course, their email address, phone number, and postal address, and sometimes we have a primary mailing address that could be different from the billing address.

Analyst: Could you have a situation where multiple people have the same address?

AP: This does happen, for example, when we contact different employees in the same office.

Analyst: What are your other responsibilities, apart from creating cases?

AP: As a case progresses, I need to record all the individuals and organizations that take part in the case activities and the specific role they play.

Analyst: Is it possible for an individual or organization to participate in multiple actions or events in different cases?

AP: Yes. Not only that, but some may play different roles within the same case. For example, the same party can be both the defendant and witness in the same case.

Analyst: So, what are these different types of roles a party can play?

AP: Plaintiff, witness, defendant, judge, an expert in some field, or an attorney.

Analyst: Can your firm's attorneys or judges be considered as parties to the case?

AP: Absolutely. And I must record the information about their involvement as well.

Analyst: Please tell me about events that occur. What information do you record?

AP: All attorneys and legal assistants record their own activities, which include the date and time when an activity occurred, a short description, and a duration, and in the case of witnesses, defendants, and judges, a list of who was involved. We also need to indicate if this event is billable or not.

Analyst: Is there anything else you record for cases?

AP: Yes, we need to know which documents were used in a case.

A.5 Rental truck company

The Right-Way Rental Truck Company rents small moving trucks and trailers for local and one-way usage. We have 347 rental offices across the western United States. Our rental stock includes a total of 5,780 vehicles including various types of trucks and trailers. We need to implement a system to track our rental agreements and our vehicle assignments. Each rental office rents vehicles that they have in stock to customers ready to take possession of the vehicle. We don't take reservations, or speculate on when the customer will return rented vehicles. The central office oversees the vehicle distribution, and directs transfers of vehicles from one rental office to another.

Each rental office has an office name like "Littleton Right-Way". Each office also has a unique three digit office number. We also keep each office's address. Each office is a home office for some of our vehicles, and each vehicle is based out of a single home office.

Each vehicle has a vehicle id, state of registration, and a license plate registration number. We have five different types of vehicles: 36 trucks, 24' trucks, 10' trucks, 8' covered trailers, and 6' open trailers. Yes, we do have a vehicle type code. For all our vehicles, we need to track the last maintenance date, and expiration date of its registration. For our trucks, we need to know the current odometer reading, the gas tank capacity, and whether or not it has a

working radio. For long moves, customers really prefer a radio. We log the current mileage just before we rent a truck, and then again when it is returned.

Most of our rental agreements are for individual customers, but a rental agreement can either be for an individual or for a company. We do rent a small percentage of our trucks to companies. We assign each company an identifying company number and track the company's name and address. No, we don't need to worry about any additional information about a company. Our corporate sales group handles all that information separately.

For each individual customer, we record the customer's name, home phone, address, and driver's license state, number, and expiration date. We like to keep track of all our customers. If a customer damaged a vehicle, abandoned it, or didn't fully pay the bill, then we tag the customer as a poor risk, and won't rent to that customer again.

We only allow a single individual or company for a given rental agreement, and we write a separate rental agreement for each vehicle. Yes, we do have customers rent two or more vehicles at the same time. Each rental agreement is identified by the originating rental office number and a rental agreement number. We also need to track the rental date, the anticipated duration of the rental, the originating rental office, the drop-off rental office, the amount of the deposit paid, the quoted daily rental rate, and the quoted rate per mile. Of course for the trailers, there isn't a mileage charge. No, we don't need to automate the financial side of our business, just our rental agreement tracking and vehicle assignment functions.

B Interface

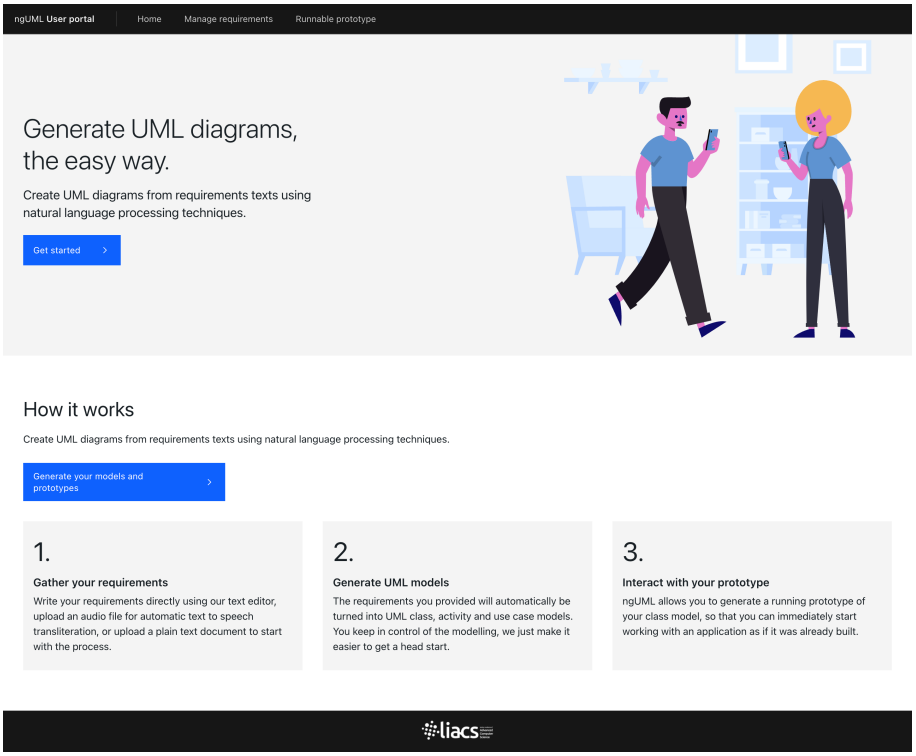


Figure 11: Landing page for the Prose to Prototype project.

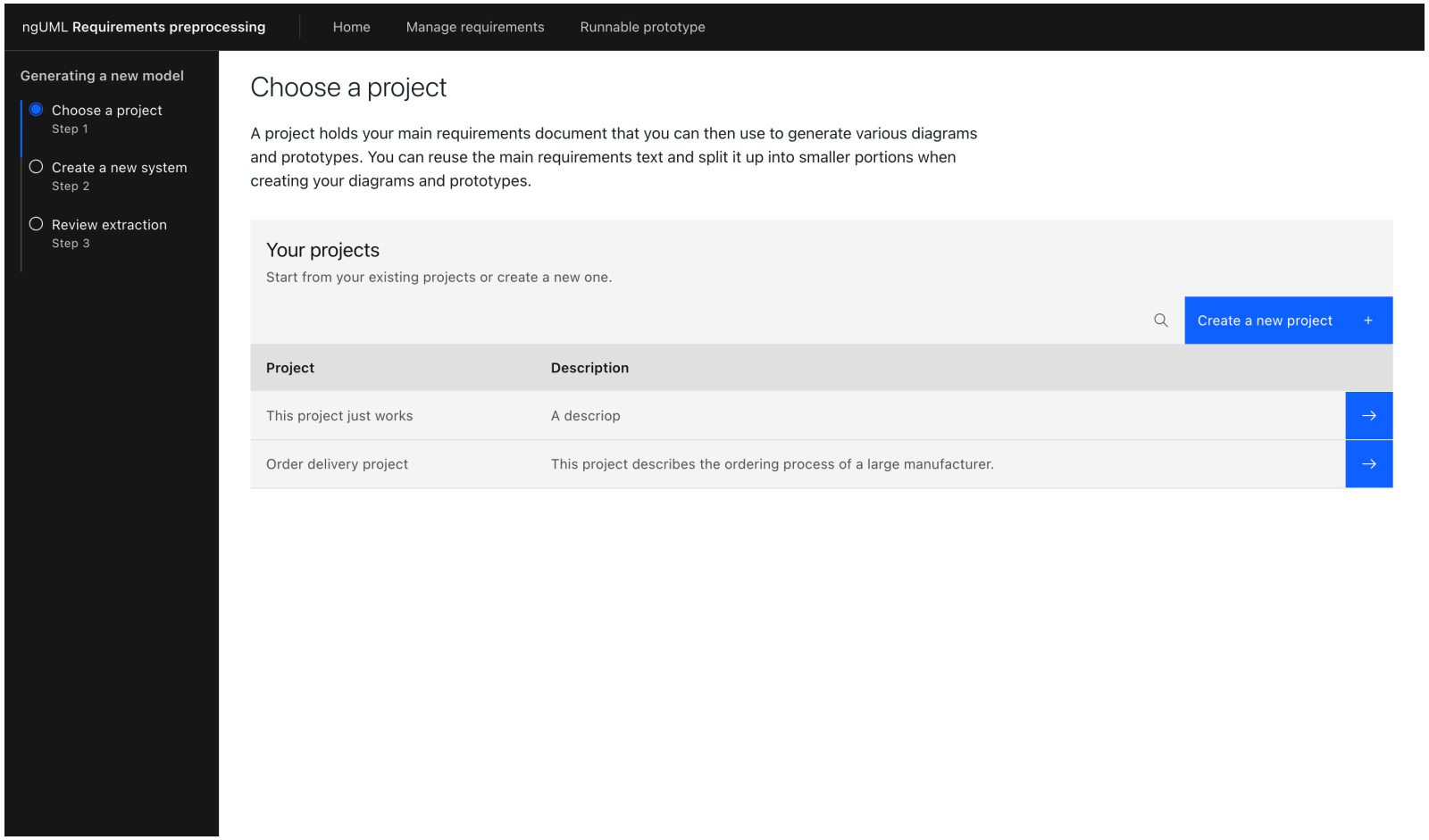


Figure 12: A user has the ability to select projects to get started.

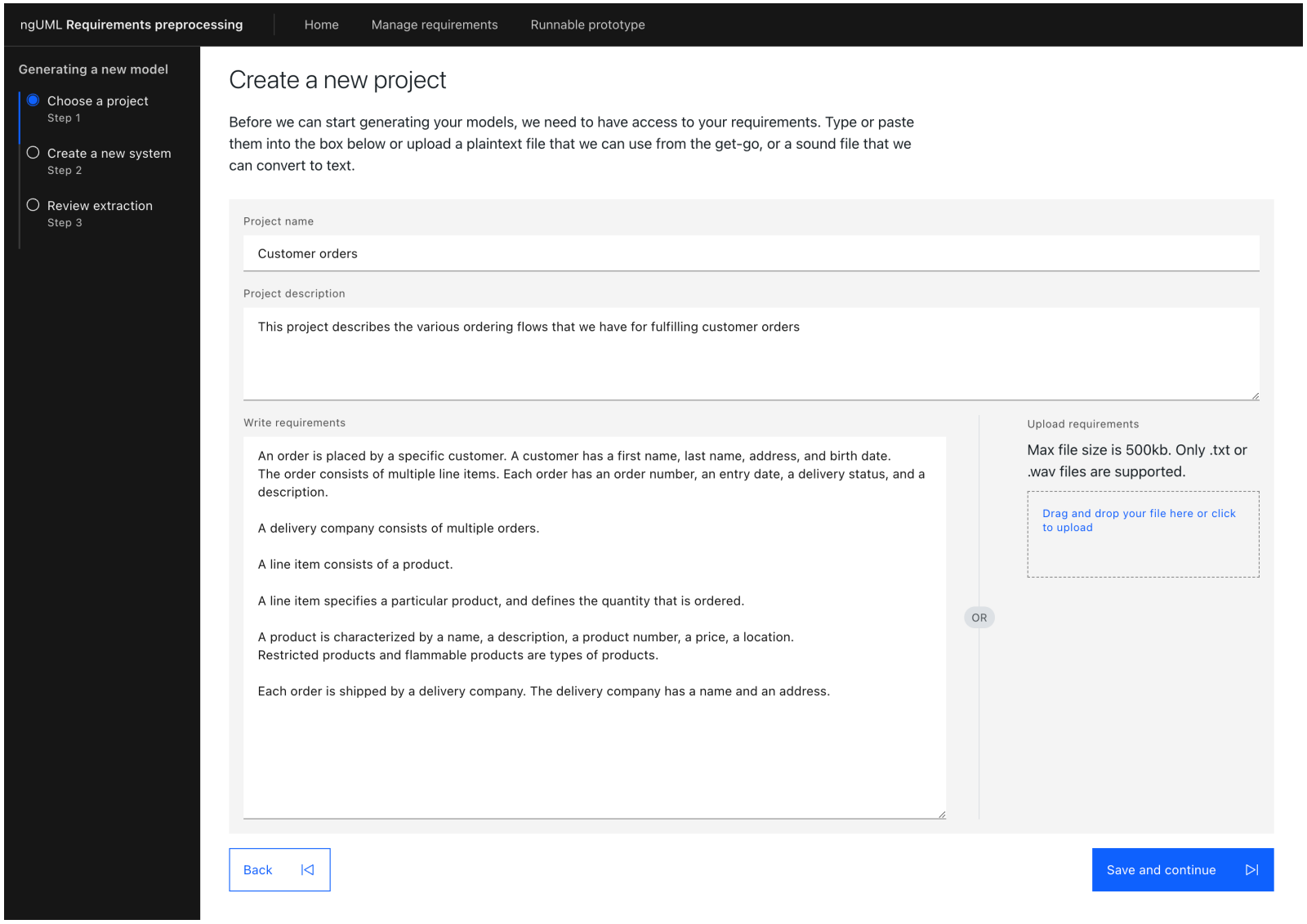


Figure 13: A user can also create new projects.

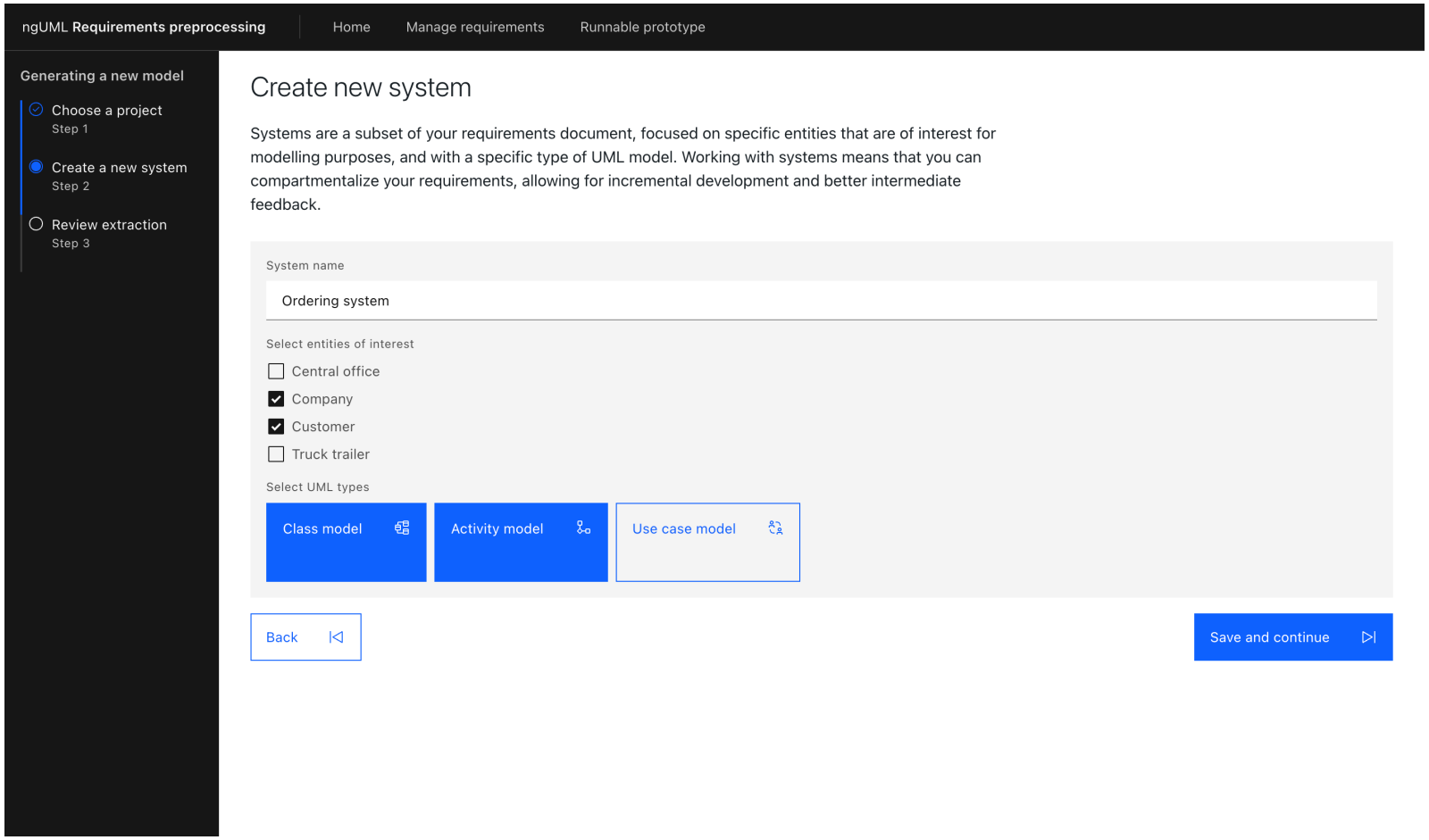


Figure 14: A system can be created to zoom in on specific UML types and entities of interest.

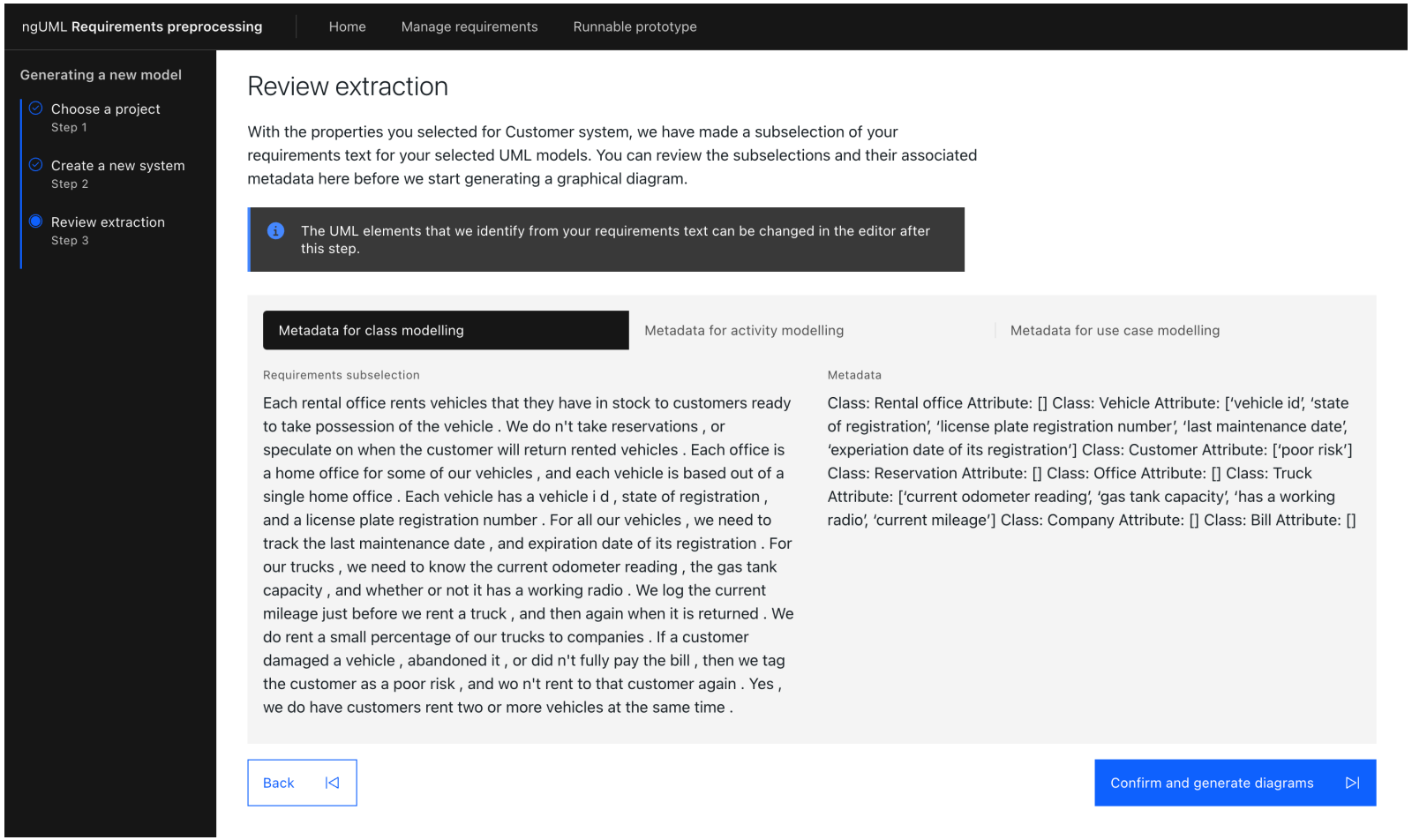


Figure 15: Before generating diagrams, a user can review the extraction.

C Bucketing rules

Below, all bucketing rows are listed for class diagrams, activity diagrams and use case diagrams.

	Rule number	Syntax dependency level	Supersense level	POS level	Lemma level
General rules	1			Has at least two nouns (NN/NNP/NNS)	
Specific rules	2	Has (passive) auxiliary and conjuct dependency relations	Has a verb.cognition supersense		
	3	Has an object of a preposition, a preposition and a nominal subject	Has a verb.stative and a noun.relation supersense		
	4	Has an object of a preposition and a numeric modifier	Has a verb.stative and a noun.artifact supersense and more than one verb.stative supersenses		
	5	Has a nominal subject, a direct object and an adjectival modifier	Has a verb.stative supersense	Has an adjective (JJ)	
	6	Has a determiner, nominal subject and a direct object	Has a verb.possession supersense		
	7	Has a determiner, a nominal subject, an object of a preposition and an auxiliary	Has a verb.stative supersense		
	8	Has a determiner, a nominal subject, an object of a preposition and an auxiliary	Has a verb.communication supersense		
	9	Has a determiner, a nominal subject, a direct object and a coordination relation	Has a verb.change supersense		
	10	Has a determiner, a nominal subject and a direct object	Has a verb.perception and a noun.artifact supersense		
	11	Has a preposition, a passive nominal subject, a passive auxiliary, a numeric modifier and a coordination relation			
	12	Has a passive nominal subject and a passive auxiliary			Has the lemma "require"
	13	Has a passive nominal subject, a passive auxiliary, a determiner and an auxiliary	Has a verb.change supersense		
	14	Has a determiner, a passive nominal subject, an auxiliary, a passive auxiliary and an object of a preposition	Has a verb.contact supersense		
	15	Has a determiner, a nominal subject, an object of a proposition and a preposition	Has a verb.stative supersense		
	16	Has a preposition and more than one object of a preposition	Has a verb.stative and a noun.artifact supersense		
	17	Has a determiner, a nominal subject, a direct object and a predeterminer	Has a verb.social supersense		

Table 6: Bucketing rules for class models.

	Rule number	Syntax dependency level	Supersense level	POS level	Lemma level
General rules	1			Has at least two nouns (NN/NNP/NNS)	
Specific rules	2	Has a nominal subject, a clausal modifier, an agent and a clausal complement	Has a verb.communication supersense and a verb.stative supersense		
	3	Has an adverbial modifier			Has one of the following lemmas: "when", "second", "if", "then", "first"

Table 7: Bucketing rules for activity models.

	Rule number	Syntax dependency level	Supersense level	POS level	Lemma level
General rules	1			Has at least two nouns (NN/NNP/NNS)	
Specific rules	2				Has one of the following lemmas: "system", "facilitate", "module", "interface", "functionality", "capability"
	3		Has one or more verb.contact supersenses		
	4		Has a verb.creation or verb.social supersense		Has the lemma "execute"
	5	Has a nominal subject	Has a noun.animal, noun.person or noun.plant supersense		

Table 8: Bucketing rules for use case models.