Minimal Data, Maximal Impact: Language Model-based Pipelines for the Automatic Generation of Use Case Diagrams from Requirements

Simon Schleifer[1,*], Adriana Lungu[2], Benjamin Kruse[2], Sebastiaan van Putten[2], Stefan Goetz[1], Sandro Wartzack[1]

[1] Engineering Design (KTmfk), Friedrich-Alexander-Universität Erlangen-Nürnberg
[2] AUDI AG

** Corresponding Author:*
Simon Schleifer
Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl für Konstruktionstechnik (KTmfk)
Martensstraße 9
91058 Erlangen
☎ *+49 9131/85-23660*
✉ *schleifer@mfk.fau.de*

Abstract

Striving for unique selling points leads to an increase in product requirements, which are prevalently written in natural language. In order to mitigate inherent ambiguities in these requirements specifications, methods of Model-Based Systems Engineering, like use case diagrams, can be utilized. However, creating model-based requirements is time-consuming. Thus, two novel pipelines for the automatic generation of use case diagrams are proposed and discussed with focus on reducing the amount of needed annotated training data. The first pipeline combines named entity recognition with active learning. The second pipeline utilizes generative large language models and prompt engineering. Both pipelines are exemplarily applied to a requirements specification from the automotive industry.

Keywords

Use case diagrams, Natural language processing (NLP), Large language models (LLM), Model-based systems engineering (MBSE), Requirements engineering (RE)

1. Introduction

The pursuit for unique selling points and fulfilment of sophisticated customer needs leads to an increase in complexity during product development. Requirements engineering contributes to the elicitation and satisfaction of said needs. A requirement describes the need of a stakeholder or a functionality of the system to be developed [1]. System requirements are used to describe latter [2]. Their documentation is still predominantly based on natural language [1], [3]. This allows for comprehensibility without special training and flexible use [1]. Nevertheless, the application of natural language causes problems due to inherent ambiguities [1] and an insufficient controllability of modern product's increasing complexity [3]. For this reason, in the context of Model-Based Systems Engineering (MBSE), model-based requirements are a common approach to overcome said limitations [4], by combining context modelling and behaviour modelling. The context modelling captures the relations between functionalities and actors involved, for example by use case diagrams, while the behaviour modelling details the behaviour of the process described by the requirements. [1]

Creating model-based requirements is primary a manual task so far. For complex products with an increasing number of requirements, this practice is time-consuming and error-prone and thus offers potential for automation [5]. Against this background, the objective of this paper is a contribution towards the automatic generation of use case diagrams for MBSE from textual natural language requirements by applying techniques of natural language processing (NLP) with the aim of reducing the amount of needed training data.

The remainder of this paper is structured as follows. In Section 2 the state of the art is presented. Section 3 derives the emerging research need followed by the detailed explanation of the proposed pipelines in Section 4. The paper closes with the presentation of the results and their critical discussion in Section 5 and a conclusion in Section 6.

2. State of the art

Context modelling by use case diagrams can be used to capture the functionality of a system from an actor's perspective. In the Systems Modeling Language (SysML), it consists of three fundamental elements and relationships between them. The use case itself defines typical interactions of the actors with the system aiming to add value from its use. It is depicted by oval shapes containing the use case name [1]. The interconnection between use cases comprise "include", "extend", and "generalize" relationships [6]. Use cases contribute to the system of interest with their behaviour. The system of interest is modelled by a block [6]. Actors represent entities which are outside to the system of interest and can represent humans, external systems or organizations, among others. Actors interact with the system by the use case, also see examples in Figure 4 to 6. This is modelled with an association relationship [6].

Against this background, there exist numerous approaches assisting the developer with the generation of use case diagrams by applying techniques of NLP. An early work in this research field by DEEPTIMAHANTI AND SANYAL [7] offers the capability to generate use case diagrams based on structured natural language requirements. The approach incorporates reconstruction rules to normalize the requirements, pronoun resolution and extraction rules based on noun and verb phrases. A more recent and extensive approach by TIWARI ET AL. [8] processes requirements with a rule-based engine. It extracts use case name, actors, dependencies, basic and alternative flows, as well as pre- and postconditions. The user's feedback is included by conducting a questionnaire as part of the approach. Both approaches have in common, that the requirements must follow specific sentence templates. A first step towards generalization is taken by MALIK ET AL. [9] by abrogating the need for strict adherence to requirement templates. The proposed approach combines the application of NLP techniques with network science. Former is utilized to extract the relevant information from the natural language requirement text whereas the latter allows linking the entities accordingly.

All of the aforementioned approaches have in common, that the manual definition of rules plays a crucial role for successfully extracting use case diagrams. Even if the limitation to well-structured requirement formulations can be overcome by sophisticated approaches, it is still bounded to the hardly achievable completeness of the defined rules and thus requires significant manual effort. Thus, the application of machine learning based NLP techniques is a promising answer to this. Instead of defining sets of rules, it is sufficient to annotate requirements with labels. VEMURI ET AL. [10] use supervised learning to train a Naive Bayes (NB) classifier which can then identify actor and use case name from the requirement text. The use case diagram is generated based on the results of the NB classifier. This is complemented by mapping the relationships between the use cases. A similar approach is pursued by TIWARI ET AL. [11] who train a named entity recognition (NER) to identify actors and use case names. The authors compare four different machine learning classifiers for this task. However, their research does not comprise relationships between the entities and the generation of a formal use case diagram. VINEETHA AND SAMUEL [12] extend latter approach [11] to multi-word use case names. Furthermore, the here applied multinomial NB classifier yields to better results comparing to TIWARI ET AL. [11]. These approaches require a substantial degree of training, which depends on the availability of both high-quality and high-quantity training data.

A number of strategies have been developed with the objective of addressing the large amount of training data required for the successful application of machine learning algorithms. A potential solution to this is active learning (AL). The iterative learning process builds on an initial training step with little annotated data. A query function identifies data samples which are then labelled by the oracle e.g., a human expert [13]. AL allows to achieve higher accuracies with less labelled training data compared to passive learning [13]. The application of AL in combination with NER is part of existing research [14], [15]. According to BROWN ET AL. [16], task-agnostic generative large language models (LLMs) allow to overcome the finetuning process overall by prompting. Instead of adjusting the weights of the LLM based on labelled training data, the given context of the prompt helps to get the desired results from the LLM. This significantly reduces necessity of labelled training data. Here, few-shot prompting provides several examples of the task within the prompt. The number of examples can be further reduced to one or zero, resulting in one-shot and zero-shot prompting, respectively. Zero-shot prompting is equivalent to merely explanatory prompts [16].

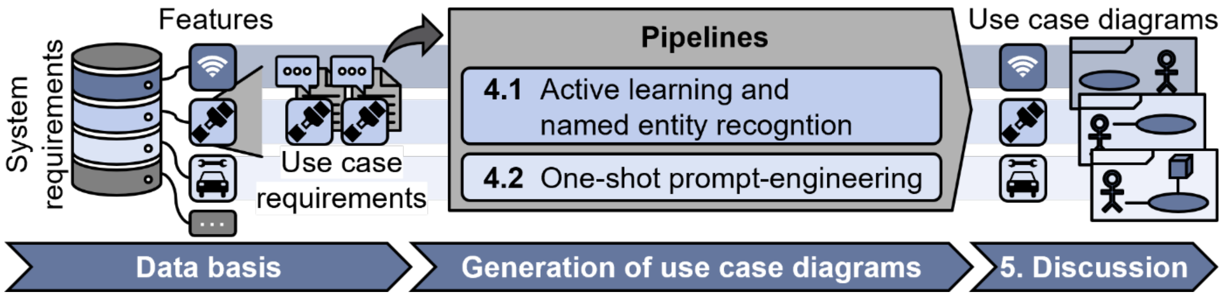
3. Research need

As shown in the previous section, the automatic generation of use case diagrams is part of current research activities. However, the identified approaches often are not suitable for the practical application in an industrial environment. Here, it is necessary to incorporate multiple natural language requirements within a single use case [17]. In the context of large scale practical applications, requirements are generally not standardized but exhibit different sentence templates leading to an inhomogeneous requirements specification [17]. Comparing the practical demands with existing research reveals that rule-based approaches are unsuitable due to the excessive effort involved in establishing the rules manually. This labour-intensive step can be mitigated by applying machine learning algorithms. However, the general application of machine learning techniques requires substantial and sound training data. Existing approaches are either bound to a specific application scenario when using little annotated training data [10] or utilize large training sets requiring over 20 thousand examples [11], [12]. Since gathering suitable training data is expensive and time-consuming [18], reducing the amount of required annotated training data for practical applications is necessary. This leads to the following research question:

How can the dependency on training data be reduced for the automatic generation of use case diagrams from interrelated and inhomogeneous natural language requirements?

4. Automatic generation of use case diagrams without extensive training data

In order to close the gap stated above, two new pipelines for the automatic generation of use case diagrams from natural language requirements are proposed focusing the practical applicability by reducing the amount of required training data. System requirements form the **data basis** and are raised for different features which correspond to superordinate sets of system requirements [1]. Examples are human-machine interaction, passenger comfort, and maintenance. Both **pipelines** are described in detail in the subsequent sections. The necessity and extend of training data as well as the quality of the resulting use case diagrams are subject to the **discussion**, see Figure 1. To keep the focus on the reduction of training data, the scope of the pipelines is limited to simple use case diagrams with one use case and its related entities per diagram each.


System requirements
Features
Use case requirements
Pipelines
4.1 Active learning and named entity recogntion
4.2 One-shot prompt-engineering
Use case diagrams
Data basis
Generation of use case diagrams
5. Discussion
Figure 1: Methodology of this research paper

4.1. Active learning and named entity recognition

Based on the application of NER for extracting relevant entities for use case diagrams as originally described in [11], the here proposed pipeline aims to reduce the needed amount of annotated requirements by utilizing the AL framework in the training step, see Figure 2. The application step utilizes different NLP techniques to generate the use case diagram.

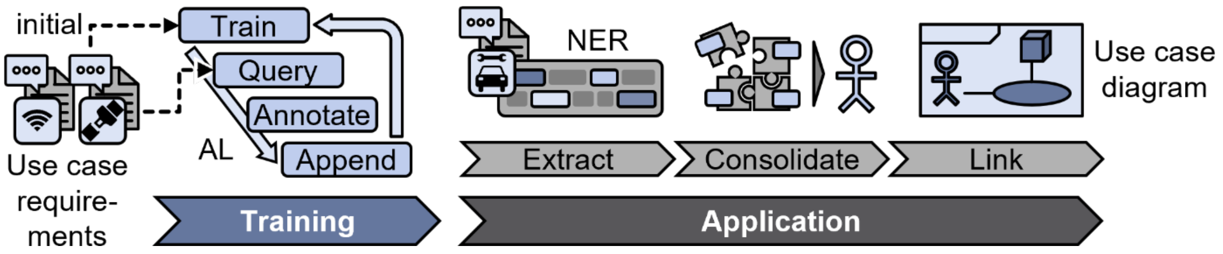


Figure 2: Active learning (AL) and named entity recognition (NER) pipeline for generating use case diagrams

AL starts with an **initial training** step based on a pool of labelled data. Here, the initial training data is labelled following the BIO tagging method marking the beginning, inside, and outside of entities. In contrast to VINEETHA AND SAMUEL [12], the entities are system, actor as well as verb and object as basis for the use case name. Similar to the work of LUO ET AL. [14], a transformer-based pretrained language model is finetuned for the NER task. The here proposed pipeline finetunes the language model “MiniLM” [19]. After the initial training step, a **query** function suggests new training examples if the model accuracy is below a certain threshold. This is the most critical step [15] as newly appended requirements need to be informative in terms of being diverse and representative for the overall requirements specification. To guarantee the positive influence on the model performance and ensure the reduction of training data, a task-specific query function is developed. First, the confidence of the current NER model for the individual entities is calculated on the unseen requirements. By combining all entities, a weighted confidence value is obtained. Second, the least confident requirements for each feature of the training pool are identified for manual labelling. A human expert is required to **annotate** this excerpt of the requirements specification manually. The obtained annotated examples are **appended** to the training pool and additionally included in the model training in the next iteration. Each training iteration begins with the original pretrained language model to prevent overfitting. The iterative nature of AL with focus on informative training examples helps to reduce the amount of manual labelling both in advance and during training.

After training the NER model, it can be applied to unseen data. For the proposed **application**, the NER model generates a candidate list for system, actor, verb, and object forming a use case, by **extracting** one or more entities from each natural language requirement.

The **consolidation** is necessary to reduce the candidate entities to the required elements for the use case diagram. This means a reduction to a single element in terms of the system. For this, a frequency-based approach is selected. If the relative frequency of one candidate system entity exceeds a threshold value, it is selected for the diagram. Otherwise, the most common candidates are assessed in detail to identify potential semantic duplicates. For instance, “vehicle” and “vehicle system” refer to the same system entity. The semantic similarity is estimated via cosine similarity based on embeddings obtained from the Sentence-BERT (SBERT) network [20] using the pretrained transformer model “all-MiniLM” [21]. In case of a cosine similarity close to 1, which indicates high semantic similarity, the most frequent of the reviewed candidates is returned. Otherwise, the developer is required for manual intervention.

To maintain coherent system models, it is common to define sets of approved model elements, such as released actors. The candidate actors are filtered and mapped to the released actors by calculation SBERT embeddings of both. The vectorized representations are examined for similarity by calculating the cosine similarity. If a candidate vector is semantically similar to a released vector, it is appended to the list of elements for the use case diagram. Otherwise, the candidate actor is neglected. This approach yields to a set of actors which are allocated to the use case.

Lastly, the use case name needs to be consolidated based on the extracted verb and object entities. The use case name compounds as verb object tuple representative for the entire set of requirements linked to the specific use case. For this, the online lexical database WordNet [22] is consulted. The basic entries in WordNet are referred to as so-called synsets [23]. A synset covers different semantic meanings of a given word. For example, the NER extracts the verb "adjust" from a requirement. To capture the meaning in the sense of "adjusting the driver's seat", meaning altering a position, the correct synset needs to be identified. For instance, another synset for "adjust" relates to "adjusting to a situation" in the sense of getting used to something. Similar as described in JURAFSKY AND MARTIN [23], the approach for word sense disambiguation uses contextualized word embeddings. The most probable synset for each extracted candidate verb and object entity is identified by comparing the entity in the context of the requirement text with the entity in the context of the synset's example sentence. The synset with the highest cosine similarity is identified as the most suitable fit. Hypernyms have broader meaning [23] and thus are suited to generalize the use case name. An extended list of candidate verbs is generated by adding the hypernyms of the chosen word senses. The extended candidate list is ranked against the originally extracted candidate verbs by word embeddings and cosine similarity in order to obtain the best suited verb for the use case name. Similarly, the object entities are consolidated without deriving hypernyms but instead comparing the semantic similarity amongst the extracted object entities. The combination of both results yields the use case name.

The last step is the rule-based **linking** of the individual entities to generate a simple use case diagram with one use case per diagram.

4.2. One-shot prompt engineering

The second pipeline uses the generative LLM “Mixtral 8x7B instruct” [24] which is instructed by an iterative one-shot prompting strategy. The prompt engineering strategy employs numerous strategies as outlined by FAGBOHUN ET AL. [25], with chain-of-thought factored decomposition prompting being the most relevant. This strategy breaks down complex tasks and helps the model to “think” which leads to more relevant outputs. Figure 3a depicts the prompt engineering strategy in general. Figure 3b gives insight to the alternating structure of user and assistant messages within the individual tasks.

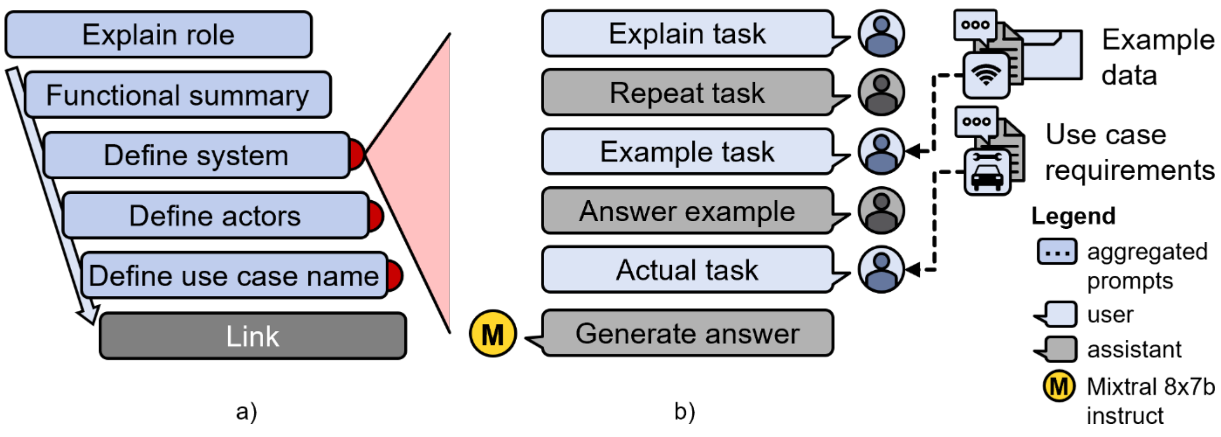


Figure 3: One-shot prompting pipeline for generating use case diagrams

At the beginning, a prompt **explains the role** to the model as this helps to improve the output quality for the specific domain [26]. The LLM is requested to resemble the activities of a developer and create simple use case diagrams based on natural language input requirements. The subsequent prompt instructs the model to create a **functional summary** of the input requirements in order to assess a general understanding of the content of the requirements. One-shot prompting is explained in the following using the example of extracting the system entity. First, a user prompt explains the task of **defining the system** based on input requirements (Explain task). The assistant’s output is simulated by a manually written answer, indicated with the avatar bubble in Figure 3b (Repeat task). The LLM is effectively adopted to the task by providing a one-shot example. This comprises example requirements in the context part of the prompt and the corresponding task of extracting the system in the instruction part. The prompt also requests a standardized answer format (Example task). Again, the output of the assistant is simulated by manually answering in the role of the assistant (Answer example). The simulated answer follows the requested output format which can be referred to the concept of constrained vocabulary [25]. With that, the one-shot part of the prompt chain ends. The prompt containing the unseen input requirements follows the same pattern as before to help the LLM with calculating a suitable output. However, it is explicitly indicated that the preceding example requirements are not subject to the new input requirements (Actual task). The prompt history is then utilized to calculate an output by the LLM indicated with the yellow bubble in Figure 3b (Generate answer). Analogue to this prompting sequence, the LLM is instructed to identify the relevant **actors** to the use case described by the input requirements. The explanatory user prompts are extended by a list of released actors which the LLM is instructed to follow. It is another example of constraining the vocabulary in the prompt. In addition to potential identified actors which correspond to the released actors, it is possible that additional actors are described by the requirements specification. The prompts consider that by asking the LLM whether it is necessary to add potential additional actors and, if so, to generate an output explaining the reason therefor. Lastly, a chain of prompts is created to define a suitable **use case name**. The use case name consists of a verb object pair as in the first approach (see Section 4.1). The instruction of the task prompts emphasizes that the chosen verb object pair must summarize the input requirements with respect to the intended user behaviour. A subsequent rule-based **linking** of the identified use case diagram elements yields the use case diagram.

5. Results and discussion

As part of a consistency analysis, the results of the two proposed pipelines are compared with manual modelling of the underlying requirements by experienced developers. The three example use cases originate from different features to demonstrate the versatility of the pipelines.

Figure 4 shows exemplary the results of a correctly modelled use case with both pipelines. Manual modelling yields the "driver" as associated actor. Pipeline 1 suggests the "user" entity as actor, which is the generalized actor for the "driver". Considering that, the associated actor is viable. Pipeline 2 adds both aforementioned actors to the diagram. Here, a possible enhancement is explicitly taking the hierarchy amongst the actors into account. For both pipelines, the use case name accurately reflects the functionality described by the requirements. This can also be observed by comparing it with the manually created use case name. The definition of the system entity provides sound results.

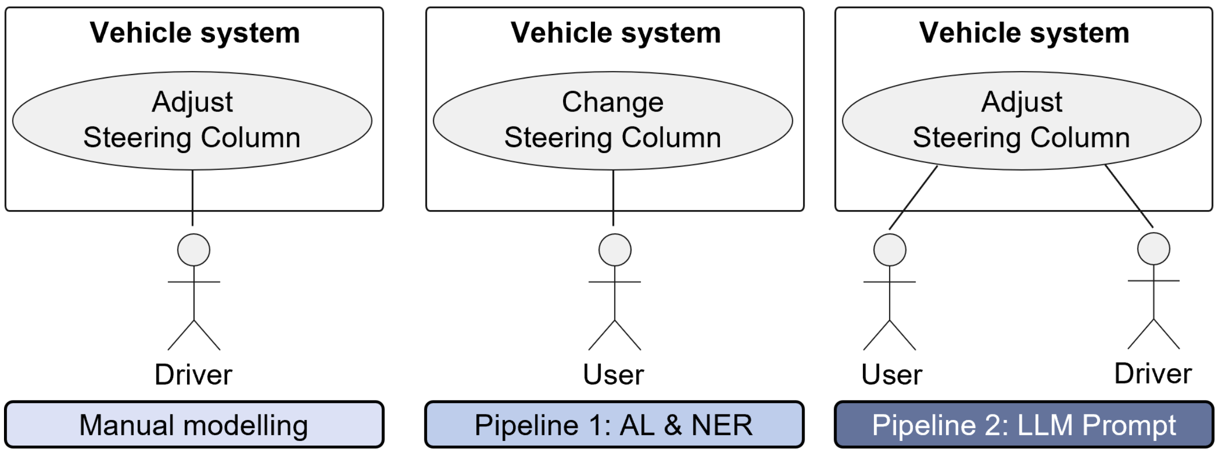


Figure 4: Results for a use case of the feature "passenger comfort"

To evaluate the results of a use case from the feature "human-machine interaction", the results presented in Figure 5 are examined. The pipeline based on NER identifies a multitude of different actors from the set of input requirements. These are mapped to different released actors. However, a meaningful generalization of the results to fewer actors associated to the use case is missing. By comparing with the manual modelling, it becomes apparent that only the most general actor "user" should be part of the use case diagram. This shows that the hierarchy between actors is an important part and must be added to both pipelines. The prompts in pipeline 2 yield to the correct actor. Here, the LLM is capable of calculating a generalized output based on the one-shot example given as input. The first pipeline shows limitations regarding the use case name. The lexical consolidation approach is not capable of finding a representative verb for the input requirements. This is mainly due to limitations of the consolidation algorithm. The word sense disambiguation can result in erroneous synsets and thus unrepresentative hypernyms. This can eventually lead to a poorly generalized verb in the use case name. The object of the use case name however is a suitable result. The second pipeline finds a similar object but also covers the personalization subject. The LLM shows better performance by defining the verb, too. As in the previous result, both pipelines are capable of defining the system entity correctly. Pipeline 1 suggests a synonym of the manual modelled system entity but still allocates the use case in the correct system.

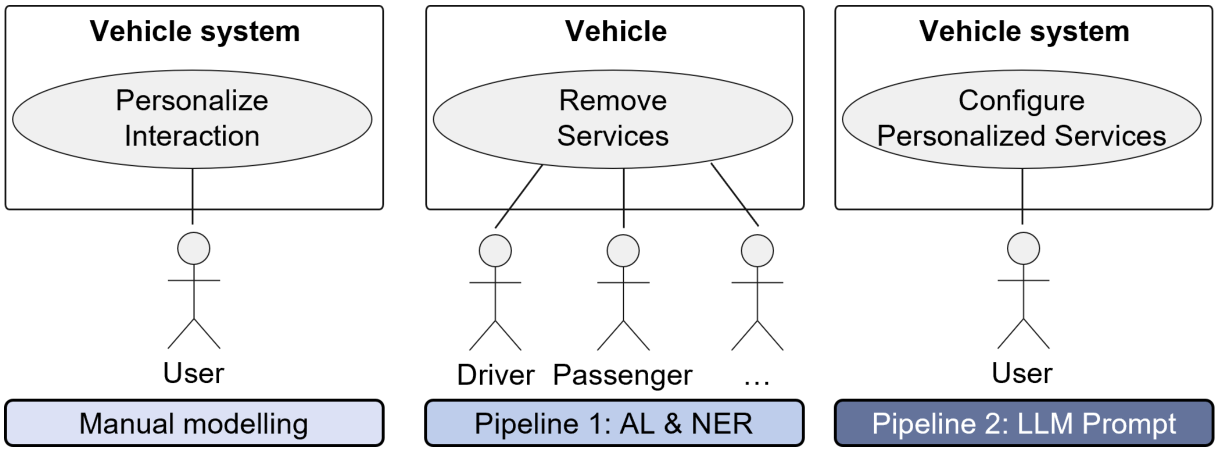


Figure 5: Results for a use case of the feature “human machine interaction”

Lastly, Figure 6 illustrates the results for a use case concerning service appointments for maintenance. The manual modelling associates two released actors to this use case, namely “user” and “workshop”. The trained NER model identifies the candidate actors “customer” and “dealer” from the input requirements but the consolidation approach is not capable of mapping the candidates to the released actors. This can be explained by too little cosine similarity between the embeddings. For this reason, the use case diagram does not display any actors. In contrast to this, the LLM pipeline with its corresponding one-shot prompts is capable of mapping the actors mentioned in the requirements to released counterparts. By reviewing the results for the use case name, it becomes apparent that both pipelines capture the basic functionality. However, both are lacking a hint to the service setting. Lastly, the identification of the related system results in correct solutions in both cases.

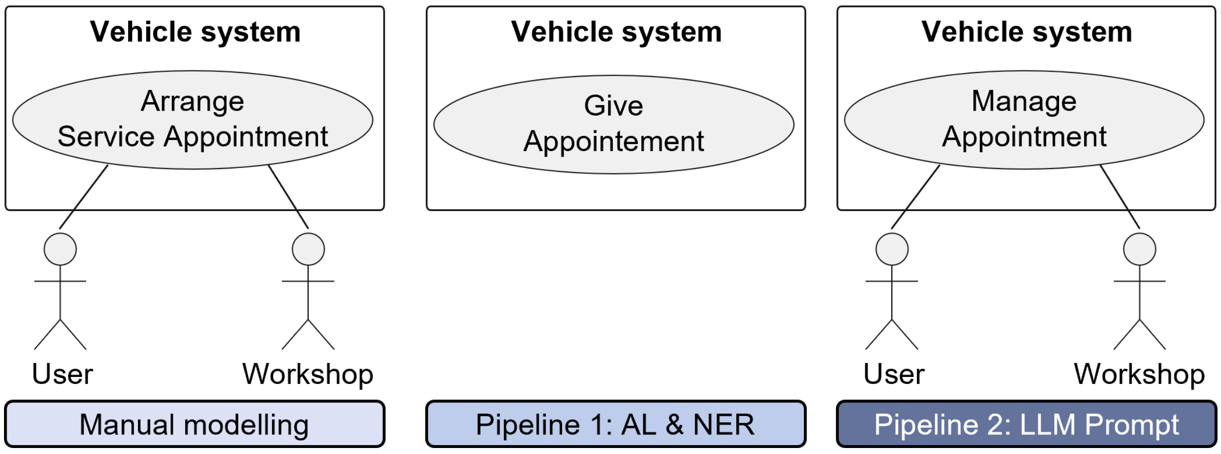


Figure 6: Results for a use case of the feature “maintenance”

It is worth mentioning, that multiple executions of both pipelines yield diverging results. The differences arise more often for the second pipeline but with little impact on the final results as mainly formulations differ. The first pipeline results in fewer variations over multiple executions which can be explained by the lack of generative LLMs involved.

Overall, it can be seen that both pipelines are generally capable of generating use case diagrams from multiple interrelated and inhomogeneous natural language requirements. However, the pipeline based on one-shot prompting leads to more meaningful results. In addition to this consistency analysis, the research question is assessed concerning the amount of required training data for both pipelines. Special focus is put on the retrieval of the annotated data.

For the initial training step of the AL framework for pipeline 1, an initial amount of 100 annotated requirements is utilized. On average five iterations are necessary to achieve an overall F-score measure of 0.85. This is consequently equal to 161 requirements for manual annotation. Training results show best performance for the system entity with an average F-score of 0.95. The F-score on average for the actor, verb and object entity is 0.87, 0.87 and 0.72, respectively. Comparing to VINEETHA AND SAMUEL [12], this is a significant reduction of required training data. The authors report 20,761 tokens whereas the here presented approach requires one fifth thereof with approximately 4,000 tokens for training.

The application of the one-shot prompting pipeline further reduces the amount of training data. There is no need for manually annotating any requirements with entity labels compared to pipeline 1 as an existing use case diagram and its corresponding requirements can be used as one-shot example with little effort. The input requirements can be appended to the prompt template without modification. To derive the simulated answers, it is sufficient to extract the model elements for actor, use case name and system from the example use case diagram.

Nevertheless, limitations remain. The replicability of the results for related research is limited due to the industrial dataset. In general, neither of the proposed pipelines considers a hierarchical relationship between the actors which can cause issues with the generalization, as shown in the results for the first pipeline in Figure 5. Further, multiple executions can lead to different results. In most cases, the derivation of a suitable use case name and the identification of the actors is viable, though. The lexical consolidation approach for the first pipeline lacks a robust generation of the use case name. In terms of the second pipeline, the LLM not always adheres to the requested output format which can cause challenges in post-processing.

Summing up, the proposed research question can be answered as follows: Both the application of NER in conjunction with AL and advanced prompt engineering strategies are suitable to reduce the dependency on training data for the generation of use case diagrams considering industrial requirement sets. Due to the slightly better results of prompt engineering in combination with the least dependency on annotated training data, this approach seems to be most promising.

6. Conclusion and outlook

This publication contributes to the reduction of required labelled training data for the automatic generation of use case diagrams based on natural language requirements. Both the AL and NER-based pipeline and the one-shot prompt engineering pipeline offer a promising solution therefor. Both pipelines reduce the manual preparation to a minimum, with the prompt engineering strategy not even requiring annotating effort at all as simple one-shot examples are sufficient. The applicability is demonstrated by an industrial example.

Further research is necessary to derive complete use case diagrams which comprise more than one use case. This incorporates the relationships between different use cases as well. The one-shot prompting strategy is a promising approach for this. However, it needs to be critically checked how the consistency of the results can be enhanced to minimize deviations in the results. Besides that, the set of requirements forming a use case must be checked for thoroughly describing the functionality regarding pre- and postcondition, main scenario, and alternative scenarios.

Acknowledgement

The authors would like to thank AUDI AG for supporting and funding the research project.

References

[1] Pohl, Klaus; Rupp, Chris: Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level. Heidelberg: dpunkt.verlag, 2021.

[2] Soares, Michel; Vrancken, Jos: Model-Driven User Requirements Specification using SysML. In: Journal of Software Vol. 3 (2008), No. 6, p. 57-68.

[3] Weber, Matthias; Weisbrod, Joachim: Requirements engineering in automotive development: experiences and challenges. In: IEEE Software Vol. 20 (2003), No. 1, p. 16–24.

[4] Inkermann, David, et al.: Model-Based Requirement Engineering to Support Development of Complex Systems. In: Procedia CIRP Vol. 84 (2019), p. 239–244.

[5] Ahmed, Sharif; Ahmed, Arif; Eisty, Nasir U.: Automatic Transformation of Natural to Unified Modeling Language: A Systematic Review. In: 20th International Conference on Software Engineering Research, Management and Applications (SERA). IEEE, 2022, p. 112-119.

[6] Friedenthal, Sanford; Moore, Alan; Steiner, Rick: A practical guide to SysML: the systems modeling language. Amsterdam; Boston: Elsevier, MK, 2015.

[7] Deeptimahanti, Deva Kumar; Sanyal, Ratna: Semi-automatic generation of UML models from natural language requirements. In: Proceedings of the 4th India Software Engineering Conference. 2011, p. 165–174.

[8] Tiwari, Saurabh; Ameta, Deepti; Banerjee, Asim: An Approach to Identify Use Case Scenarios from Textual Requirements Specification. In: Proceedings of the 12th Innovations on Software Engineering Conference. New York: Association for Computing Machinery, 2019, p. 1–11.

[9] Malik, Maryam I.; Sindhu, Muddassar A.; Abbasi, Rabeeh A.: Extraction of use case diagram elements using natural language processing and network science. In: PLoS ONE Vol. 18 (2023), No. 6.

[10] Vemuri, Sandeep; Chala, Sisay; Fathi, Madjid: Automated use case diagram generation from textual user requirement documents. In: 30th Canadian Conference on Electrical and Cumputer Engineering. IEEE, 2017.

[11] Tiwari, Saurabh, et al.: Identifying Use Case Elements from Textual Specification: A Preliminary Study. In: 28th IEEE International Conference on Requirements Engineering. IEEE, 2020, p. 410–411.

[12] Vineetha, V. K.; Samuel, Philip: A Multinomial Naïve Bayes Classifier for identifying Actors and Use Cases from Software Requirement Specification documents. In: 2nd International Conference on Intelligent Technologies (CONIT). IEEE, 2022.

[13] Settles, Burr: From Theories to Queries: Active Learning in Practice. In: Active Learning and Experimental Design workshop In conjunction with AISTATS 2010: JMLR Workshop and Conference Proceedings, 2011, p. 1–18.

[14] Luo, Haocheng, et al.: Re-weighting Tokens: A Simple and Effective Active Learning Strategy for Named Entity Recognition, arXiv (2023).

[15] Şapci, Ali O.B, et al.: Focusing on potential named entities during active label acquisition. In: Natural Language Engineering Vol. 30 (2023), No. 3, p. 602-624.

[16] Brown, Tom, et al.: Language Models are Few-Shot Learners. In: Advances in Neural Information Processing Systems Vol. 33 (2020) p. 1877–1901.

[17] Schleifer, Simon; et al.: Automatic Derivation of Use Case Diagrams from Interrelated Natural Language Requirements. In: Proceedings of the Design Society Vol. 4 (2024) p. 2725-2734.

[18] Bansal, Aayushi; Sharma, Rewa; Kathuria, Mamta: A Systematic Review on Data Scarcity Problem in Deep Learning: Solution and Applications. In: ACM Computing Surveys Vol. 54 (2022), No. 10s, p. 1-29.

[19] Wang, Wenhui, et al.: MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers, arXiv (2020).

[20] Reimers, Nils; Gurevych, Iryna: Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks, arXiv (2019).

[21] huggingface: sentence-transformers/all-MiniLM-L6-v2. URL https://huggingface.co/sentence-ransformers/all-miniLM-L6-v2 (accessed on 2024-03-01).

[22] Princeton University: „About WordNet“. URL https://wordnet.princeton.edu (accessed on 2024-06-19).

[23] Jurafsky, Daniel; Martin, James H.: Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Third Edition draft (2023).

[24] Jiang, Albert Q., et al.: Mixtral of Experts, arXiv (2024).

[25] Fagbohun, Oluwole; Harrison, Rachel M.; Dereventsov, Anton: An Empirical Categorization of Prompting Techniques for Large Language Models: A Practitioner's Guide. In: Journal of Artificial Intelligence, Machine Learning and Data Science Vol. 1 (2023), No. 4, p. 1–11.

[26] Bsharat, Sondos M.; Myrzakhan, Aidar; Shen, Zhiqiang: Principled Instructions Are All You Need for Questioning LLaMA-1/2, GPT-3.5/4, arXiv (2024).