Elicitation and Classification of Security Requirements for EVerest

Master Thesis of

Debora Maria Marettek

At the KIT Department of Informatics
KASTEL – Institute of Information Security and Dependability

First examiner:	Prof. Dr.-Ing. Anne Koziolek
Second examiner:	Prof. Dr. Ralf Reussner
First advisor:	M.Sc. Sophie Corallo
Second advisor:	Dr.-Ing. Tobias Hey

13. November 2023 – 13. May 2024

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Abstract

Incomplete and unverified requirements can lead to misunderstandings and misconceptions. Specifically in security, violated requirements can be indicators for potential vulnerabilities. To avoid vulnerabilities, security reuqirements are linked to their implementations. For further verification, security analyses can be used to check whether a software design fulfills its required properties. Therefore, specific attributes of requirements have to be identified and linked to the design. In this thesis, security requirements for the open-source software EVerest are elicited. EVerest provides a full stack environment for electric vehicle charging stations. For the elicitation, a two-step approach is used. First, a questionnaire is developed that elicits coarse-grained requirements of the security categories confidentiality, integrity, availability, and authentication. Afterwards, four EVerest software developers are interviewed to refine the coarse-grained requirements to 93 design-level security requirements. Prompt engineering and fine-tuning are used to classify design elements and extract their respective mentions from the retrieved requirements. GPT's quality for the design element classification and extraction of their mentions is determined in three evaluation scenarios both for individual elements and per requirement. First, it is examined how well GPT classifies the elements in the requirements. Then, GPT's ability to extract their mentions in the requirements is determined. Finally, it is examined how well GPT extracts the mention of the design element in the requirement in case the design element is correctly classified. The results indicate that regarding multi-class design element classification in requirements works well in prompt engineering and fine-tuning (F1-score: 0.67-0.73). However, regarding the extraction of design elements, fine-tuning (F1-score: 0.7) surpasses prompt engineering (F1-score: 0.52). If both tasks are combined, fine-tuning (F1-score: 0.87) also outperforms prompt engineering (F1-score: 0.61). However, in prompt engineering, GPT seems to be inappropriate to classify all design elements or extract all mentions per requirement, while the results in fine-tuning for the design element classification (F1-score: 0.18-0.2) and the mention extraction (F1-score: 0.15-0.18) are acceptable for a multi-label classification.

Zusammenfassung

Unvollständige und nicht überprüfte Anforderungen können zu Missverständnissen und falschen Vorstellungen führen. Gerade im Sicherheitsbereich können verletzte Anforderungen Hinweise auf potenzielle Schwachstellen sein. Um Schwachstellen zu vermeiden, werden Sicherheitsanforderungen an deren Implementierung geknüpft. Zur weiteren Verifizierung kann mithilfe von Sicherheitsanalysen überprüft werden, ob ein Softwaredesign seine geforderten Eigenschaften erfüllt. Daher müssen spezifische Anforderungsattribute identifiziert und mit dem Design verknüpft werden. In dieser Arbeit werden Sicherheitsanforderungen für die Open-Source-Software EVerest erhoben. EVerest bietet eine Full-Stack-Umgebung für Ladestationen für Elektrofahrzeuge. Für die Erhebung wird ein zweistufiger Ansatz verwendet. Zunächst wird ein Fragebogen entwickelt, der grobkörnige Anforderungen der Sicherheitskategorien Vertraulichkeit, Integrität, Verfügbarkeit und Authentifizierung erhebt. Anschließend werden vier Softwareentwickler von EVerest interviewt, um die grobkörnigen Anforderungen auf 93 Sicherheitsanforderungen auf Designebene zu verfeinern. Mithilfe von Prompt Engineering und Fine-tuning werden Designelemente klassifiziert und ihre jeweiligen Erwähnungen aus den erhobenen Anforderungen extrahiert. Die Qualität von GPT für die Klassifizierung und Extraktion von Designelementen wird in drei Bewertungsszenarien sowohl für einzelne Elemente als auch pro Anforderung ermittelt. Zunächst wird untersucht, wie gut GPT die Elemente in den Anforderungen klassifiziert. Anschließend wird die Fähigkeit von GPT bestimmt, die Erwähnungen von Designelementen in den Anforderungen zu extrahieren. Abschließend wird untersucht, wie gut GPT die Erwähnung des Designelements in der Anforderung extrahiert, wenn das Designelement korrekt klassifiziert ist. Die Ergebnisse deuten darauf hin, dass die Mehr-Klassen-Klassifizierung von Designelementen in Anforderungen sowohl bei Prompt Engineering als auch bei Fine-tuning gut funktioniert (F1-Score: 0,67-0,73). In Bezug auf die Extraktion von Designelementen übertrifft Fine-tuning (F1-Score: 0,7) jedoch Prompt Engineering (F1-Score: 0,52). Wenn beide Aufgaben kombiniert werden, übertrifft Fine-tuning (F1-Score: 0,87) ebenfalls Prompt Engineering (F1-Score: 0,61). Im Prompt Engineering scheint GPT jedoch ungeeignet zu sein, um alle Designelemente zu klassifizieren oder alle Erwähnungen pro Anforderung zu extrahieren, während die Ergebnisse im Fine-tuning für die Klassifizierung der Designelemente (F1-Score: 0,18-0,2) und die Extraktion der Erwähnungen (F1-Score: 0,15-0,18) akzeptabel für eine Mehr-Label-Klassifizierung sind.

Contents

List of Figures

List of Tables

1. Introduction

There are multiple stages in the process of requirements engineering. First, the requirements engineer elicits the requirements for the system to be built from all relevant stakeholders. In this step, their needs and expectations of the system to be built are discovered [74]. Then, the requirements are documented to persist them, so that stakeholders can discuss them and agree to a version that should be implemented [58]. Usually, natural language is used to guarantee that all stakeholders understand the requirements. An important task in software engineering is the verification of the requirements [34]. Requirements are verified to check whether they are fulfilled within the implemented system [35]. This is particularly important to ensure that clients are satisfied with the software. However, only existing requirements can be verified. Therefore, the elicitation of requirements is also important for the verification.

Poor and incomplete requirements elicitation can lead to unsatisfied stakeholder expectations. This can even result in project failure [74], as mistakes during the requirements phase are much more costly than if they occur later in the development process [67]. Specifically, concerning the security of the software, it is important to distinguish possible threats and vulnerabilities early in the process to derive adequate security requirements. If a security incident happens because of missing requirements, this can lead to possible loss of money or reputation [67]. An example of that is *CardSystems Solutions*, a credit card processing company, which had a severe security issue in 2005 [14]. The problem was that they did not require the separation of historical and account data in their security planning. Thereby, historical data was stored along with account data and the attackers were able to compromise the information of 40 million credit cards. At first, it was assumed that *CardSystems Solutions* would be charged with a heavy fine of several million dollars [29]. This was not the case as the company was already financially ruined. Instead, *CardSystems Solutions* was obliged to have regular security checks carried out by independent third parties [9]. It is therefore noticeable, that requirements elicitation is essential. The engineer has to ask the right questions to all relevant stakeholders and needs therefore enough domain knowledge [56].

Even though requirements are elicited completely, they must be verified. This step is often supported by traceability approaches that link requirements to other software artefacts, e.g., the architecture model. This can include, for example, the search for different element types in the requirement so that they can later be mapped to architectural elements. A trace link can then be established between the requirement and the architecture [27]. The conceptual model SecLan[57] provides an overview of elements used in security specifications on software artifacts. A classification of the requirements into its element types can be a first step towards requirements verification. Carrying out a classification manually is error-prone and time-consuming [73]. Several works performed classifications based on machine learning algorithms e.g., decision tree and support vector machine (SVM) [40, 31, 65]. To train models with machine learning, large datasets are necessary. However, especially in security, data is often highly confidential and not publicly available. In contrast to SVMs, large language models (LLMs), like GPT, are already trained on huge datasets and are ready for use.

For this work, the software project EVerest was selected to elicit and classify security requirements. EVerest is an open-source software for electric vehicle (EV) charging stations [70]. Currently, the EV sector is booming, and the number of people driving electric vehicles continues to increase [53]. With this trend, there are also many risks and potential threats that have to be taken into account, e.g., acquiring sensitive user data. Therefore, security is an important issue in the EV sector [59]. The aim of this thesis is to elicit the security requirements of EVerest. Furthermore, it is investigated how well architectural elements, mentioned in these requirements, can be identified. The classification and extraction tasks are carried out by a large language model that applies natural language processing techniques. GPT is currently one of the largest LLMs available and thus presumably a very powerful model, and is therefore used in this thesis. First, a classification of security-related architectural elements is done. As a taxonomy, elements of the SecLan[57] model are used. Second, the mentions of these elements in the requirements should be extracted to ease the later identification and tracing of these elements. This leads to the following research questions:

1. What are the security requirements of EVerest?
2. How well can GPT map the security requirements of EVerest to system model elements of SecLan?
   a) How well does GPT classify SecLan's element types in the requirements?
   b) How well does GPT extract the indicative phrases in the requirements?
   c) How well does GPT extract indicative phrases in dependence of its requirement element types?

The foundations for this thesis are in Chapter 2. Chapter 3 provides the related work in elicitation of requirements as well as information extraction that is needed for the classification. EVerests architecture and components are introduced in Chapter 4. Chapter 5 describes the process of the security requirements elicitation for EVerest, including the design and conduction of a questionnaire and interviews. Their results are presented in this chapter as well. In Chapter 6, the classification approach of this work is described and evaluated. Moreover, the evaluation results of prompt engineering and fine-tuning are presented and compared. Finally, Chapter 7 concludes and summarizes the findings of this thesis.

2. Foundations

This chapter provides an overview of the foundations of this thesis. First, techniques for requirements elicitation such as questionnaires and interviews are examined as they are applied in this work. Then, the tool LimeSurvey and the concepts of the Palladio Component Model, which are used in this thesis, are presented. Section 2.4 introduces the software EVerest, by which the security requirements are elicited. The classification of the requirements is based on the model elements of SecLan's[57] system model which is described in Section 2.5. Moreover, foundations for natural language processing are introduced as the elicited security requirements are in natural language. Furthermore, large language models (LLMs) are described, and two options for how to use them for specific tasks. LLMs can be used to classify text with prompt engineering and fine-tuning. Finally, metrics that are used for the evaluation of the LLM are introduced in Section 2.8.

2.1. Requirements Elicitation

Requirements elicitation is the first step of the software development process [51]. It involves identifying stakeholders, their needs, and requirements [74]. The requirements engineer should have solid domain knowledge and communication skills to gather correct, complete, and unambiguous information from the stakeholders [4]. Traditionally, questionnaires, and interviews are often used [74]. However, there is no requirements elicitation technique that guarantees to find all requirements. So the best choice is to use multiple techniques.

2.1.1. Questionnaires

Questionnaires are a method to efficiently collect qualitative data. They can be conducted anonymously, so participants can be honest and open [48]. At the beginning of a questionnaire, there should be an introduction that also states the aim of the survey [7, 48]. Dependent on the purpose of the questionnaire, different kinds of questions can be used, e.g., open and closed questions [77]. In open questions, people are asked to formulate free-text responses, whereas in closed questions predefined answer options are given. Questions should be formulated clearly, well-defined, and precise to avoid misunderstandings [74]. Furthermore, it is important to conduct a pilot test for the questionnaire to ensure that the participants understand the questions, and if not, to be able to reformulate them if necessary. The terminology of the domain should be understood by the participant and the questionnaire designer [77]. The designer needs to have enough domain knowledge to prepare the questionnaire [16]. Thus, especially for novices, the reuse of existing questionnaires is recommended to ensure the validity and reliability of the questionnaire [48].

2.1.2. Interviews

There are three types of interviews, namely the structured, the unstructured, and the semi-structured interview [77]. In structured interviews, the questions are predefined and only a few additional questions for clarification are asked. This does not allow the participant to share new ideas freely [74]. Unstructured interviews are similar to discussions on a specified theme. They hold the risk of neglecting topics and focusing too much on details in some areas [77]. The semi-structured interview is a combination of the other two types. Both predefined and unplanned questions are asked in semi-structured interviews. This method can help uncover problems that were not known to the interviewer [74]. Independent of the type of interview, there are several things that need to be considered for its conduction. First, a shared terminology of the domain is important. Then, the interview should begin with a good opening so that the participant feels comfortable. Questions in the right order and a summary of the findings at the end to confirm understanding mark a good interview [4]. Finally, the interviewer should be open-minded and should not approach the interview with preconceived ideas about possible requirements to prevent the participant from becoming biased [36].

2.2. LimeSurvey

LimeSurvey is a free, open source software application for creating surveys [46]. Anyone can host a server to conduct their own surveys and control the environment. Furthermore, LimeSurvey provides a lot of features for the creation of questionnaires, such as dynamic text field appearance and random order of question groups [46]. Participants are guided through the survey and can easily fill forms and questionnaires. For this thesis, a local hosted LimeSurvey was used.

2.3. Palladio Component Model

The Palladio component model (PCM) is a metamodel and a modelling language that is implemented in the Eclipse Modeling Framework (EMF). It is developed to specify software architectures. The software Palladio uses the PCM to predict the performance of a software application early in the development process based on its architecture. Design decisions can then be made based on these results [43]. The Palladio component model follows the principles of component based software engineering (CBSE). Components require and provide interfaces [43]. An example can be seen in Figure 2.1. *ComponentA* provides *Interface1*, whereas *ComponentB* requires it. This means, that *ComponentA* offers a service through *Interface1*, so that other components such as *ComponentB* are able to use it. The fact that *ComponentB* requires *Interface1* indicates that it needs the services of the interface to work.

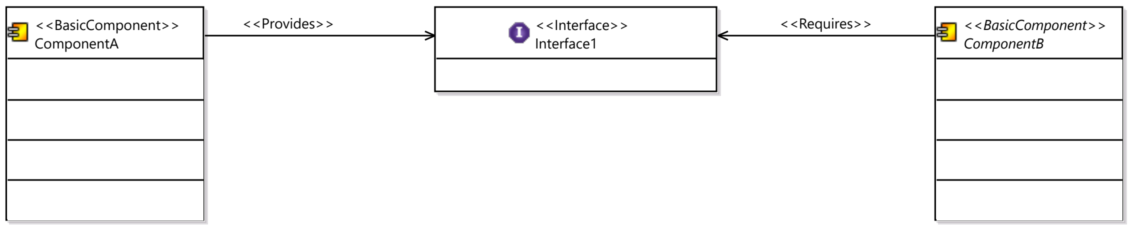


Figure 2.1.: Example Components with Provided/Required Interface

2.4. EVerest

EVerest is a software that provides an open-source full-stack implementation for electric vehicle (EV) charging stations. It is a framework that offers the option to choose between different modules or implement a new module to configure the respective charging scenario. EVerest was founded by the company PIONIX and is a project of the Linux Foundation Energy. EVerest aims to facilitate the development of EV charging and increase efficiency by using the advantages of open-source [70]. Stakeholders of EVerest are, among others, companies and collaborators that use the software or are interested in using EVerest, as well as developers of PIONIX. EVerest's GitHub consists of several repositories, including *EVerest*, *everest-framework*, *everest-core*, *libocpp*, and *libevse-security*. The *EVerest* repository contains the documentation and information about contributing, the code of conduct, and how to join the community [25]. *Everest-framework* is responsible for starting and managing the modules and their dependencies [19]. The *libocpp* repository contains an implementation of the open charge point protocol (OCPP) standard for version 1.6 and 2.0.1 [21]. Security related operations such as certificate signing requests can be found in the *libevse-security* repository [20]. The *everest-core* repository includes all available modules for charging scenarios [18].

2.5. Security Metamodel SecLan

SecLan is a conceptual model of security specifications on software artifacts [57]. In this thesis, SecLan is used for the classification of security requirements. SecLan aims at creating a relationship between system elements, which are analyzed by security checks, and security domain specific languages (DSLs), which annotate the element types. Its

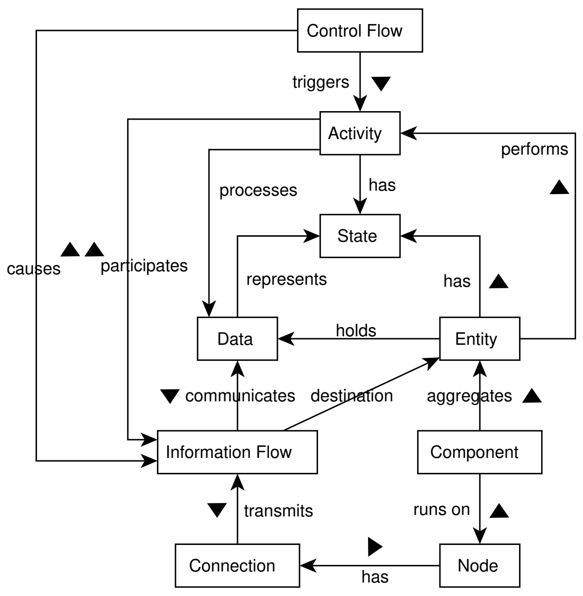


Figure 2.2.: SecLan system model [57]

intention is to improve the understanding of how secure design and implementation are related and to justify the existing relationships. SecLan consists of four parts, the *security model*, the *security domain specific language description*, the *security analyzer description*, and the *system model*. The *security model* includes common concepts in software security such as a security objective or a threat. The *security domain specific language description* provides security-related elements that should be considered in a security check. The *security analyzer description* offers insights about the security checks. The *system model* includes security-relevant system elements that are investigated by the security check. In this thesis, the elements of the SecLan *system model* are used to classify the requirements. The *system model* contains nine different types of model elements that are related to each other and can be seen in Figure 2.2. *Data* contains information and can, for example, represent a *state*. *State* is defined as the state of an *activity* or *entity*. Entities hold *data*. They can have a *state* or perform an *activity*. An *entity* can be a physical actor, a software object, or an external system. A *component* aggregates one or multiple entitites with a specified aim and it can be deployed on a *node*. Nodes are physical devices that can have connections to other nodes. A *connection* can transmit *data* via an *information flow*. The latter communicates data between entities or activities. An *activity* processes *data*, can communicate with other activities, and has a *state*. They are orchestrated and executed in an order that is defined in a *control flow* [57].

2.6. Natural Language Processing

Natural language processing (NLP) refers to a set of techniques for the automatic processing of natural language [11]. Usually, pre-processing of a text is required to prepare it for

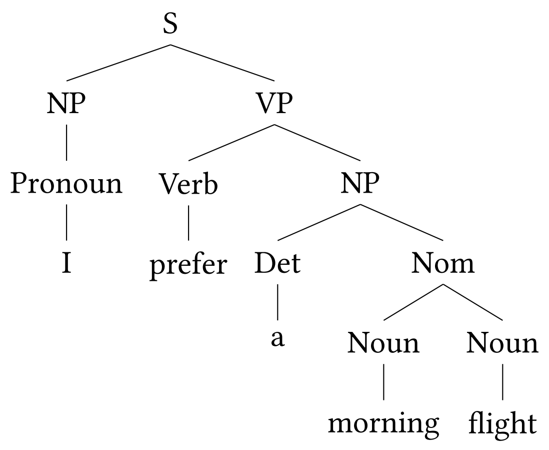


Figure 2.3.: Exemplary syntax tree of “I prefer a morning flight”

syntactic or semantic analysis. Common techniques include tokenization, and part-of-speech (POS) tagging [64]. Tokenization divides the text into tokens such as meaningful words, but also punctuation, symbols, and characters. Then, part-of-speech tagging can be carried out based on the tokenized text. This involves assigning a grammatical category to each word in the text such as noun, verb, adjective, or adverb. The POS tags of the PENN Treebank are typically used for this purpose. The syntax analysis looks for patterns in a sentence, such as a noun or verb phrase. For that purpose, a syntax tree is created that uses POS tagging. A noun phrase (NP) contains a noun and its corresponding modifiers such as adjectives, articles, and pronouns before or after the noun. The verb phrase (VP) consists of a main verb with additional auxiliary verbs [44]. An example of a syntax tree of the sentence “I prefer a morning flight” can be seen in Figure 2.3. Syntax trees form a significant intermediate step for information extraction [41]. Information extraction is an important task in natural language processing [62]. Its objective is the extraction of structured text from unstructured plain text [69]. This filters out interesting knowledge from plain text. There are various information extraction tasks such as named entity recognition, relation extraction, event extraction, and aspect-based sentiment analysis to enable different target results of information extraction [30].

2.7. Large Language Models

Language Models (LM) are a general term for models processing and learning natural language. LMs that are pretrained with massive amounts of data are so-called large language models (LLM) [28]. They excel at solving natural language processing tasks. The currently most promising LLMs are all transformer-based, but differ in their internal structure. There are encoder-only LLMs like BERT (Bidirectional Encoder Representations from Transformers), encoder-decoder LLMs like T5, and decoder-only LLMs like GPT (Generative Pre-trained Transformer) [32]. The GPT series are LLMs created by the company OpenAI. The latest version, GPT 4, was released in March 2023. It considerably outperforms earlier versions like GPT 3.5 turbo on existing benchmarks [54]. GPT 4 was trained on a large dataset taken from the internet. As its predecessors, GPT 4 also suffers from halluzinations, meaning it asserts facts that are not true. Moreover, GPT's responses are non-deterministic. This means that if the same request is sent to GPT twice, a slightly different response is returned. Interaction with GPT can be carried out via the OpenAI API. Different approaches to use GPT are prompt engineering and fine-tuning.

2.7.1. Prompt Engineering

In prompt engineering, a prompt is created that indicates what GPT should do. The prompt is continuously refined to optimize the performance of the specific task [10]. The quality of the prompt determines the quality of the generated output of the LLM [71]. There are different prompt engineering tasks, including zero-shot, few-shot, and chain-of-thought prompting. In zero-shot prompting, only the instruction with no training data is provided to the LLM [2]. Contrary to zero-shot prompting, some training examples are given in few-shot prompting to guide the model in the output creation [8]. In chain-of-thought prompting, the LLM is asked to perform a task step-by-step. This should help the LLM to achieve better results, as a complex task is split into multiple simpler tasks [76]. An advantage of prompt engineering is that there is no need for labeled data which is often not available [47]. In addition, no training of the LLM is required. On the other hand, the designed prompt must be precise so that GPT understands the instructions correctly. Otherwise, the output of the LLM will be of poor quality [71].

2.7.2. Fine-tuning

Fine-tuning is a technique to adapt LLMs to a specific task based on a dataset. All or some of the model parameters are updated via gradients from training samples. Another option is to add additional model parameters and train on them [47]. With fine-tuning, performance improvements can be achieved for a specific task [32]. Disadvantages include the risk of overfitting, non-robust learning on small datasets, and the possibility of catastrophic forgetting. That means, the LLM forgets things it was able to do before the fine-tuning [47]. Another drawback of fine-tuning is that the efficiency depends on the size of the training data [32].

2.8. Metrics

In this thesis, the metrics precision, recall, and F1-score are used. For the calculation of these metrics, different measures are necessary. They are calculated based on the measures of true positives (TP), false positives (FP), and false negatives (FN).

Precision Precision measures the proportion of entries that are correctly classified as positive out of the total number of entries predicted as positive.

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

Recall Recall measures the share of the correctly positive predicted values compared to all correctly classified results, the positives and the negatives.

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

F1-score The F1-score is the harmonic mean and a trade-off between precision and recall.

$$F1\text{-}Score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.3}$$

There are different additional metrics to calculate the average of the previous calculated values. In this thesis, the micro, macro, and weighted averages for precision, recall, and F1-score are calculated. For the micro average, the example of precision is given, but the formulas for recall and F1-score follow the same pattern. Macro and weighted average are kept abstract with a placeholder for the respective metric.

Micro Average The micro average adds up the true positives, false positives, and false negatives for each case and calculates the metric (c.f. Eq. (2.1), Eq. (2.2), Eq. (2.3)) as usual. It is preferably used when there is an imbalance between the classes in the dataset. As an example, the formula for precision is given. It sums the true positives and false positives of all classes and calculates the precision using Equation (2.1).

$$Precision_{\mathrm{micro}} = \frac{\sum TP_{\mathrm{i}}}{TP + FP} \tag{2.4}$$

Macro Average The macro average sums the metric of all classes and divides them by the number of classes n. It is used if all classes are to be weighted equally.

$$Metric_{\mathrm{macro}} = \frac{\sum Metric_{\mathrm{i}}}{n} \tag{2.5}$$

Weighted Average The weighted average takes into account the possibly unequal sizes of evaluation sets.

$$Metric_{\mathrm{weighted}} = \sum Metric_{\mathrm{i}} \cdot \frac{EvalSize}{TotalSize} \tag{2.6}$$

3. Related Work

This chapter gives an overview of the related work in the categories of requirements elicitation as well as information extraction. First, some works are summarized that describe current techniques for requirements elicitation such as interviews or questionnaires. For the classification task of this thesis, information needs to be extracted from the requirements. Therefore, several works that evaluate information extraction tasks and their approaches are regarded as well, in particular, approaches using machine-learning techniques and LLMs.

3.1. Elicitation of Requirements

Requirements elicitation involves the identification of stakeholders, their needs and requirements (c.f. Section 2.1). Different techniques, such as interviews, questionnaires, but also misuse cases, and security use cases can be used. Many works investigate the elicitation of requirements [15, 74, 77]. In particular, Yousuf and Asger [74] give a good summary of the existing techniques such as document analysis, questionnaires, interviews, as well as observation, and group techniques like brainstorming or prototyping. They point out that using multiple techniques is important to gain good coverage for the requirements. The authors list the respective advantages and disadvantages of the techniques so that the disadvantages of one technique can be neutralized by using a second technique for requirements elicitation. However, they do not provide a concrete list of good combinations for requirements elicitation.

Furthermore, Zowghi and Coulin [77] describe the requirements elicitation process. This includes, among others, developing an understanding of the domain and selecting the appropriate elicitation methods. They list different techniques for requirements elicitation, including interviews, questionnaires, and domain analysis, as well as group work, brainstorming, observation, and prototyping. They briefly describe each of the techniques and explain when it is best to use them. Based on their practical experience and their evaluation of the literature, the authors provide information on which techniques should be used additionally or alternatively. Domain analyses, for example, can be carried out in addition to interviews, while group work is an alternative to interviews. The authors present 20 techniques, but select only eight of them to indicate whether to use them in addition to each other or alternatively. Like Yousuf and Asger [74], Zowghi and Coulin come to the conclusion, that the use of several different techniques for requirements elicitation is important for a good result due to the different characteristics of the techniques.

There also exist elicitation techniques that are developed specifically for security requirements elicitation. Sindre and Opdahl [63] developed misuse cases. Misuse cases describe undesirable behavior and specify interaction sequences that prevent the successful execution of a use case. They can be presented as a diagram or in text form. The textual representation of a misuse case specifies, among others, a security threat, the corresponding interaction sequences, assumptions, and potential misusers. Sindre and Opdahl define the term of a misuser as someone who launches a misuse case. An example of a misuser and a misuse case is an outside fraudster who aims at stealing credit card information from an e-store. They describe five steps that are executed iteratively to elicit security requirements with misuse cases. First, critical assets have to be identified. Second, security aims must be defined for each critical asset. Third, potential security threats to the goals have to be identified. Fourth, the risks posed by the potential threats must be identified and analyzed. Finally, in the fifth step, the security requirements are formulated, if possible with the help of a taxonomy. In their discussion, Sindre and Opdahl come to the conclusion that misuse cases help to deal with security early on in the development process. As misuse cases can be easily understood, they can help in communicating with the stakeholders. However, Sindre and Opdahl also admit that misuse cases are not equally suitable for all threats. Furthermore, it is not always possible to identify a misuser or a sequence of actions. Even though misuse cases seem to be helpful in general, they differ too much from usual security requirements as misuse cases aim to identify security threats and not requirements. Moreover, for the security requirements in this thesis, natural language is used, so diagrams are unsuitable.

Firesmith [23] extends Sindre and Opdahl's approach with security use cases. Security use cases specify security requirements. They are designed as tables containing fields for the security threat, user and misuser interactions, as well as system requirements, interactions, and actions. Furthermore, pre- and postconditions are also specified in the security use case. Misuse cases drive security use cases. In addition, Firesmith discusses the differences between misuse cases and security use cases, e.g., the former is used by the security team and the latter is used by the requirements team. He argues that misuse cases can help to identify security threats but they do not directly contain security requirements. Firesmith offers exemplary security use cases for access control, integrity, and privacy that have been kept abstract for good reusability. Then, he provides guidelines for developing own security use cases in case the exemplary security use cases do not fit. In contrast to misuse cases, security use cases include system requirements for security. Security use cases are a special subtype of how requirements can be expressed. Accordingly, this is not always 100% transferable and also restricts the format. It is therefore not used in this thesis.

3.2. Information Extraction

Information extraction (IE) is concerned with extracting interesting information from unstructured plain text. Recognizing and extracting phrases from natural language texts has been actively researched in recent decades. Various approaches have been tested, recently, large language models have also been used for information extraction. In 2012, Mendes et al. [49] developed *DBpedia Spotlight*, a system for concept tagging that can recognize different types of phrases. Mendes et al. test different approaches for recognizing the phrases, e.g. a simple string comparison with the lexicon database where each phrase in the text that has an entry in the database is recognized. This approach generates a lot of false positives because of ambiguous words, e.g., the function word *up* also gets a tag, because of the movie *Up* from Pixar. Next, Mendes et al. try different combinations for noun phrase recognition and finish by combining named entity recognition (NER) with noun phrases. They test their various phrase extraction techniques with different models, e.g., from the OpenNLP project and the LingPipe Exact Dictionary-Based Chunker, and find out that the NER with noun phrase recognition has the best recall with 68.3% but also generates a lot of false positives. The recall is especially relevant in this case as subsequent steps are based on the input from phrase extraction but precision is also relevant. Although the authors classify different elements, this thesis attempts to achieve better evaluation results by using a different approach with the utilization of a large language model.

Sainz et al. [61] pursue a different approach for information extraction by fine-tuning the Code-Llama model with annotation guidelines that were written by experts. The resulting model *GoLLIE (Guideline-following Large Language Model for IE)* performs also well on unseen IE tasks. Sainz et al. use a code-based Python representation to specify the input and output of the model. In their approach, labels, e.g., *person*, are represented as Python classes, and the corresponding docstring contains the label definition. Python comments include the possible annotation candidates. The model without the guidelines is taken as a baseline to compare against. To build a robust model that is more generalizable, noise is introduced during training. Furthermore, only datasets of the two domains, *news* and *biomedical*, are used for training. The model is trained on five different IE tasks, among others named entity recognition and relation extraction. The evaluation is performed on the training datasets whereby datasets with new domains such as *science*, and *wikipedia* are also added. Sainz et al. find out that the baseline and *GoLLIE* have similar results. This is due to the implicit learning of the label definitions by the provided data. *GoLLIE* performs significantly better than other state-of-the-art methods, on average, the state-of-the-art methods achieve an F1-score of 42,6% whereas *GoLLIE* obtains 60%. The performance on unseen labels decreases slightly, but not as much as other works. The error analysis shows that there are conflicts between fine and coarse-grained label categories. Moreover, it is difficult if the annotations in the used datasets do not match the guideline definitions. The authors conclude that guidelines with clear definitions are essential for receiving good results. In this thesis, fine-tuning is also applied but on a different model, namely on GPT 3.5 turbo. The input structure for this model is usually a text in natural language and not a code representation. Therefore, this thesis will follow a different approach concerning the input and output representations.

Another approach by Han et al. [30] evaluate ChatGPT on 14 different IE sub-tasks to evaluate how well ChatGPT extracts information. Among others, the IE tasks event detection as well as flat and nested entity recognition are tested on 17 different datasets. Flat entity recognition identifies each entity as a separate entity whereas in nested entity recognition, an entity can include other sub-entities. The authors do not clearly mention which version of ChatGPT they are using. For each task, they provide at least four datasets. Han et al. perform zero-shot, few-shot, and chain-of-thought prompting and evaluate ChatGPT's performance, robustness, and error types. They handcraft five different prompts for the zero-shot scenario and reuse the best prompt for the few-shot and chain-of-thought scenarios. In the few-shot scenario, they provide five examples along with the prompt. The explanation prompt for the chain-of-thought scenario is created with the help of ChatGPT and added to the best zero-shot prompt. The evaluation results show that ChatGPTs performance is far below the state-of-the-art and that the chain-of-thought scenario is not much better than zero-shot prompting. This is illustrated by the F1-score for the flat NER scenario, which is 94.6% for state-of-the-art models, while ChatGPT only reaches 65.13% for the common CoNLL03 dataset. Han et al. discover that ChatGPT outputs longer phrases than what was annotated. Therefore, they propose a softmatching algorithm that calculates the similarity of the head and tail of a phrase, that must be above a certain threshold. This means the phrase no longer has to completely match the annotated one. The new evaluation results show an improvement but are still behind the state-of-the-art. This approach is questionable and it is unclear how well the results can be relied upon as the metric was adjusted after the evaluation. Therefore, in this thesis, a different approach is used.

Zhang et al. [75] also perform evaluations on GPT 3.5 turbo with zero-shot prompting to retrieve requirement related information out of different datasets. Contrary to Han et al., Zhang et al. focus on requirements retrieval tasks and therefore choose two general and two specialized datasets for the evaluation. The general datasets include app reviews and app descriptions whereas the specialized datasets contain user stories, and non-functional software requirements specifications (SRS). The authors evaluate different tasks such as multi-class and multi-label classification, as well as term and feature extraction. In a multi-label classification, several labels can be correct for one instance at the same time, whereas with multi-class only one of several classes can be correct. The prompts are handcrafted by multiple persons. Then, GPT results are parsed, and compared against the baselines from the studies of the selected datasets. These show that the GPT results have a higher recall and lower precision than the baseline. Moreover, the specialized datasets obtain better results than the general ones because of less noise in the dataset. The authors conclude that GPT shows promising potential in recognizing and classifying requirements. Contrary to the first three works that work with natural language text, Zhang et al. consider requirements, even non-functional requirements, but not only those that are security-related. This thesis extracts different elements from requirements, while all the previously mentioned works extract, among others, named entities, events, or requirements information.

4. EVerest Software Architecture Derivation

EVerest provides a framework for electric vehicle charging stations [19]. It is used as a case study in this thesis. To get a deeper understanding of EVerest, it is described with its different modules and functionalities. EVerest cannot be assigned to a clear architectural scheme. It includes various aspects of different schemes, e.g., a partially layered architecture and also microservice approaches. The modules are mainly arranged in layers and communicate via MQTT, a publish and subscribe pattern [52]. In terms of deployment, there is one manager process that orchestrates the other processes and their execution. The EVerest implementation adheres to the standards OCPP and ISO 15118. OCPP is a manufacturer-independent open charge point protocol. ISO 15118 contains protocols on multiple layers, indicated by dashes, which specify the bidirectional communication between vehicle and grid. *EVerest* comprises several repositories, including *everest-core*, which contains all available modules for charging scenarios. EVerest's core can be divided into different layers. These layers are, among others, the hardware layer, the protocol layer for communication between the vehicle and the charging station, the authentication layer, and the application layer with OCPP. Other layers include the `EvseManager` that controls a charge point, and the energy management.

4.1. Architecture and Components

A simplified view of EVerest's architecture including its modules can be seen in Figure 4.1. The hardware layer consists of several drivers for hardware. `DPM1000PowerSupply` provides a power supply for DC charging, including, among others, limits for the power, current, and voltage. The `YetiDriver` offers a power meter that measures the energy, voltage, current, and power, as well as board support for AC charging. The layer also includes other hardware modules such as `MicroMegaWatt`, a charger for both AC and DC charging from PIONIX, and other power meter modules. The `SerialCommHub` enables the communication with attached serial devices. It can be used, e.g., by the `GenericPowermeter` for serial power meter hardware. These drivers are used by the `EvseManager` to be able to charge a vehicle and measure the amount of charged energy.

Furthermore, the `EvseManager` uses the protocols of the protocol layer that provides multiple charging protocols. `EvseV2G` and `PyJosev` implement the charging protocols DIN SPEC 70121 and ISO 15118-2, both of which support DC charging. In addition, ISO

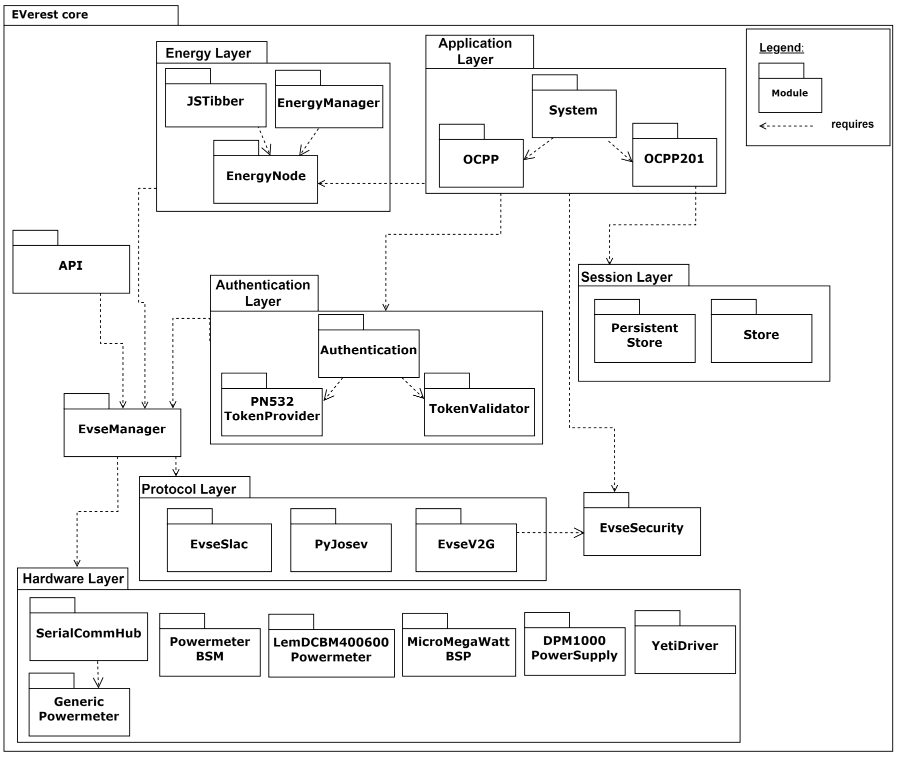


Figure 4.1.: Simplified view of the layers of EVerest with their modules

15118-2 offers AC charging as well. The module `EvseSlac` provides an implementation of ISO 15118-3, which is responsible for the data link negotiation. It works with signal level attenuation characterization (SLAC) and ensures the physical connection between the electric vehicle (EV) and the charging station. The EV sends an ethernet broadcast message, and each receiving electric vehicle supply equipment (EVSE) calculates the signal strength and sends it back to the EV. The charging station with the highest signal strength is identified as the connected EVSE [12]. The EV joins its logical network, and encrypted information on the higher layers can now be exchanged [26]. The communication in ISO 15118 can be secured with TLS where the charging station and the electric vehicle exchange stored certificates. These are stored in the `EvseSecurity` module.

A main component of EVerest is the `EvseManager` that controls an EVSE. It has a connector ID and can manage multiple physical chargers if they are not used simultaneously [1]. The EvseManager needs a board support module for AC charging, whereas the power supply for DC charging is optional. Furthermore, the `EvseManager` controls the charging process and keeps track of its state and how much energy has been charged so far. Modules in the upper layers call the `EvseManager`, e.g., the authentication module.

The authentication layer contains the `TokenProvider` and `TokenValidator` as well as the `Authentication` module. Both `TokenProvider` and `TokenValidator` are required for the `Authentication` module to function. It receives the token from the `TokenProvider` and sends it to the `TokenValidator` that validates it. An example of a token provider is the `PN532TokenProvider` module that reads an RFID or NFC card and publishes the token via

MQTT. The `TokenValidator`, e.g., an electric mobility service provider where the RFID card is registered, can check if the token is valid. If yes, and one connector is available or already plugged in, the `Authentication` module assigns the token to the `EvseManager` of that connector and calls it to start the transaction. The `Authentication` module also stops the transaction, e.g., if an RFID card is swiped for the second time. Furthermore, it manages the reservations of connectors. The `PersistentStore` and `Store` modules are located in the session layer. They provide key-value stores for OCPP 2.0.1.

The `API` lies on top of the authentication and session layers. It provides an interface that is currently not used in the *everest-core*. The module regularly publishes variables for each connected `EvseManager`, e.g., for the current charging session. Optional, OCPP variables can be published if an `OCPP` module is provided to the `API`. To obtain these values, the `API` subscribes to variables of the `EvseManager` and the `OCPP` module.

The energy management consists of an `EnergyNode`, an `EnergyManager`, and the `JSTibber` module. An `EnergyNode` provides an energy grid and requires a module that consumes the energy. It represents a current fuse. For the energy prices, it subscribes to the `JSTibber` module that retrieves data from the Tibber Price Energy Forecast API. The `EnergyManager` offers load balancing and optimization for more sophisticated energy management when using multiple `EvseManagers`. Currently, its interface is not yet in use in the *everest-core*.

The application layer contains `OCPP` modules which support two different OCPP versions, as well as the `System` module. OCPP is responsible for the backend communication between the charging station and the charging station management system [72]. Information, e.g., about the charge point state or firmware updates is transmitted. There are two versions implemented in EVerest, version 1.6 and 2.0.1. The latter is still under development. In contrast to OCPP 2.0.1, OCPP 1.6 supports several formats. OCPP 2.0.1 supports ISO 15118 Plug & Charge. It was also improved concerning the security and smart charging support. Both versions can be used as token providers and validators in EVerest. They both require the `System` module that is responsible for logging updates, firmware updates, and setting the system time. OCPP also requires the `EvseSecurity` module.

4.2. EVerest Architecture Model

The architecture of EVerest was derived to get a deeper understanding of the system and to prepare for the creation of trace links between the requirements and the architecture. All EVerest modules that are involved in the charging process were derived. For the creation of an architecture model, EVerest's documentation and code were studied. The documentation of EVerest is available online [70]. Furthermore, videos on YouTube helped to gain a deeper understanding of EVerest [17]. The information obtained was checked for validity as mentioned by Yousuf and Asger [74]. For this purpose, an employee of PIONIX was asked about the actuality of the data. PCM is chosen for the representation of the architecture. The component model was created based on the state of the GitHub repository of the 4th of December 2023.[1] In EVerest, each module has a `manifest.yaml` file where the provided and required interfaces are specified. The connections of the modules to interfaces were derived from these files. A module in EVerest corresponds to a component in PCM. Therefore, the term component is used in the following. The component model for EVerest was created in the Palladio Bench. To structure the component model, it was organized in the layers described in Section 4.1. The rules that were applied to the creation of the component model are:

1. Remove simulation components, as only the production environment is relevant.
2. Remove unused interfaces and components.
3. Remove components that are only used for setting up EVerest.
4. Remove examples, debugging, and deprecated components.
5. Remove dummy components and rename those that are the only ones that implement a used interface. These only exist to simplify the creation of own components.
6. Add explaining names to drivers for hardware boards.

Since EVerest only provides a framework that is designed for further customization, there are individual components, such as the `API` and the `EnergyManager`, that are not connected to others. In addition, there are connections between components and interfaces that are not shown in the complete component diagram, as the components only optionally require these interfaces. This relationship cannot be mapped in Palladio. The connections for which an interface is optionally required by only one component are shown, whereas the others are deleted, starting at the lower layers. OCPP versions 1.6 and 2.0.1 both optionally require the `OCPPDataTransfer` interface. That relationship is displayed in both cases as it would be confusing if only one component needs that interface and the other does not. The complete component model contains 24 components with 20 interfaces (c.f. Appendix A.4). For comprehensibility, some labels and unused properties are hidden.

[1]https://github.com/EVerest/everest-core/commit/5be1ef7d24f5eff6230ce5d8fc3078ae2b175def

5. Security Requirements Elicitation

Security requirements of the open-source software EVerest are elicited to answer the first research question of this thesis. To ensure a good coverage of the requirements and collect both coarse-grained and fine-grained requirements, different methods are chosen [74, 77]. As stated by Yousuf and Asger [74], the analysis of documents is a good start for requirements elicitation to gain a better understanding of the software project before meeting stakeholders. Therefore, the documentation of EVerest was studied before the requirements elicitation. The EVerest architecture modell has emerged from the study of the domain (c.f. Chapter 4). The elicitation process is split into two parts. At the beginning, requirements are collected by conducting a pilot-tested questionnaire where the granularity level is not predefined. The questionnaire is sent to stakeholders of EVerest, e.g., companies that work with EVerest or software developers of EVerest, via a mailinglist. More fine-grained requirements are necessary for the classification task addressed in the second research question. Therefore, in a second step, the usually coarse-grained requirements from the questionnaire are further refined in interviews with software developers of EVerest. This chapter first describes several elicitation techniques with their advantages and disadvantages in Section 5.1. Then, the design of the questionnaire developed in this thesis is presented in Section 5.2. Section 5.3 contains the pilot study of the questionnaire and resulting adaptions. In Section 5.4, the conduction of the questionnaire and an overview of its results are provided. After that, Section 5.5 describes the interview design and Section 5.6 provides the pilot study of the interviews. In Section 5.7, the conduction of the interviews and the resulting dataset are described. Finally, the threats to validity for this chapter are discussed in Section 5.8.

5.1. Elicitation Techniques

Common techniques for requirements elicitation are questionnaires and interviews as well as group works and observation [77]. Questionnaires are a cost-effective way to get responses from multiple stakeholders [74]. Moreover, a questionnaire can be conducted anonymously, which can lead to more open and honest answers [48]. However, a limitation to this method may be a low response rate [48, 68]. In addition, there is no option for further inquiry if there are any misunderstandings or ambiguities regarding the wording of the questions. The disadvantages of questionnaires correspond to the advantages of interviews. Further clarification is possible and depending on the type of interview, new topics may be discussed that were not previously known to the interviewer. Interviews are time-consuming. Due to time and cost limitations, it is usually only feasible to collect information from a small number of people [74]. Instead of interviews, group work can be carried out [77]. Group work is a moderated collaborative meeting where stakeholder needs are considered and requirements are discussed. For this technique to work, all participants must feel comfortable talking openly. It is less time-consuming than conducting interviews with the same amount of people, but it is hard to find willing participants and a common appointment [74]. Another technique for requirements elicitation is observation. It is a method without direct interaction where the analyst observes the user as he performs current processes [77]. This is a time-consuming approach, and users sometimes change their behavior if they know they are being watched [74]. An advantage of observation is that the analyst gets an impression of how the user behaves in the target domain and how he performs current activities [77]. Observation cannot be realized in the context of this work. For example, an analyst would have to observe a hacker attacking an electric charging station in order to derive adequate security requirements. Group work is suitable, but not feasible due to working and thus busy stakeholders. It is highly unlikely that a joint appointment could be found as different stakeholders have regular meetings that could overlap. In this thesis, a two-step approach is chosen to elicit security requirements of EVerest. First, a questionnaire elicits requirements from stakeholders of EVerest on different granularity levels. This is not too time-consuming for the stakeholders and provides initial requirements from multiple participants. The granularity level of the requirements cannot be ensured in questionnaires. Therefore, individual interviews are conducted with developers of EVerest to further refine the usually coarse-grained requirements. The goal of this two-step process is to get detailed security requirements on the design level that can be mapped to SecLan's system model elements.

5.2. Questionnaire Design

An existing validated questionnaire should preferably be used [48, 68]. This saves time and resources for the researcher as she does not have to develop a questionnaire on her own [7, 48]. However, for the elicitation of security requirements, no appropriate questionnaire could be found. Only a questionnaire could be retrieved that was used to evaluate a technique for identifying and prioritizing requirements [45]. It has different objectives, involves a different software project and was thereby not transferable. Therefore, a new questionnaire was developed.

When designing a questionnaire, several guidelines should be followed regarding the question length, their wording, the question type, the order of questions, the questionnaire length, and the presentation [48]. Boynton and Greenhalgh [7] state as a guideline that questions should be short with a maximum of 12 words, so that it is easier for the participants to understand the questions. Concerning the wording of questions, Oppenheim [55] provides guidelines, e.g., to ask no double-barrelled or double-negative questions and use simple words. There are two types of questions, closed and open questions (c.f. Section 2.1.1). Closed questions are easier and faster to analyze [68]. However, the format of open questions should be preferred if the aim is to get a deeper and qualitatively higher level of information. Stakeholders can answer in their own words and might give responses that the questionnaire designer has not considered [68]. The presentation of the questionnaire should stay clear and not appear too overloaded in order to offer the participants a good presentation. Contact information should be provided in the questionnaire and a possibility to get access to the results of the study [48]. Furthermore, a consistent format of the questionnaire helps the participants to work through it [68]. There are also recommendations to increase the response rate of the questionnaire. Oppenheim [55] suggests using logos of sponsors. Moreover, Marshall [48] advises starting with simple questions as well as keeping the questionnaire short enough so that many stakeholders participate. Walker [68] states that a questionnaire should not take more than 30 minutes to complete. These guidelines were kept in mind when designing the questionnaire.

The questionnaire aims to elicit security requirements so that as many security categories with different objectives as possible are covered. Typical categories are confidentiality, integrity, and availability [13]. In addition, Firesmith [22] lists 12 different kinds of security requirements, e.g., integrity, authentication, and privacy. For each category, he gives examples as well as guidelines on how to specify the requirements. In this thesis, the categories should cover as much as possible, and at the same time be few, as the space in the questionnaire is limited. Miller [50] names several non-functional requirement categories and provides aspects to each category. The security-related categories are access security, integrity, and availability. After an investigation of the security categories and aspects, access security was split into the more well-known security categories of confidentiality and authentication & access control. Finally, the four security categories confidentiality, integrity, availability, and authentication & access control were chosen as they cover the range of security requirements fairly well.

The initial questionnaire is described in the following. It consists of seven sections, comprising an introduction, a general part, the four security categories availability, integrity, confidentiality, and authentication & access control, and a section for other security requirements. The welcome screen of the questionnaire mentions its aim, the elicitation of security requirements for EVerest. Furthermore, contact information as well as the duration of the questionnaire are provided. The author of this thesis tested it out and found that filling out the questionnaire takes 10 to 15 minutes if security requirements for all categories are written down. As stated above, logos on the front page can be helpful regarding the response rate. In the case of this questionnaire, there are no sponsors, but the logos of the university conducting the questionnaire, the KIT, and the project to which the questionnaire relates, EVerest, are added to the front page of the questionnaire, also for reasons of clearness for the participants. A note on the anonymity of the questionnaire is placed on the welcome screen as no personal data is needed for the elicitation of security requirements.

The next section first explains the structure of the questionnaire. Then, the participant is encouraged to write down as many requirements as possible. It is explained that new text fields are dynamically displayed when a text field is filled in. Up to 10 text fields can appear. The section provides a definition of the term “requirement” and an exemplary security requirement [22]. Furthermore, characteristics of requirements and the resulting writing recommendations for good requirements are given [38]. The participants are asked whether they have read the text above and they have to click on the corresponding button to continue the survey. If not pushed, they will be reminded once but have the option to skip this answer.

In the next section, general information about the participants is elicited. The first two questions are multiple-choice questions about the interest in EVerest and the profession of the participant. The participants can select whether they are developers at PIONIX, the company that founded EVerest, whether they are from a company that uses EVerest, or wants to use it. The professions to choose from include software architect, software developer, requirements engineer, and project manager. For both questions, there is also the option "other" if none of the provided answers apply. Participants can then enter their individual answers in a text field. In the third question, participants are asked about their confidence in the domain of software security using a 5-point Likert scale. These three questions collect criteria to describe the participants.

After the general questions, the main part begins. Four security categories are selected to cover the range of security requirements while keeping the questionnaire short. They are presented to the participants in separate sections. As discussed above, the categories confidentiality, integrity, availability, and authentication & access control were chosen. In each section, a short introduction of each security category is given to avoid misunderstandings. The definitions are taken from ISO 27000 [37]. Then, several aspects are listed to help the participants to come up with security requirements in EVerest for the respective category. Since the mentioned aspects are not exhaustive, three dots are added at the end of each list. The aspects are taken from Miller [50]. She lists aspects for different non-functional requirements, among others, for the security categories access control, integrity, and availability. The aspects describe important parts of the respective security category and should therefore be considered when eliciting requirements for the respective category. The aspects for confidentiality and authentication & access control are taken from the category access security. Encryption is part of the category confidentiality, therefore, the aspect of data encryption methods is added. After the introduction of each category, the participants are asked for respective requirements regarding EVerest. They can write up to 10 requirements for each category. The answers are all optional, as there may be participants who cannot think of a requirement in one security category. This avoids them having to come up with something. In each of the categories where requirements could be entered, there is a reminder for the participant that additional fields appear dynamically, but not all of them have to be filled. The order in which the categories are displayed is random to prevent the last category from receiving fewer security requirements.

The last section of the questionnaire gives the participant the opportunity to write down requirements that did not fit into any of the previous categories. At the end of the questionnaire, the participant is thanked and an e-mail address is provided in case the participant would like to receive the results. The questionnaire was created on the platform LimeSurvey. It can be found in Appendix A.1.

5.3. Pilot Study Questionnaire

A pilot study is a small preliminary study that is carried out before the actual study. It should reveal problems of the initial questionnaire (c.f. Section 5.2), e.g., unclear instructions, or inappropriate questions. The pilot study ensures the reliability and validity of the questionnaire [48]. Reliability means, that the questionnaire measures consistently the same responses from the participants. If they would fill in the questionnaire a second time, the responses should stay the same [6]. A questionnaire is valid if it measures what it intends [48]. It is best to test the questionnaire on the same group as the target group [68]. The results of the pilot study should not be used for the later study [3]. The target group of the questionnaire is relatively small, which is why, Ph.D. students who are familiar with the requirements engineering and security were asked to participate in the pilot study to achieve the highest possible response rate for the later questionnaire. Therefore, an overview of EVerest with an exemplary charging scenario and its corresponding component diagram were provided to the participants as they do not know the software. That should provide them with a certain amount of background knowledge to be able to respond to the questionnaire and come up with security requirements for EVerest.

At the beginning of the pilot study, there is a short introduction as well as an explanation of the procedure in five steps. The participant has to read the instructions and the scenario and look at the corresponding component diagram. The duration of the questionnaire is measured to ensure that it conforms to the indicated time [3, 48]. Therefore, the participant starts a timer when starting the initial questionnaire and stops the timer when the questionnaire is completed. After the explanation of the pilot study procedure, EVerest is presented in general terms, followed by a description of its different layers. Then, a charging scenario of EVerest is provided as well as the corresponding component diagram. In the scenario, the different components of EVerest are described together with their functionalities. Explanations about the authentication process and how to conduct a charging session are provided as well. The scenario was mostly taken from EVerest's documentation website [1]. The simulation components were removed from the scenario as security requirements should not be elicited for the simulated part but for a production setting. The names of some components were also changed to have more expressive names, e.g., *token_provider* was renamed in *RfidReader* and *token_validator* in *electricMobilityServiceProvider*. After reading this information, the participant is asked to answer the questionnaire. Then, he has to fill in a feedback form and report his findings. He is questioned to fill in his duration of the questionnaire. After that, he is asked if the instructions in the questionnaire were comprehensible and clear. The options to select are "yes", "no", and "partially". If selected "no" or "partially", the participant should specify what was unclear. After that, the participant is invited to make suggestions for improvement. At the end, the participant is thanked for her time and feedback. The pilot study can be found in Appendix A.2.

Two Ph.D. students participated in the pilot study. Both participants needed 18 minutes to fill out the questionnaire. From the point of view of the author of this thesis, 10 to 15 minutes are appropriate for the later questionnaire, as the test persons needed extra time being not familiar with EVerest. They stated that the given instructions and questions were partially clear. One participant said that confidentiality and access control could be mixed up easily. They suggested introducing the categories at the beginning or at least making clear to which category access control belongs. This recommendation was implemented in the later questionnaire. The other participant commented that the desired level of detail of the requirements was hard to grasp despite the one exemplary requirement in the introduction. The participant found the aspects of each security category not helpful either for writing requirements. The participant commented that especially for a person who is not a security expert, it would be hard to go through all requirement types and aspects, and the person might skip these questions. The participant suggested deleting the dots at the end of the aspects because it would introduce uncertainty. This was not considered necessary. Both participants requested exemplary requirements for each security category. One participant stated that the examples should capture the aim of the category. The other participant would prefer examples in different granularities, and suggested deployment requirements for availability or integrity. Therefore, one example for each security category was added to each section, with two examples being more abstract and two being more detailed. Both participants also provided feedback on the pilot study itself. One participant said that the introduction of EVerest was too short and detailed information was missing. Therefore, it was difficult to write down non-generic security requirements in the questionnaire. This is not considered as a problem for the later participants in the questionnaire as they are stakeholders of EVerest and know the software. Furthermore, the participant also noted that EVerest exists, meaning that some requirements might already be addressed. This is not a problem, as these requirements can be elicited together with the others. The other participant gave inline feedback in the PDF file, asking questions on what the charge point is, if the layers should be visible in the component diagram, and what the RFID token looks like. Moreover, he advised repeating the name of the physical connection protocol SLAC in the description of the scenario. As the scenario is no longer used, no further adjustments were made to it.

As a reaction to the results of the pilot study, the four security categories were listed in the introduction of the questionnaire. Access control was moved from the security category authentication & access control to confidentiality and added as an aspect of that category. Furthermore, one example for each security category was added. Two examples have a higher abstraction level, two examples are more specific. The two abstract exemplary requirements for integrity and authentication were taken from Firesmith [22]. Initially, no examples were given to prevent participants from simple adapting the exemplary requirement to EVerest. However, since both participants suggested that examples would be helpful, one exemplary requirement was inserted in each security category section of the questionnaire.

5.4. Questionnaire Conduction

It was planned to present the survey at one of the weekly meetings to inform stakeholders and encourage participation. Then, the link to the survey should be distributed via the mailing list, as it is assumed that all participants of the weekly also subscribed to the mailing list. It had about 250 subscribers at the 14th of December 2023. The subscribers are people who are interested in electric vehicles, but also companies that use or want to use the software and contribute to the project. The developers of PIONIX are part of the mailing list as well. Therefore, it is assumed that the people on the mailing list have suggestions regarding the security requirements of EVerest as they desire a secure data processing. The survey was open for one week. A reminder via the mailing list was planned to be sent two days after the first mail with the link to the questionnaire. It was arranged with PIONIX that the survey could be presented at the weekly community meeting of EVerest on Tuesday 12th of December 2023. At the meeting, the community's motivation for completing the questionnaire, the improvement of the security of EVerest, was expressed. It was stated that no prior in-depth knowledge is required and that the survey takes about 10 to 15 minutes if requirements for each category are written down. Furthermore, it was mentioned in the meeting, that if the participants do not have 10 minutes left, already one or two requirements would be helpful. Then, the link to the questionnaire was shared with the invitation to fill it out right after the meeting. The survey link was also sent via the EVerest mailing list to the community of EVerest after the meeting. Two days later, a reminder email was sent with the emphasis on filling out the questionnaire, even if only two requirements would be written down. The subscriber of the mailing list can decide for the intervals of the emails whether he wants to receive individual emails or a summary for each day. Since there is no knowledge of how many people have opted for the individual mail, the assumption has been made that many people want to receive a summary. Therefore, the exact time of the reminder is considered irrelevant. In the next weekly community meeting on Tuesday 19th of December 2023, there was a reminder to fill out the survey by the end of the day. The survey officially ran for one week from Tuesday 12th of December to 19th of December 2023. It was activated on Monday 11th of December and closed on Wednesday 20th of December to ensure that the answers in the late evening of Tuesday, the 19th of December were collected, especially because some stakeholders of EVerest are located in the US.

Although 40 people participated in the questionnaire, only six completed it. One other participant wrote security requirements but forgot to submit the questionnaire. This was not a problem as his answers were still saved. The rest of the participants did not answer any questions or filled out only the general part. Therefore, only the responses of the seven people who wrote security requirements were considered in the following. A total of 67 requirements were collected, comprising 15 on availability, 11 on integrity, 20 on confidentiality, 16 on authentication, as well as five others. Three PIONIX employees and four people from companies that use EVerest participated. Five participants stated that they work both as a software developer and as a software architect. One person answered that he works as a software architect, and one project manager also participated in the questionnaire. The participants' confidence concerning software security ranged from three to five with an average of four.

To clean up the participant's answers to the questionnaire, the following rules were applied:

1. Ignore everything not security-related or too general responses, especially functional requirements
2. Resolve co-references
3. Remove sentences not written as a requirement
4. Rearrange access control requirements from the category authentication to confidentiality. The question asking for authentication requirements inadvertently included access control as well. Therefore, some participants misclassified the requirements, although access control was explicitly listed as an aspect of confidentiality.
5. Remove answers that refer to examples from the questionnaire, noting that it is not relevant for EVerest

An exemplary requirement that was not considered security-relevant and therefore has been removed in this process due to the first rule was "Everest shall not have any failed session when try to charge a vehicle that is working according to the specifications of the charging protocols". The requirement "It should be tested with fuzzed input" was adapted to "EVerest should be tested with fuzzed input" by resolving the co-reference using the second rule. The third rule was applied to requirement "We should work on storing all data in a centralized place to simplify backups in the final product. Right now there are many files and folders that contain databases for persistent storage" that has been removed too. A requirement that was deleted due to the fifth rule is "Authentication for management web interfaces etc is usually out of scope for EVerest as it is provided by the OS around it". After this process, 57 requirements were left, two of them are multi-labeled concerning the security objective. The requirements need to be further specified so that they can be mapped to SecLan's system model elements and that an automatic verification is possible in the future. Therefore, interviews were conducted with the developers of EVerest.

5.5. Interview Design

For a further specification of the elicited security requirements from the questionnaire to the design level, interviews were performed. They are one of the most common and widely used techniques for requirements elicitation [15, 36, 77]. There are three types of interviews, structured, unstructured, and semi-structured interviews (c.f. Section 2.1.2). For this work, semi-structured interviews were selected. They serve best for the purpose of this thesis as they allow further inquiries, but the risk of focussing too much on one area is low. Structured interviews are not recommended for qualitative research because they follow a strict protocol [66]. Unstructured interviews are not appropriate either, especially since the requirements to be specified already introduce a certain structure.

Before conducting an interview, the interviewer has to be well prepared to get useful results. Therefore, an interview protocol was designed [66]. Regarding its content, structure, question order, and wording, several guidelines have to be observed. The protocol should contain an introduction and ending as well as questions with optional probes [39]. At the beginning, the interviewer should briefly introduce herself and her research topic [60]. Moreover, Bolderston [5] points out the importance of adhering to ethical rules such as consent, privacy, and confidentiality. This includes reminding the interviewee of his right to withdraw from the interview at any time [5]. Then, the interviewer should start with simple, open-ended questions so that the participant feels comfortable [42]. Bolderston mentions three categories of questions, the main, the planned, and the spontaneous follow-up questions. The main questions deal with the research aim while the second category, also known as probes, is helpful to get more detailed answers. Spontaneous follow-up questions can be used after listening to the participant to ask for further clarification of his response [5]. Prompts to get the interviewer talking are silence and repeating the question [60]. In general, the questions should be clearly expressed, understandable, and relatively short [5, 66]. Rowley [60] emphasizes the correct wording of the questions so that they are not formulated too vaguely or contain implicit assumptions. At the end, the interviewer should ask the participant if there is anything else he wants to add [5, 66]. Finally, the interviewer should thank the interviewee [5].

The interview is divided into three parts, an introduction with initial questions, followed by the specification of requirements in the main part, and finally, a short ending. The introduction includes a greeting and a short presentation of the interviewer and her research goals. The duration of the interview is also indicated. Then, the interviewee is reminded of the declaration of consent and his rights to withdraw from the interview at any time. After that, initial questions are asked whether the interviewee participated in the questionnaire (c.f. Section 5.4), and general information about the interviewee is collected. The interviewee is asked whether he works as a software developer and how long he already works for her company. Then, the interviewee is invited to list the components of EVerest he is working on and to show them in the previously shared component model. At the end of the introductory questions, he is asked how confident he is in the domain of software security on a Likert scale from one to five while five is the best.

Next, the task of the interviewee to specify requirements is explained and illustrated using two exemplary requirements and their possible specifications. For the purpose of requirements specification, a shared GoogleDocs document containing the two examples as well as the requirements to be specified is used. Furthermore, the interviewer states thought-provoking impulses for the refinement of the requirements. The interviewee is instructed to think aloud about possible specifications of the requirement. He could think about the deployment, components, people, or data involved and the information flow of data. Then, he should write down an English sentence below the requirement to specify it. The interviewer also explains the structure of the main part of the interview to the participant and provides the definitions of the four security categories availability, integrity, confidentiality, and authentication. These four categories are worked through one after the other. At the end of each category, there is the opportunity to formulate additional requirements for the respective category. After the last category, the participant could write down further security requirements that did not fit in the aforementioned categories.

In the main part, about 30 elicited security requirements taken from the questionnaire are discussed by working through the four security categories as well as the remaining category for other security-related requirements. For each requirement, the interviewee should write down a further refinement. At the end, the participant is asked if he is available for further inquiries and thanked for his time and knowledge. The final interview protocol with the adjustments of the pilotstudy can be found in Appendix A.3.

Two interview protocols with different requirements from the questionnaire were created. Each of the protocols comprised about 30 requirements written by three or four different authors. The requirements were separated along author boundaries. One protocol contained all the requirements written by PIONIX authors and the other contained those written by companies that use EVerest. The interviewees were asked beforehand, if they participated in the questionnaire. If yes, depending on their company, they received the protocol with the requirements of the other company. If they worked for PIONIX, they would receive the requirements written by companies that use EVerest, and vice versa. This prevents the potentially unpleasant situation of a participant having to specify his own requirements.

5.6. Pilot Study Interviews

Conducting pilot interviews is important to test the interview protocol. It ensures that the questions are understandable, that the question flow is appropriate, and that the research goals have been met [66]. Furthermore, practicing the interview is necessary to develop important interviewer skills such as asking appropriate questions and encouraging the participant to talk [33]. It is best if the participants of the pilot interviews are close to the target group so that the protocol can be refined and adapted to them [39]. There are also guidelines for the conduction of the interview. Concerning timing, the interview should not be too long, otherwise, finding willing participants will be difficult. The interviewer is responsible for adhering to the agreed time limit during the interview [39]. Therefore, it is helpful to schedule the meeting with enough time buffer in case of eventualities [60]. Furthermore, disruptions should be avoided [5].

Four Ph.D. students in the domain of software engineering were asked to participate in the pilot interviews. Although the data obtained from the pilot interviews will not be published, a consent form was sent out one day before the interview date for the sake of completeness. The interview protocols were shortened for the pilot interviews, so the participants only needed to specify five security requirements taken from multiple security categories. The interviews were conducted online via Microsoft Teams because the later interviews will also be conducted remotely. They were anticipated to take about 15 minutes. In the end, they lasted between 20 and 50 minutes because the participants were not familiar with EVerest, asked questions, and had to come up with ideas for the specification of the requirements. At the end of the pilot interviews, the participants were asked for feedback and whether they noticed anything negative. One participant stated that the font of the component model was small and there was a lot of free space between the components. It would be easier for the people if everything would be bigger and closer together. That was not feasible but to compensate, the component model was presented and briefly explained in the interview introduction to gain a rough overview of EVerest. Furthermore, the names of the components were unknown to the participants of the pilot interviews. That was not considered a severe problem, as the developers interviewed later were already familiar with the system. Another participant stated that the questions were formulated clearly, the atmosphere was comfortable and there was no pressure to answer. It was also noted that not all four security categories were present in the interview. That was due to the brevity of the pilot interviews.

As a result of the pilot interviews, organizational matters were adjusted and the order of questions was slightly changed to improve the transitions and make the interview smoother. Therefore, the explanation of the component model was moved after the second of the initial questions. Furthermore, the definitions of the security categories will be no longer all read out at the beginning but only at the start of the corresponding category. Moreover, the question of whether the interviewee participated in the questionnaire will be asked in advance as it is relevant for the selection of the interview protocol. The pilot interviews were also helpful in practicing how to respond to participants' spontaneous reactions. In addition, time was measured to approximate how much time the interviewee needs per requirement and what can be expected in the interview. The schedule for the later interviews was set up accordingly.

5.7. Interviews Conduction

When selecting the participants, it is important to consider who has the required information [60]. The goal of the interviews is to specify to which parts of EVerest's software architecture the requirement relates. Therefore, developers that know the EVerest components well were selected as participants. The contact person at PIONIX helped to select these developers and arranged three out of four appointments. Three PIONIX developers and one developer from Chargebyte, a company working with EVerest, agreed to an interview. Previously, they were asked via mail if they participated in the questionnaire to select the appropriate interview protocol. One of the PIONIX employees did participate in the questionnaire, so he received the interview protocol without PIONIX answers. The declaration of consent was sent one day before the interview and all interviewees returned it quickly. The EVerest component model depicted in Appendix A.4 was sent one hour before the interview as a common foundation and that the interviewee could open it on his own computer. All interviews were conducted in German and remotely, three via GoogleMeet and one via Skype. They lasted about an hour and a half and were not recorded. Technical problems in form of an unstable internet connection occured in three interviews, but these only affected the interview for a few seconds to minutes. In the first two interviews, the link to the Google Docs document was shared during the interview while in the last two interviews, it was sent out in advance. The first interviewee commented that the interview would have been shorter if he had received the requirements in advance. Therefore, this was tested with participant three and four. There were no major difference in interview duration. That could be due to the fact that the third and fourth interviewee only skimmed the received protocol before the interview. It is therefore not assumed that the slightly different experimental setup significantly influenced the results. The requirements to be specified were the same for the first and last interview as well as for the second and third interview. Only the order of the security categories was changed to allow a balanced specification of all requirements as the later requirements were usually addressed faster. At the beginning of the interviews, information about the interviewee was collected. All four interviewees were software developers. The PIONIX developers have been working there for two to three years. The Chargebyte developer has worked with EVerest for one and a half years. The level of confidence in the domain of software security was three for three of the four developers. One developer gave himself only an one in the security confidence level. The first two developers worked on various components, mainly in the higher levels, e.g., the OCPP and Auth components. One developer stated, he has been working a lot on the everest-framework, while the other focused more on different components as EvseSecurity or ISO 15118. The last two developers mainly worked on components of the lower levels such as ISO 15118, the EvseManager, Slac, or EvseV2G, Powermeter, and the board support interface. All interviewees took time to understand what the requirements mean and thought about how it works in EVerest. Then, they expressed their thoughts mostly aloud. The interviewer asked probing questions if the interviewee's specification of the requirement was not detailed enough, such as which components or parts of the software architecture were affected or involved. She also made sure she had understood things correctly. If the participant explained a lot and did not write everything important in the specified requirement, the interviewer asked whether the participant could add what he just mentioned to the requirement. Sometimes, the interviewee did not know e.g., which component should be responsible for a requirement or did not understand the requirement. In that case, a comment was made and then, the interview continued with the next requirement.

In the four interviews, 91 requirements were collected. The interviewees responses were slightly adapted for multiple reasons. When time was short in the interviews, the interviewee tended to write down his answer only in key words. These answers had to be formulated in a complete sentence ensure a consistent format. If it was obvious what the interviewee wanted to convey, the interviewer formulated the sentence in order to relieve the interviewee of some of the work. Otherwise, the interviewee was asked to formulate the sentence himself with respect to the requirements writing style mentioning what something *should* or *shall* do. Minor changes were made to some responses. The interviewees did not always follow the requirements writing style, also because EVerest already fulfills some of the security requirements. For some requirements, the writing style was adapted, for example, “Making sure that transactions are Eichrecht compatible, so that the power meter driver gets signed powermeter values at the appropriate times (eg. at the end and/or beginning of a transaction) and transmit this via OCPP” was changed to “Transactions should be Eichrecht compatible, so that the power meter driver gets signed powermeter values at the appropriate times (eg. at the end and/or beginning of a transaction) and transmit this via OCPP”. Moreover, multiple specifications of one original requirement were separated and it was made sure that the requirements specification could stand on its own and did not need the original requirement next to it. Furthermore, unknown abbreviations that are commonly used in the domain of electric vehicles were added to the requirements for clarification. Moreover, if it was not clear who was responsible for some activities mentioned in the requirements, the interviewees were asked to further specify them. For the creation of the dataset, tags were introduced for the handling of the responses. For some requirements, the interviewees also gave a reasoning. That is not part of the requirement itself. Therefore, parts of responses of two requirements were tagged as *rationale* and omitted in the dataset. 18 requirements were tagged as *not applicable to EVerest*. The interviewees stated that the fulfillment of the given requirement was not the responsability of EVerest or that this requirement was not correct for EVerest. Therefore, they did not specify the requirements and these were also omitted in the dataset. Six requirements were tagged as *not refineable* meaning that the interviewee did not come up with a more fine-grained specification of the requirement. These six were included unchanged in the dataset. If the interviewee could not come up with a specification of the requirement either because he was not sure what the requirement meant or because he was not confident in that part of EVerest, it was marked with *insufficient knowledge*. The 14 affected requirements were also omitted in the dataset. All changes were sent to the respective interviewee, so that he could confirm or correct them. At the end of each security category, the interviewee had the opportunity to write down additional requirements for that category. As a result, three requirements were added to the dataset. Some requirements referred to standards and specifications, such as the OCPP 1.6 security whitepaper and the OCPP standards. They were not described in more detail because that would exceed the time of the interviews. Moreover, the standards are openly accessible. At the end of this process, a total of 93 security requirements were retrieved, comprising 41 confidentiality, 14 integrity, 18 availability, 10 authentication, and 10 other security requirements.

5.8. Threats to Validity

The threats to validity for the topics of this chapter are discussed in the following. A distinction is made between internal, construct, and external validity. Internal validity refers to the extend to which a causal relationship can actually be shown. A small threat to internal validity is that the later participants of the questionnaire could have been annoyed by the second e-mail reminder. The impact of this threat is considered to be low as it is assumed that very annoyed e-mail recipients simply did not participate in the questionnaire. Then, it has no impact on the answers from the questionnaire but only on the number of participants in the questionnaire. Two of the four interviewees received the interview protocol with the requirements to be specified in advance. That could have influenced their answers in the interview. The interviewer asked the interviewees if they read the protocol in advance, to which one interviewee responded in the negative and the other said that he had only scrolled through it briefly. It is therefore assumed that no major bias has arisen.

Construct validity describes the quality of the measurement that is used to evaluate a concept. One risk to construct validity is that the interviewer explicitly indicated during the interview that the requirements should be at the design level. Thereby, the answers of the interviewees were controlled concerning the desired abstraction level. As a consequence, the specified requirements contain many trace links to EVerest's architecture. It is assumed that otherwise not so many explicit trace links would have been elicited. Another threat to construct validity is that the answers of the interviews were adapted afterward by the interviewer because of the sentence structure and style. An attempt was made to rewrite only the obvious requirements, e.g., form a sentence out of keywords. If the responses were more complex, the interviewee was asked to rewrite the sentence himself. Moreover, all changes were confirmed by the interviewees. Thereby, that threat is weakened.

External validity is concerned with the generalizability of the results. The small number of interviewees poses a threat to external validity. To mitigate this threat, all requirements are specified twice in two different interviews. Moreover, the interviewees represent a relatively homogeneous group as they all work as developers in the electric vehicle sector. One interviewee was from a different company than the other three which could have affected his responses as well. However, no major difference compared to the other responses was determined.

6. Security Requirements Classification

This chapter addresses the second research question that examines GPT's ability to map the security requirements of EVerest to system model elements of SecLan[57]. The classification is a first step in the creation of trace links, which are required for automatic verification. The SecLan model helps to create a relationship between system elements and security domain specific languages. It includes a system model containing nine element types, among others, *component*, *entity*, and *data*. As classification techniques, prompt engineering and fine-tuning of GPT 4 and 3.5 turbo are used. Prompt engineering uses GPT 4, while the latest GPT model available for fine-tuning is GPT 3.5 turbo. In the end, the GPT results of both approaches are evaluated. The evaluation has multiple objectives. The first aim is to find out how well GPT identifies SecLan's element types in the security requirements. Then, it is determined how well GPT extracts the corresponding phrases. Finally, the quality of extraction of phrases is evaluated regarding their element types. For the classification, an extendable Python framework is developed as part of this thesis. Its architecture is presented in Section 6.1. Then, the approaches of prompt engineering and fine-tuning are described in Section 6.2. Section 6.3 contains the evaluation scenarios, the gold standard creation, and its results. The chapter concludes with a discussion of the evaluation results and the threats to validity in Section 6.4.

6.1. Architecture

For the classification task, a CLI tool is designed and implemented in Python. The tool has several functionalities. It can be used for prompt engineering and fine-tuning. Finally, the results can be evaluated in different scenarios using a gold standard. The code is grouped into multiple packages: evaluation, prompting, fine_tuning, and helper. A rough overview of the architecture containing the most important classes can be seen in Figure 6.1. The abstract class `Runner` contains an abstract `run()` method. `Runner` has four different child classes, `RunnerPrompting`, `RunnerDataPreparerTuning`, `RunnerTuning`, and `RunnerEval` that are not shown in Figure 6.1. `RunnerPrompting` and `RunnerDataPreparerTuning` both need prompts to work. The enum `Prompt` offers the prompts `CLASSIFY_ALL`, `IDENTIFY_ALL`, `JSON_ALL`, and `NOT_FOUND_ALL`. A second enum that holds the definitions of the element types is not depicted in the diagram for simplicity. Prompts can be concatenated using the `PromptBuilder`. The users can choose which pre- and post-prompts he wants to use and whether he wants to include the definitions of the element types or not. The definition prompt flag reflects the decision of the user. Pre-prompts come before the definition prompt while post-prompts come after it. The attribute element type can be specified as well in case one specific element type is desired that should be named in the prompt. The string that stands directly in front of the requirement to be prompted can also be changed by the user by choosing another value of the enum `RequirementIntroducer`. This enum is also not shown for simplicity. `RunnerPrompting` can either prompt the `AIModel` for a single requirement or a whole JSON file using the respective methods of the `PromptExecutor`. The single requirement can be inserted by the user via the command line. The file contains a content list with entries that contain the key "requirement" with the respective requirement. If the user inputs a JSON file, the output will be in a second JSON file with the results of the prompting task. The `AIModel` directly calls the OpenAI API which is encapsulated in its methods. If the user wants to prompt a different LLM than GPT, she can implement the defined methods of AIModel that are kept independent from the OpenAI API.

`RunnerDataPreparerTuning` is responsible for splitting up the data for fine-tuning in the respective training, validation, and evaluation set. The user can choose between a k-fold split and a manual specification of the training and validation set sizes. The training and validation file follow the schema and format specified on the OpenAI website. They consist of a system message containing the prompt, a user message with the requirement, and an assistant message with the expected output of the fine-tuned model. The required format for GPT is JSON Lines (JSONL). The evaluation file contains the requirement, its ID, and the expected output. The training and validation files can directly be used in `RunnerTuning`, which performs the fine-tuning task using the `AIModel`. The user can retrieve the state of the fine-tuning task with the help of `RunnerTuning`. In case of a successful completion, the corresponding method of `AIModel retrieve_status_tuning()` returns the ID of the fine-tuned model.

`RunnerEval` takes two JSON files as input, the gold standard and the GPT results. They are compared against each other according to their element types, and indicative phrases. The results of the comparison and the evaluation metrics are written in CSV files. The abstract class `EvalStrategy` provides the abstract method `evaluate()`. It only stores the number of true positives, false positives, and false negatives as well as the complete evaluation results with the requirement ID to be flexible with regard to the calculation of the evaluation scenario. `EvalElementType` and `EvalIndicativePhrase` provide the template method `evaluate()` for their child classes that implement a concrete evaluation scenario. Additional evaluation classes that inherit from `EvalStrategy`, `EvalElementType`, and `EvalIndicativePhrase` are not shown for simplicity. Classes of the helper package are not shown either. The omitted classes are responsible for creating and processing JSON and CSV files.

The architecture has been designed to be easily extendable and interchangeable. There are default values set for prompts, element types, evaluations, train, and validation sizes as well as the GPT model in case nothing else is specified. By default, the definitions of the element types are used as more context helps GPT to achieve better results. New prompts can be added to the enum and directly used. The user decides if the prompt should be sent as a system or user prompt to the `AIModel`. For the fine-tuning task, train and validation size can be passed as parameters in the CLI. The user can choose which evaluations

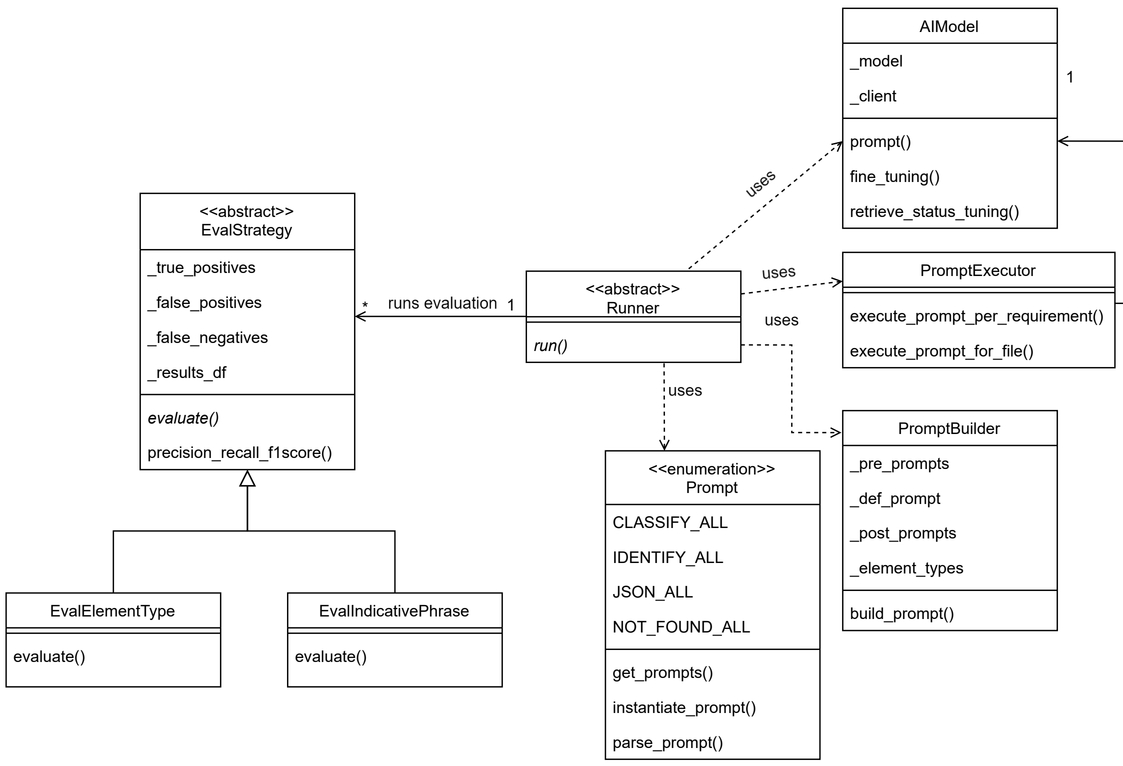


Figure 6.1.: Rough overview of the architecture

she wants to perform, by default, all are carried out. The file names for the prompting, fine-tuning, and evaluation tasks have to be passed in as command-line arguments.

6.2. Classification Techniques

This thesis aims to classify elements in a security requirement and extract the corresponding phrase. The classified elements are element types of SecLan's[57] system model. The corresponding phrase to be extracted from the requirement is referred to as the indicative phrase. The classification forms a basis for the creation of a trace link between the requirement and the architecture model. Trace links can then be used for an automatic verification of the requirement. For the classification, prompt engineering (c.f. Section 2.7.1) and fine-tuning (c.f. Section 2.7.2) are used. Prompt engineering uses GPT 4, whereas fine-tuning stays with GPT 3.5 turbo, as this is the latest version available for fine-tuning. Both approaches work with prompts. A prompt can be composed of an instruction, context, input data, and output indicators [24]. Han et al. [30] state that a prompt should include five elements, an instruction of the task to be performed, the target labels, a description of the output format, the input data, and examples in case of few-shot prompting. That is consistent with Girays[24] suggestion, which is why it is used as the basis for constructing the prompts in this work. Instructions can be sent to GPT in a system or user message.

The system message can be used to better control the output of GPT through the provided style and instructions. It has a different influence on the model than the user message.

Multiple prompts were designed by the author of this thesis to instruct GPT. They are listed below.

CLASSIFY_ALL Classify the requirement whether it contains the element types activity, component, data, entity, state or not. Extract the indicative phrases for the element types in the requirement.

IDENTIFY_ALL Identify phrases in the requirement that are of the element type activity, component, data, entity, or state.

JSON_ALL List all the elements of all element types in a JSON array, providing for each entry the keys indicativePhrase and elementType.

NOT_FOUND_ALL If there is no element type in the requirement, return an empty list.

The two pre-prompts *CLASSIFY_ALL* and *IDENTIFY_ALL* ask GPT to classify the requirement into the given element types and extract the respective indicative phrases. Then, the post-prompt *JSON_ALL* specifies the output format that should contain valid JSON with the correct keys. The second post-prompt *NOT_FOUND_ALL* indicates what GPT should do if no element type is present in the requirement to be able to automatically process GPT's output. Between the pre- and post-prompts, the definitions of the element types can be provided as context to GPT. SecLan's[57] system model consists of nine element types. The classification is restricted to five element types and performed on the element types *activity*, *component*, *data*, *entity*, and *state*. All element types are explicitly named in the prompt. Their definitions can be found in Section 2.5. By default, *IDENTIFY_ALL* is set as pre-prompt and *JSON_ALL* and *NOT_FOUND_ALL* as post-prompts.

Zero-shot prompt engineering was performed using GPT 4, the newest GPT model at the time of writing the thesis. First, *IDENTIFY_ALL* was used as a pre-prompt, then, *CLASSIFY_ALL* was tested as well. The default post-prompts were used in both cases as well as the definitions of the element types. Both prompt scenarios, with the instructions in the system and in the user message were tested and compared. The case where the instructions were in the system message is referred to as system mode, whereas the other case is referred to as without system mode. The requirement to be classified was always sent in the user message at the end of the prompt, introduced by a string such as, for example, *REQUIREMENT:* . Furthermore, all four different prompt scenarios were carried out three times to be able to assess the stability of the results. The input for GPT comprised all requirements elicited from the interviews (c.f. Section 5.7). Then, the output of GPT was evaluated by comparing it against the gold standard. Due to the fact that the above prompts were used, the results may not be generalizable.

GPT 3.5 turbo was used for fine-tuning. A 5-fold cross-validation was performed to evaluate the quality of the fine-tuned models. In a 5-fold cross-validation, four of the five folds are used for training and the remaining one for the evaluation of the fine-tuned model until all folds have been used once for evaluation. Therefore, the requirements were shuffled and the dataset was split into five folds. Since the dataset consists of 93 requirements, the first four folds contained 18 elements, while the last one was composed of 21 elements. OpenAI expects the data for fine-tuning in JSONL format that contains a set of messages with a system, user, and assistant role. The system message included the above prompts that were also used in the prompt engineering task, whereas the user message contained the requirement. The ideal response that the model should learn was provided in the assistant message. It consisted of a list with the pairs of the element type and the minimal term of the indicative phrase in JSON format. Therefore, all five element types *activity*, *component*, *data*, *entity*, and *state* were requested simultaneously as they can be part of the expected element types for a requirement. For each training, ten percent of the training data were removed from the training set and then, these seven elements were used to validate the fine-tuned model after training. In total, five models were fine-tuned in one iteration. The performance of the fine-tuned model was determined by reusing the fine-tuning prompt and performing prompt engineering with the requirements of the evaluation fold. Both prompting with and without system mode were performed. The results were compared with the correct solutions in the evaluation fold. Each of the four scenarios with the two different pre-prompts and the system and user message was run three times to assess the stability of the results.

6.3. Evaluation

In this section, the evaluation scenarios that are addressed in the second research question are described in detail. The first goal *element type classification* is to find out, how well GPT classifies SecLan's[57] element types in the security requirements of EVerest. The *evaluation of the element type classification* (eval1) considers every element type instance of a requirement separately, whereas *evaluation of the element type classification per requirement* (eval2) analyzes how well GPT classifies all expected element types per requirement. The second goal *phrase extraction* addresses GPT's ability to extract indicative phrases out of the requirement. The *evaluation of the phrase extraction* (eval3) analyzes how well GPT extracts correct phrases over all requirements independent of its element type. The *evaluation of the phrase extraction per requirement* (eval4) determines how well GPT extracts all expected phrases per requirement. Eval4 is the equivalent of eval2, just as eval1 and eval3 are equivalent. The third goal *phrase extraction regarding the element type* covers GPT's ability to extract the correct phrase regarding the element type. The *evaluation of the phrase extraction regarding the element type* (eval5) checks each indicative phrase separately if its element type is correct, contrary to *evaluation of the phrase extraction regarding the element type per requirement* (eval6) that determines how well GPT extracts all indicative phrases per requirement correctly, provided all element types for the requirement are correct. All evaluation scenarios are performed for both prompt engineering and fine-tuning. Prompt engineering uses GPT 4 and fine-tuning is performed on GPT 3.5 turbo. For the evaluations, a gold standard is created which contains a list of pairs of element type and its corresponding indicative phrase for each requirement. The indicative phrase is divided into minimal term and phrase. The minimal term is the term that is at least expected by GPT to indicate that the element type has been identified in the requirement. The phrase contains the minimal term and describes the part of the requirement that is still acceptable. For the evaluations, the five annotated element types *activity*, *component*, *data*, *entity*, and *state* are considered. In all comparisons of the GPT results with the gold standard, capitalization and the order of the pairs of element type and indicative phrase are not taken into account. The six evaluation scenarios are described in more detail in Section 6.3.1, Section 6.3.2, and Section 6.3.3, two in each section. The creation of the gold standard is provided in Section 6.3.4. At the end, the evaluation results are presented in Section 6.3.5.

6.3.1. Element Type Classification

The first goal *element type classification* is to find out, how well GPT classifies element types in a requirement as this is a first step towards the mapping of a requirement to a system element. The *evaluation of the element type classification* (eval1) considers every element type separately. The element types *activity*, *component*, *data*, *entity*, and *state* are checked individually to see how well they are classified by GPT across all requirements. Eval1 addresses a multi-class problem, since for one instance of an element type in the GPT results, it is determined whether this element type occurs in the gold standard for the respective requirement or not. Multiple instances of one element type can exist in one requirement as well. A true positive is counted for an element type if an instance of that element type is found in the gold standard and the GPT results for a requirement. Multiple true positives per requirement are possible. If the gold standard expects an instance of an element type that is not present in the GPT results, this is a false negative for the respective element type. On the contrary, a false positive is counted, if GPT outputs an instance of an element type that was not expected by the gold standard for this requirement. If there are elements in the GPT results list, that do not correspond to one of the five given element types, they are counted as false positives for the class *other*. The true positives, false positives, and false negatives are used to calculate the metrics introduced in Section 2.8. First, precision, recall, and F1-score are calculated. Then, the macro, weighted, and micro averages are determined. The 5-fold cross-validation uses the macro average over the precisions, recalls, and F1-scores of the five models to have an unweighted average of one cross-validation. Additionally, the weighted averages are calculated as the size of the last evaluation set is slightly higher than the other four. Furthermore, the micro averages of precision, recall, and F1-score are calculated to assess how well the approach works for all element types. The micro average is used because the dataset is imbalanced, as there are 146 components and only 49 states. It gives equal weight to each instance by taking the sums of the true positives, false positives, and false negatives of all element types. In the end, the macro macro averages and macro micro averages for fine-tuning and the macro average for prompt engineering are calculated to have an unweighted average of the three runs as a general statement.

The *evaluation of the element type classification per requirement* (eval2) addresses a multi-label problem per requirement as multiple element types can be correct for one requirement. If all the element types of one requirement match the gold standard for that respective requirement, a true positive is counted. Otherwise, the length of both lists is compared. If the GPT results contain more elements, the false positive counter is increased by one. If the gold standard has more elements, the GPT result is counted as a false negative, since element types are missing. In case, the lists have the same length but not all element types are matching, this is counted as a false negative and a false positive as at least one element of the gold standard is missing but at the same time, GPT mistakenly found some other element. For eval2, first, precision, recall, and F1-score (c.f. Section 2.8) are calculated to determine the quality of the GPT results. Then, for the 5-fold cross-validation, the macro and weighted averages are calculated as well. Finally, the macro average of all three executions of prompt engineering and fine-tuning is calculated to have an unweighted average.

6.3.2. Phrase Extraction

The second goal *phrase extraction* determines GPT's ability to extract indicative phrases out of the requirements, as a trace link creation needs a classified element in the requirement to map it to one of the system elements. As in the previous goal *element type classification* (c.f. Section 6.3.1), all phrases are first considered individually across all requirements and then per requirement. For both evaluations, the extracted phrases are compared with the gold standard to determine whether the minimal term is present in the extracted phrase and whether the extracted phrase lies within the permitted phrase of the gold standard. The *evaluation of the phrase extraction* (eval3) considers every indicative phrase separately and counts a true positive if the extracted phrase contains the minimal term of the gold standard and lies in the accepted phrase of the gold standard. Multiple true positives for multiple correct phrases of one requirement are possible in eval3. If the extracted phrase does not meet the above-described criteria, it is considered a false positive and false negative because it is not the expected but a different phrase. If the number of extracted phrases in one requirement exceeds the number of indicative phrases from the gold standard, the remaining phrases are counted as false positives. Otherwise, if the number of extracted phrases is smaller than the expected phrase number, additional false negatives corresponding to the number of missing extracted phrases are counted. Precision, recall, and F1-score are calculated for prompt engineering and fine-tuning as described in Section 2.8. For the 5-fold cross-validation, the macro and weighted average are calculated to have both an unweighted and weighted value. In the end, the macro average is calculated over the three runs of prompt engineering and fine-tuning.

The *evaluation of the phrase extraction per requirement* (eval4) checks all indicative phrases per requirement. It is assessed whether all indicative phrases of a requirement in the GPT results match the defined phrases of the gold standard for the respective requirement with the method described at the beginning of this subsection. If yes, a true positive is counted for that requirement. If no, depending on the length of the two lists containing the indicative phrases, a false positive, false negative, or both are counted analogously to *evaluation of the element type classification per requirement* (c.f. Section 6.3.1). Unlike eval3, there is only one classification result per requirement. As in *evaluation of the element* *type classification per requirement* (c.f. Section 6.3.1), precision, recall, and F1-score are calculated. For the 5-fold cross-validation, the macro and weighted averages are calculated. In the end, the macro average over all three runs of prompt engineering and fine-tuning is calculated.

6.3.3. Phrase Extraction regarding the Element Type

The two previous goals *element type classification* and *phrase extraction* separately analyze element types and phrases. The third goal *phrase extraction regarding the element type* deals with GPT's ability to extract the indicative phrases regarding the element type. For a trace link creation, it is important that both the correct element type and the correct associated phrase are found. Therefore, in the next two evaluations, only phrases where the associated element type is a true positive are compared against the gold standard to see if the phrase is also correct. The comparison of the extracted phrase with the gold standard is performed analogously to *evaluation of the phrase extraction* (eval3) (c.f. Section 6.3.2). The *evaluation of the phrase extraction regarding the element type* (eval5) analyzes the indicative phrases of the correct element types for each pair of element type and associated phrase individually and separately for each element type. Multiple true positives are possible for one requirement. The metrics are calculated analogously to *evaluation of the element type classification* (c.f. Section 6.3.1), as both evaluations examine the results separately by element type as well as aggregated over all element types. Therefore, first, precision, recall, and F1-score are calculated. For the 5-fold cross-validation macro and weighted averages are calculated. In the end, for all three runs, the macro macro and macro micro averages are calculated for fine-tuning, and the macro average is calculated for prompt engineering.

The last evaluation, *evaluation of the phrase extraction regarding the element type per requirement* (eval6) checks how well GPT extracts the indicative phrases for a requirement in the case that all element types are correct. First, for each requirement, it is checked if all its element types are correct (c.f. Section 6.3.1). If yes, the requirement is analyzed, whether also all corresponding phrases match those in the gold standard for the respective requirement (c.f. Section 6.3.2). A true positive is counted if all phrases of the requirement include the minimal term and lie in the accepted phrase of the gold standard. If one or more phrases do not match the gold standard, a false positive and false negative are counted, as GPT incorrectly found another phrase but the correct phrase is missing. All metrics are calculated analogously to *evaluation of the element type classification per requirement* (eval2) (c.f. Section 6.3.1) because both evaluations are per requirement and no further distinction between the different element types are made. Precision, recall, and F1-score are calculated, as well as macro and weighted average for the 5-fold cross-validation. For a general statement on all three runs of prompt engineering and fine-tuning, the macro average is determined.

6.3.4. Dataset and Gold Standard Creation

To evaluate the GPT results, a gold standard is required against which the results can be compared. Therefore, a new dataset was created in which each entry represents a requirement. The requirements are the specified requirements from the interviews (c.f. Section 5.7). The security objective, which was collected through the questionnaire, is also part of each entry. The pseudonym of the author of the requirement, who is one of the interviewees, and his confidence in the domain of software security were added as additional information to each entry as well. Furthermore, a numerical identifier, and the identifier of the original requirement are part of each entry. Thereby, it is possible to trace each fine-grained requirement from the interview to its original, coarse-grained requirement from the questionnaire. The identifier of requirements that were elicited during the interviews without previous coarse-grained requirements was set to minus one. The dataset was created as a CSV file and then transformed into a JSON file. The evaluation requires pairs of element types and their corresponding indicative phrase for each requirement in the gold standard.

The creation of the gold standard was done by the author of this thesis. The minimal terms were retrieved manually from the requirements whereas the phrases were added semi-automatically to the gold standard. To classify the element types manually, the definitions of the system model elements described in Section 2.5 were used. In this thesis, the element types *activity*, *component*, *data*, *entity*, and *state* were annotated. The remaining four element types were omitted from the gold standard. In the following, the procedure for classifying the different element types and extracting the minimal term is described.

For the annotation of activities, the definition of Section 2.5 has been expanded so that activities in which the actor is not explicitly mentioned are also collected, e.g., “the database should be backed up”(ID4) or “Tokens used for authentication should not be stored in plain text”(ID5). The fine-grained requirements that were elicited in the interviews still have a level of abstraction that is often too high to identify individual classes that perform an *activity*. Therefore, the actions of components are also counted as activities. Only the verb itself is taken as the minimal term for the element type *activity*. If the activity contains a negation such as “not written”, the negation is also part of the minimal term, otherwise, the minimal term would indicate the opposite. No activities are, e.g., “should not leak”, “must comply”, or “implement”, as this does not describe an active operation of a system *component* or *entity*.

Software parts of EVerest are annotated as components. The name of the *component* is taken as minimal term, for example, “OCPP”, “ISO15118”, “Auth”, and “EvseSecurity” in the requirement “OCPP, ISO15118, Auth module and EvseSecurity should be tested with fuzzed input” (ID9). In the case of more general formulations such as “all modules”, “EVerest modules”, or “certain modules” only “modules” is specified as the minimal term. OCPP and ISO 15118 are both, specifications and names of components in EVerest. If mentioned as a specification, they are not collected as a *component*. If a requirement contains multiple comma-separated components, the gold standard classifies them as separate components as can be seen in the example above. “EVerest system” or “the application” are too coarse-grained and therefore not retrieved as components.

Files, messages, tokens, or any other information are categorized as the element type *data*. The corresponding minimal term should describe the data sufficiently to know what is meant. It is sufficient to take “authentication token” as a minimal term out of the requirement “The OCPP modules and the authorization related modules (Auth, TokenProvider, TokenValidator) shall ensure that no plaintext authorization tokens are logged” (ID25).

Persons such as technicians and users who interact with the system as well as external systems or devices such as backend and EV are categorized as *entities*. Interfaces of EVerest such as “token_provider” are also classified as entities as they are software objects but not a complete component. In this case, “interface” must occur in the minimal term to ensure the distinguishability of components with the same name.

For the annotation of the element type *state*, its definition has been extended as well. In Section 2.5, *state* is described as the state of an *entity* or *activity*. For this work, the state of *components* is considered as a state too, since the requirements are often at a high abstraction level, so that only components and not individual classes, which would correspond to entities, can be annotated. Therefore, in the sentence “The state of EVerest modules should always be consistent” (ID1), “consistent” is categorized as a *state* of the *component* “modules”. The remaining four element types *information flow*, *control flow*, *connection*, and *node* are not annotated, as this would go beyond the scope of this work. For all element types, the minimal term was not taken coherently out of the requirements. If one word occurs multiple times in a requirement, it will be included in the gold standard multiple times. In some requirements, the interviewees wrote examples, further specifications, or abbreviations in brackets after the respective requirement. The words in brackets were classified if they describe an element type in more detail, e.g., a *component* or a *data*. Otherwise, they were left out as they are usually a repetition.

Phrases were extracted out of the requirements and included in the gold standard, so that GPT does not have to output exactly the minimal term, but a slightly longer phrase is also acceptable. A different phrase was extracted depending on the element type. The element types *component*, *data*, and *entity* are classified in a requirement when they are mentioned as nouns. Therefore, noun phrases were chosen for these three element types. The noun phrase can start with a determiner, possibly followed by an adjective or a gerund verb. Then, at least one noun or proper noun in singular or plural follows. After that, a preposition, noun, coordinating conjunction, and another noun may follow. That covers simple and complex noun phrases. This pattern extracts the *entity* “no unauthorized persons” in the requirement “No unauthorized persons shall be able to access more than the user GUI of a charging station” (ID40). In contrast to *components*, *data*, and *entities*, *activities* are usually expressed using verbs. Therefore, the verb phrase is selected for the element type *activity*. A verb phrase may consist of a modal verb, possibly followed by an adverb. At least one main verb in its base form has to be part of the verb phrase, possibly followed by a verb in past participle or an adjective. With this pattern, the phrase of the *activity* “shall be verified” is extracted in the sentence “Delivery of updates should

element type	#elements
activity	98
component	146
data	112
entity	112
state	49

Table 6.1.: Number of elements per element type

happen over secure channels (e.g HTTPS), update signatures shall be verified by the System module" (ID48). In opposition to the first four element types, *states* are usually expressed by adjectives and adverbs. With the regex of the element type *state*, all adjectives, adverbs, and also verbs in past participle are extracted. Sometimes, verbs in the past participle describe the state of a *component* or *entity*, e.g., "authorized" in the requirement "It shall be ensured that the API module that exposes the MQTT interface is used only by authorized external components" (ID29).

The NLTK library is used to extract noun phrases, verb phrases, adjectives, and adverbs from the requirements. NLTK first tokenizes the requirements and tags the words. After that, the regex parser extracts phrases based on a defined regex structure. Some words are incorrectly tagged due to their ambiguity, e.g., "crashes" can be the plural of a crash or the conjugated verb in the third person. Nonetheless, this process is chosen because it is easier to decide which phrase to put in the phrase field if there are already suggestions. Therefore, the results of the NLTK script are taken as a baseline that can be used as a reference, but which is not taken without verification. The script calculates all verb phrases, not only those that correspond to the definition of an *activity*. Therefore, the adapted phrases are manually matched to their corresponding minimal term. If, e.g., two activities appear one after the other in a phrase, the same phrase is selected for both elements. A phrase is determined manually for the elements that do not appear in the results of the phrase calculation. The number of annotated elements for each element type in the gold standard can be found in Table 6.1.

6.3.5. Results of Evaluations

This subsection describes the evaluation results of the six evaluation scenarios (c.f. Section 6.3.1, Section 6.3.2, and Section 6.3.3). For the evaluations, the gold standard (c.f. Section 6.3.4), which was derived from the elicited security requirements of EVerest at design level (c.f. Chapter 5) is used. In each evaluation, the results of prompt engineering on GPT 4 are compared with fine-tuning on GPT 3.5 turbo. Most of the evaluations indicate that fine-tuning delivers much better results than prompt engineering even if fine-tuning uses GPT 3.5 turbo and prompt engineering the newer version GPT 4.

During the evaluations of the GPT results, GPT sometimes did not output a valid JSON format. This was corrected manually for both prompt engineering and fine-tuning. Mostly, the requirement with the identifier five was affected, sometimes also requirement 34. Requirement five is especially long and consists of eight sentences which may be a reason for GPT's mistake. In the classification, GPT appended the same element pair, e.g., the element type "entity" with the indicative phrase "modules" dozens of times till the output length was reached. To fix this error, all repeating element pairs were deleted. This occured very rarely but can have an influence on the evaluation results. For prompt engineering, in some cases, GPT did not answer in valid JSON format and wrote some sentences before adding the JSON format. These sentences were deleted as well. The results of the evaluations regarding the element type classification are presented in Section 6.3.5.1. Section 6.3.5.2 contains the results of the evaluations concerning the phrase extraction. In Section 6.3.5.3, the results of the evaluations of phrase extraction regarding the element type are presented. All evaluations compare the best prompt engineering approach with the best and worst fine-tuning approach. They compare the different prompts and modes as well and determine how good the achieved results are.

6.3.5.1. Element Type Classification

The results of *evaluation of the element type classification* (eval1) (c.f. Section 6.3.1) (n=517) show that the difference between the prompt engineering and fine-tuning results varies depending on the element type. For the element type *component*, fine-tuning delivers slightly better results than prompt engineering. The best prompt engineering approach (F1 - Prompt$_{\text{classify\_all}}$ system mode: 0.8) is only 1% point lower than the worst fine-tuning result (F1 - Prompt$_{\text{classify\_all}}$ system mode: 0.81) and 4% points lower than its best prompt (F1 - Prompt$_{\text{identify\_all}}$ system mode: 0.84). The element type *data* achieves equally good results in prompt engineering and fine-tuning (F1 - Prompt$_{\text{identify\_all}}$ system mode: 0.79) for the same prompt. The worst fine-tuning result (F1 - Prompt$_{\text{identify\_all}}$ without system mode: 0.75) is only 2% points better than the worst prompt engineering result (F1 - Prompt$_{\text{classify\_all}}$ with/without system mode: 0.73). For the element type *entity*, the results of the two prompts used differed noticeably, therefore, they are compared separately. The best prompt engineering result (F1 - Prompt$_{\text{identify\_all}}$ without system mode: 0.65) even exceeds fine-tuning (F1 - Prompt$_{\text{identify\_all}}$ without system mode: 0.62) by 3% points. For the other prompt, prompt engineering (F1 - Prompt$_{\text{classify\_all}}$ in system mode: 0.6) is 1% point lower than fine-tuning (F1 - Prompt$_{\text{classify\_all}}$ in system mode: 0.61). The element type *state* obtains better results in fine-tuning than in prompt engineering. The best prompt engineering result (F1 - Prompt$_{\text{identify\_all}}$ with system mode: 0.57) is only 1% lower than the worst fine-tuning result (F1 - Prompt$_{\text{identify\_all}}$ with system mode: 0.58) and 8% points lower than the best fine-tuning result (F1 - Prompt$_{\text{identify\_all}}$ without system mode: 0.65). For the element type *activity*, fine-tuning performs slightly worse than prompt engineering. The best prompt engineering result (F1 - Prompt$_{\text{classify\_all}}$ with system mode: 0.68) is 1% point better than the best fine-tuning result (F1 - Prompt$_{\text{identify\_all}}$ without system mode: 0.67) and 5% points better than the worst fine-tuning result (F1 - Prompt$_{\text{classify\_all}}$ with system mode: 0.63). This can have multiple reasons. The definition of *activity* in the prompt is very strict. Therefore, the fine-tuned models could output less activities than the prompt engineering approach. One observation is that the recall of prompt engineering is 31% points higher than for fine-tuning whereas the precision of prompt engineering is 17% points lower than in fine-tuning. It follows that GPT classifies more activities in prompt engineering than in fine-tuning. It might be that the fine-tuned models got too less training examples as the gold standard for *activity* only contains 98 elements whereas *component*, for example, had 146 examples (c.f. Table 6.1).

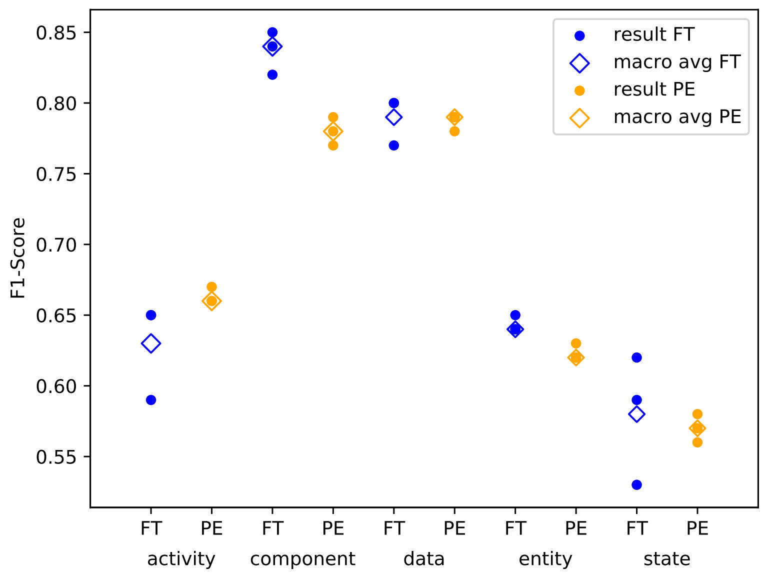


Figure 6.2.: Evaluation of the element type classification: Comparison of fine-tuning (FT) and prompt engineering (PE) for all element types for all three runs

Figure 6.2 depicts the prompt engineering and fine-tuning results for the overall best prompt engineering prompt Prompt$_{\text{identify\_all}}$ in system mode. The diagram also shows that the results for common understanding such as *component* or *data* are better than the other three element types. *Component* and *data* are relatively clearly defined concepts for which the common understanding is not restricted by the definitions in this thesis. More detailed results of the individual element types can be found in Table A.1 and Table A.2 in Appendix A.5.

Table 6.2 shows the macro micro averages of all element types of the three runs of prompt engineering and fine-tuning. It can be seen that fine-tuning performs slightly better than prompt engineering. The best prompt engineering result (F1 - Prompt$_{\text{identify\_all}}$ with/without system mode: 0.7) is equally good as the worst fine-tuning result (F1 - Prompt$_{\text{classify\_all}}$ with system mode: 0.7) and 3% points worse than the best fine-tuning result (F1 - Prompt$_{\text{identify\_all}}$ with/without system mode: 0.73). The results of both prompts differ only slightly. The fact that the recall in prompt engineering is higher than precision ranging from 3 to 15% points is especially due to the high recall of the element type activity. For the other element types, the recall of prompt engineering and fine-tuning do not differ as much or is higher for fine-tuning. In summary, these results show that GPT can actually be used well for the element type classification in fine-tuning (F1: 0.7-0.73). Even prompt engineering (F1: 0.67-0.7) is still usable if no training data is available for the elements to be classified.

		Precision	Recall	F1-score
Prompt$_{\text{identify\_all}}$	**Prompt Engineering**			
	without system mode	0.63	0.78	0.7
	with system mode	0.64	0.78	0.7
	Fine-tuning			
	without system mode	0.7	0.76	0.73
	with system mode	0.73	0.73	0.73
Prompt$_{\text{classify\_all}}$	**Prompt Engineering**			
	without system mode	0.65	0.7	0.67
	with system mode	0.67	0.7	0.69
	Fine-tuning			
	without system mode	0.71	0.73	0.72
	with system mode	0.71	0.7	0.7

Table 6.2.: Evaluation of element type classification: Macro micro averages over all element types, Prompt engineering uses GPT version 4, Fine-tuning uses GPT version 3.5 turbo, n=517

Table 6.3 presents the macro averages of the *evaluation of the element type classification per requirement* (eval2) (c.f. Section 6.3.1) (n=93) of the three runs of prompt engineering and fine-tuning. It is apparent that fine-tuning performs much better than prompt engineering in precision, recall, and F1-score. The best prompt engineering prompt (F1 - Prompt$_{\text{classify\_all}}$ with system mode: 0.09) is 10% worse than its fine-tuning result (F1 - Prompt$_{\text{classify\_all}}$ with system mode: 0.19). The best fine-tuning result (F1 - Prompt$_{\text{identify\_all}}$ with/without system mode: 0.2) performed 11% points better than the best prompt engineering result, whereas the worst fine-tuning result (F1 - Prompt$_{\text{classify\_all}}$ without system mode: 0.18) still is 9% points better than the best prompt engineering result. The differences between the two prompts used and the modes with and without system mode are small. For fine-tuning, Prompt$_{\text{identify\_all}}$ delivers slightly better results (2% points) in the F1-score, whereas Prompt$_{\text{classify\_all}}$ performs better in the F1-score for prompt engineering without system mode (2% points). As this evaluation addresses a multi-label problem, the fine-tuning results (F1-score: 0.2) are acceptable. However, prompt engineering seems to be inappropriate to classify element types correctly per requirement. These results indicate that it is generally hard for GPT to identify all element types per requirement correctly.

6.3.5.2. Phrase Extraction

Table 6.4 provides the macro averages of the three runs for prompt engineering and fine-tuning of the *evaluation of the phrase extraction* (eval3) (c.f. Section 6.3.2) (n=517).

		Precision	Recall	F1-score
Prompt$_{\text{identify\_all}}$	**Prompt Engineering**			
	without system mode	0.05	0.1	0.07
	with system mode	0.07	0.12	0.09
	Fine-tuning			
	without system mode	0.19	0.22	0.2
	with system mode	0.22	0.2	0.2
Prompt$_{\text{classify\_all}}$	**Prompt Engineering**			
	without system mode	0.08	0.1	0.09
	with system mode	0.09	0.09	0.09
	Fine-tuning			
	without system mode	0.18	0.18	0.18
	with system mode	0.2	0.18	0.19

Table 6.3.: Evaluation of the element type classification per requirement: Macro averages of precision, recall, F1-score, Prompt engineering uses GPT version 4, Fine-tuning uses GPT version 3.5 turbo, n=93

What stands out is that fine-tuning delivers noticeably better results than prompt engineering. In contrast to the best prompt engineering approach (F1 - Prompt$_{\text{identify\_all}}$ with system mode: 0.52), fine-tuning had an increasement of 18% points for its best prompt (F1 - Prompt$_{\text{classify\_all}}$ without system mode: 0.7) and 16% points for its worst (F1 - Prompt$_{\text{classify\_all}}$ with system mode: 0.68). As in the *evaluation of element type classification per requirement* (eval2), Prompt$_{\text{identify\_all}}$ works better for prompt engineering, whereas Prompt$_{\text{classify\_all}}$ delivers better results for fine-tuning. A closer inspection of GPT's prompt engineering results shows that GPT outputs longer phrases. This is consistent with the observations of Han et al. [30]. The extracted phrases are often longer than the accepted gold standard phrases. The fine-tuning results are generally much shorter phrases and, therefore, lie more often in the accepted phrase. In fine-tuning, the number of phrases produced by GPT is relatively similar for the two prompts used, with Prompt$_{\text{identify\_all}}$ showing greater variation. In addition, the results of the two prompts are similar, whether with or without system mode. GPT's prompt engineering results are slightly better when using the system mode. Furthermore, GPT extracted more phrases with Prompt$_{\text{identify\_all}}$ (n=647-673) than with Prompt$_{\text{classify\_all}}$ (n=604-629) and in general much more than expected by the gold standard (n=517). Although the number of phrases of the two prompts differs noticeably and Prompt$_{\text{identify\_all}}$ delivers more phrases, its precision is equally good or better (3-4% points). Moreover, the recall for Prompt$_{\text{identify\_all}}$ is 8% points better than for Prompt$_{\text{classify\_all}}$ resulting in a better F1-score for Prompt$_{\text{identify\_all}}$ (7% points). This may be because Prompt$_{\text{identify\_all}}$ produces in parts shorter phrases than Prompt$_{\text{classify\_all}}$ resulting in more accepted phrases. Moreover, GPT separates individual elements of a phrase better with Prompt$_{\text{identify\_all}}$. In the sentence "In the case of a OCPP connection from the charger to the CSMS transaction related data, telemetry and information about updates is shared with relevant parties.", GPT with Prompt$_{\text{identify\_all}}$ extracts the three phrases for *data*: “CSMS transaction related data”, “telemetry”, and “information about updates”. Contrary, GPT with Prompt$_{\text{classify\_all}}$ extracts only the element “transaction related data, telemetry and information about updates”. Overall, GPT extracts the phrases well during fine-tuning, while prompt engineering only delivers mediocre results.

		Precision	Recall	F1-score
Prompt$_{\text{identify\_all}}$	**Prompt Engineering**			
	without system mode	0.45	0.55	0.49
	with system mode	0.48	0.57	0.52
	Fine-tuning			
	without system mode	0.65	0.71	0.68
	with system mode	0.69	0.68	0.68
Prompt$_{\text{classify\_all}}$	**Prompt Engineering**			
	without system mode	0.41	0.44	0.42
	with system mode	0.45	0.47	0.45
	Fine-tuning			
	without system mode	0.69	0.71	0.7
	with system mode	0.69	0.67	0.68

Table 6.4.: Evaluation of the phrase extraction: Macro averages of precision, recall, and F1-score, Prompt engineering uses GPT version 4, Fine-tuning uses GPT version 3.5 turbo, n=517

Table 6.5 shows the macro averages for the *evaluation of the phrase extraction per requirement* (eval4) (c.f. Section 6.3.2) (n=93). It illustrates that fine-tuning delivers noticeably better results than prompt engineering. The best prompt engineering prompt (F1 - Prompt$_{\text{identify\_all}}$ without system mode: 0.06) performs 10% points worse than its fine-tuning result (F1 - Prompt$_{\text{identify\_all}}$ without system mode: 0.16). The best fine-tuning result (F1 - Prompt$_{\text{classify\_all}}$ with system mode: 0.18) is 12% points better and the worst fine-tuning result (F1 - Prompt$_{\text{classify\_all}}$ without system mode: 0.15) is still 9% points better than the best prompt engineering result. Whether the system mode is used or not seems to have no significant effect on the results. For prompt engineering, both prompts perform slightly better without system mode. In fine-tuning, the best results are achieved by Prompt$_{\text{classify\_all}}$ with system mode, while Prompt$_{\text{identify\_all}}$ still has an equally good F1-score with and without system mode. In summary, it can be seen that GPT has difficulties in finding all correct phrases per requirement which fits with the results of the respective *evaluation of the element type classification per requirement.*

6.3.5.3. Phrase Extraction regarding the Element Type

The results of the *evaluation of the phrase extraction regarding the element type* (eval5) (c.f. Section 6.3.3) indicate variations between prompt engineering and fine-tuning regarding the element type. For the element type *component*, the best prompt engineering prompt

		Precision	Recall	F1-score
$\text{Prompt}_{\text{identify\_all}}$	**Prompt Engineering**			
	without system mode	0.04	0.08	0.06
	with system mode	0.04	0.07	0.05
	Fine-tuning			
	without system mode	0.15	0.17	0.16
	with system mode	0.18	0.16	0.16
$\text{Prompt}_{\text{classify\_all}}$	**Prompt Engineering**			
	without system mode	0.05	0.06	0.05
	with system mode	0.03	0.03	0.03
	Fine-tuning			
	without system mode	0.16	0.15	0.15
	with system mode	0.19	0.17	0.18

Table 6.5.: Evaluation of the phrase extraction per requirement: Macro averages of precision, recall, F1-score, Prompt engineering uses GPT version 4, Fine-tuning uses GPT version 3.5 turbo, n=93

(F1 - $\text{Prompt}_{\text{identify\_all}}$ without system mode: 0.75) performs 14% points worse than the corresponding fine-tuning result (F1 - $\text{Prompt}_{\text{identify\_all}}$ without system mode: 0.89). The best fine-tuning result (F1 - $\text{Prompt}_{\text{classify\_all}}$ with/without system mode: 0.89) is 14% points better and the worst fine-tuning result (F1 - $\text{Prompt}_{\text{identify\_all}}$ with system mode: 0.87) is 12% points better than the best prompt engineering result. The element type *data* achieves 5% points better results in fine-tuning (F1 - $\text{Prompt}_{\text{identify\_all}}$ with system mode: 0.89) than with the best prompt engineering prompt (F1 - $\text{Prompt}_{\text{identify\_all}}$ with system mode: 0.84). The best fine-tuning result for *data* (F1 - $\text{Prompt}_{\text{classify\_all}}$ with system mode: 0.91) is 7% points better, and the worst fine-tuning result (F1 - $\text{Prompt}_{\text{identify\_all}}$ with system mode: 0.85) is still 1% point better than the best prompt engineering result. For the element type *entity*, the best prompt engineering prompt (F1 - $\text{Prompt}_{\text{identify\_all}}$ with system mode: 0.67) is 19% points worse than its fine-tuning result (F1 - $\text{Prompt}_{\text{identify\_all}}$ with system mode: 0.86). The best fine-tuning results (F1 - $\text{Prompt}_{\text{classify\_all}}$ with system mode: 0.9) are 23% points better and the worst fine-tuning results (F1 - $\text{Prompt}_{\text{classify\_all}}$ without system mode: 0.85) are 18% points better than the best prompt engineering results. The results for prompt engineering and fine-tuning differ most for the element type *state* whereas fine-tuning noticeably performs better than prompt engineering. The best prompt engineering result (F1 - $\text{Prompt}_{\text{identify\_all}}$ without system mode: 0.25) is 47% worse than its fine-tuning result (F1 - $\text{Prompt}_{\text{identify\_all}}$ without system mode: 0.72). The fine-tuning results differ only by 6% points. The element type *activity* has similar differences in the prompt engineering and fine-tuning results. In contrast to the best prompt engineering approach (F1 - $\text{Prompt}_{\text{identify\_all}}$ with system mode: 0.28), fine-tuning has an increasement of 52% points for its best prompt (F1 - $\text{Prompt}_{\text{classify\_all}}$ with/without system mode: 0.8) and 47% points for its worst (F1 - $\text{Prompt}_{\text{identify\_all}}$ with system mode: 0.75).

The F1-scores of all three runs for prompt engineering and fine-tuning as well as their macro averages of all element types separately are depicted in Figure 6.3 for the average best prompt engineering prompt, $\text{Prompt}_{\text{identify\_all}}$ with system mode. Especially for the element types *activity* and *state*, it can be seen, that fine-tuning noticeably outperforms prompt engineering. It can be observed that general concepts such as *component*, *data*, or *entity* can be recognized relatively well by GPT, also in prompt engineering. For *state* and *activity*, the common understanding diverges from the definitions used in this work as, for example, not all verbs are classified as activities but only those that refer to an *activity* of a *component* or *entity*. This affects not only the classification of the element type, but also the phrase extraction. In prompt engineering, GPT often classifies an *activity* or *state* but does not correctly extract the corresponding phrase. In the requirement "The state of EVerest modules should always be consistent", GPT extracts the phrase "The state of EVerest modules" as a *state* whereas the gold standard expects in the description of the *state*, namely "consistent" as minimal term. Often, the extracted phrase is longer than the accepted phrase in the gold standard. On one hand, there are cases in which the minimal term lies not in the extracted phrase such as the extracted phrase for *state* "the security requirements" (ID5) that does not contain the minimal term "persistent". On the other hand, there are cases in which the minimal term is included in the extracted phrase but the phrase is too long for the accepted phrase such as the extracted phrase for *activity* "is responsible for restarting and orchestrating the EVerest modules" (ID11) that contains the minimal term "restarting". This is especially true for the element types *activity* and *state*, whereas for *component* and *entity*, shorter phrases are extracted by GPT. In some cases, the gold standard may also be too strict. More detailed results to the individual element types can be found in Table A.3 and Table A.4 in Appendix A.5.

Table 6.6 presents the macro micro averages of eval5 over all element types (c.f. Section 6.3.3). It is calculated over all three runs for prompt engineering and fine-tuning. The best prompt engineering approach (F1 - $\text{Prompt}_{\text{identify\_all}}$ with system mode: 0.61) performs 26% points worse than the best fine-tuning (F1 - $\text{Prompt}_{\text{classify\_all}}$ with system mode: 0.87). The worst fine-tuning (F1 - $\text{Prompt}_{\text{identify\_all}}$ without system mode: 0.84) is still 23% points better than the best prompt engineering approach. $\text{Prompt}_{\text{identify\_all}}$ performs better in prompt engineering whereas the best results in fine-tuning are achieved by using $\text{Prompt}_{\text{classify\_all}}$. The results of both, prompt engineering and fine-tuning, are slightly better when using the system mode, namely 1 to 3% points in F1-score. As the evaluation is based on correctly classified element types, the sample size differs in every case. It can be seen that $\text{Prompt}_{\text{identify\_all}}$ has a higher sample size in most cases than $\text{Prompt}_{\text{classify\_all}}$ for both prompt engineering and fine-tuning. Moreover, $\text{Prompt}_{\text{identify\_all}}$ performs better in prompt engineering with a difference of 6% points in F1-score compared to $\text{Prompt}_{\text{classify\_all}}$. However, its fine-tuning results are slightly worse than those for $\text{Prompt}_{\text{classify\_all}}$.

The macro averages of *evaluation of the phrase extraction regarding the element type per requirement* (eval6) (c.f. Section 6.3.3) over all three runs are depicted in Table 6.7. There is a clear improvement from prompt engineering to fine-tuning. In contrast to the best prompt engineering approach (F1 - $\text{Prompt}_{\text{identify\_all}}$ without system mode: 0.75), fine-tuning achieves 17% points better results in its best (F1 - $\text{Prompt}_{\text{classify\_all}}$ with system mode: 0.92) and 2% points in its worst approach (F1 - $\text{Prompt}_{\text{classify\_all}}$ without system

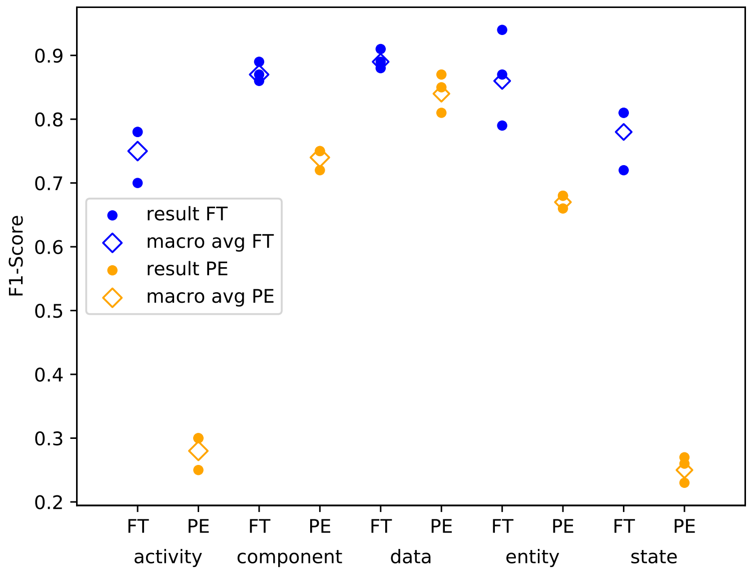


Figure 6.3.: Evaluation of the phrase extraction regarding the element type: Comparison of fine-tuning (FT) and prompt engineering (PE) for all element types for all three runs

mode: 0.77). In prompt engineering, Prompt$_{identify\_all}$ is still better than Prompt$_{classify\_all}$. Both prompts achieve noticeably better results in prompt engineering without using the system mode. Overall, the results of eval6 should be treated with caution. Only those requirements where all element types have been correctly identified are looked at to determine whether all indicative phrases are correct as well. Therefore, the sample sizes are relatively small. For prompt engineering, it ranges from 4 to 6 independently of the used prompt. For fine-tuning it ranges from 8 to 14. In summary, GPT better extracts phrases per requirement for fine-tuning than for prompt engineering which was already indicated in the independent *evaluation of the phrase extraction* (eval3) Table 6.4. However, the results of eval6 are less generalizable and should not be given too much weight due to the low sample size compared to the number of the requirements in the gold standard.

6.4. Discussion and Threats to Validity

This section summarizes and discusses the results of the six evaluation scenarios before examining possible threats to validity for this chapter. *Evaluation of the element type classification* (eval1) (c.f. Section 6.3.5.1) shows GPT's ability to correctly classify most of the element types for prompt engineering (F1-score: 0.67-0.7) as well as fine-tuning (F1-score: 0.7-0.73). The results of the element types *component*, *data*, and *state* reveal equally good or better values for fine-tuning compared to prompt engineering. For *entity*, it depends on the prompt used whether the prompt engineering results are slightly better

		Precision	Recall	F1-score	Sample Size
$\text{Prompt}_{\text{identify\_all}}$	**Prompt Engineering**				
	without system mode	0.58	0.58	0.58	400, 401, 404
	with system mode	0.61	0.61	0.61	401, 397, 401
	Fine-tuning				
	without system mode	0.84	0.84	0.84	387, 408, 380
	with system mode	0.85	0.85	0.85	371, 397, 383
$\text{Prompt}_{\text{classify\_all}}$	**Prompt Engineering**				
	without system mode	0.52	0.52	0.52	366, 364, 358
	with system mode	0.55	0.55	0.55	370, 365, 354
	Fine-tuning				
	without system mode	0.86	0.86	0.86	388, 369, 370
	with system mode	0.87	0.87	0.87	352, 366, 356

Table 6.6.: Evaluation of the phrase extraction regarding the element type: Macro micro averages over all element types, Prompt engineering uses GPT version 4, Fine-tuning uses GPT version 3.5 turbo

or the fine-tuning results. However, the element type *activity* achieves slightly better results for prompt engineering than for fine-tuning. The results of the *evaluation of the phrase extraction* (eval3) (c.f. Section 6.3.5.2) are slightly worse than eval1 but still acceptable, at least for fine-tuning (F1-score: 0.68-0.7). The *evaluation of the element type classification per requirement* (eval2) (c.f. Section 6.3.5.1) and the *evaluation of the phrase extraction per requirement* (eval4) (c.f. Section 6.3.5.2) show that GPT has difficulties to correctly classify all element types or phrases for one requirement. Prompt engineering seems to be inappropriate to classify element types (F1-score: 0.07-0.09) or extract phrases (F1-score: 0.03-0.06) per requirement. As eval2 and eval4 are multi-label classifications, the results for fine-tuning are acceptable, both for the classification (F1-score: 0.18-0.2) and the extraction (F1-score: 0.15-0.18) task. The results of eval2 are slightly better, which can be explained by the fact that element types are recognized better on average than phrases. This can be seen by comparing the results of *evaluation of the element type classification* (c.f. Section 6.3.5.1) and *evaluation of the phrase extraction* (c.f. Section 6.3.5.2). The *evaluation of the phrase extraction regarding the element type* (eval5) (c.f. Section 6.3.5.3) shows that GPT mostly correctly extracts the indicative phrase if the corresponding element type is correct for fine-tuning (F1-score: 0.84-0.87) while prompt engineering performs worse (F1-score: 0.52-0.61). However, differences in phrase extraction can be found between the respective element types. *Component*, *data*, and *entity* achieve good results for both prompt engineering and fine-tuning, whereas for the element types *activity* and *state*, the results of prompt engineering (activity: F1-score: 0.26-0.28, state: F1-score: 0.19-0.25) and fine-tuning (activity: F1-score: 0.75-0.8, state: F1-score: 0.72-0.82) differ noticeably by around 50% points. It can be observed that although the element type classification delivers good results, the phrases of *activity* and *state* are often not correctly extracted. That also puts the quite good prompt engineering results for *activity* in eval1 (c.f. Section 6.3.5.1)

		Precision	**Recall**	**F1-score**	**Sample Size**
Prompt$_{\text{identify\_all}}$	**Prompt Engineering**				
	without system mode	0.75	0.75	0.75	4, 4, 4
	with system mode	0.49	0.49	0.49	6, 4, 5
	Fine-tuning				
	without system mode	0.87	0.87	0.87	11, 12, 11
	with system mode	0.86	0.86	0.86	14, 12, 8
Prompt$_{\text{classify\_all}}$	**Prompt Engineering**				
	without system mode	0.62	0.62	0.62	6, 5, 4
	with system mode	0.34	0.34	0.34	6, 5, 4
	Fine-tuning				
	without system mode	0.77	0.77	0.77	9, 13, 9
	with system mode	0.92	0.92	0.92	11, 13, 9

Table 6.7.: Evaluation of the phrase extraction regarding the element type per requirement: Macro averages of precision, recall, and F1-score, Prompt engineering uses GPT version 4, Fine-tuning uses GPT version 3.5 turbo

into perspective. The good results of *evaluation of the phrase extraction regarding the element type per requirement* (eval6) (c.f. Section 6.3.5.3) should be treated with caution as they are calculated on a low sample size and are therefore not generalizable. Prompt engineering shows major differences between the version with system mode (F1-score: 0.34-0.49) and without (F1-score: 0.62-0.75). This can be explained by the fact that a false positive or a false negative result has a greater influence due to the low sample size. The results for fine-tuning for both versions are noticeably better (F1-score: 0.77-0.92). The sample size is bigger for fine-tuning which also explains the noticeably better fine-tuning results of eval2.

Another observation are strong variations in the results of the different fine-tuned models of a cross-validation. These occured in all evaluation scenarios, in some more strongly, in others less. They ranged from 7 to 11% points for eval3 to over 50% for eval6. This questions the stability of the fine-tuning results. An example of the variations in all three cross-validations is shown in Figure 6.4 for eval1 for the element type *activity* and Prompt$_{\text{identify\_all}}$ without system mode.

Common differences between the GPT results in prompt engineering and fine-tuning and the gold standard are discussed in the following. The deviations in phrase extraction affected primarily eval3, eval4, eval5, and eval6, while the element type classification influenced eval1, eval2, eval5, and eval6. For prompt engineering, GPT often outputs much longer phrases than expected. Thereby, it is not clear, if GPT really recognized the element type or just chose a phrase. Sometimes, GPT classifies an element type but with the wrong phrase. One reason for this could be that the gold standard is too specific, that the definition of the element type is not clear, or that the common understanding overwrites the specific definition. This is sometimes disadvantageous for the evaluation results of

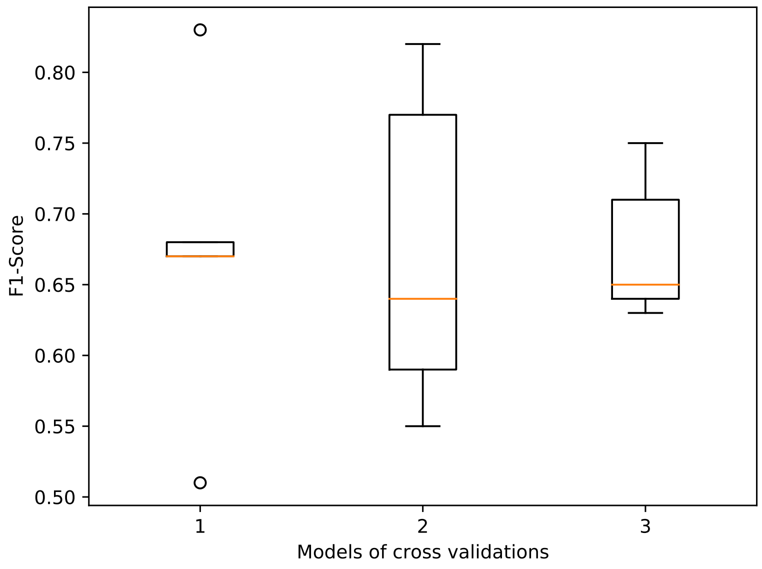


Figure 6.4.: Evaluation of the element type classification: F1-score variations in the cross-validations for the element type activity for Prompt$_{\text{identify\_all}}$ without system mode

entity. “EVerest system” is also often wrongly classified as *component* or *entity*. However, the system as such is not part of any class as it is not an element type of itself according to SecLan [57]. GPT also does not recognize well multiple *components* or *activities* in a listing, e.g., “OCPP, EV communication (EvseV2G), and API”. Another mistake of GPT was that in some cases, OCPP was recognized as a *component*, even though the requirement was about the standard. The differentiation between the *component* OCPP and the standard is generally difficult due to the same naming. The results for fine-tuning show other deviations of the GPT results from the gold standard. For fine-tuning, the phrases are much shorter because the minimal term of the gold standard is used to train the models. Moreover, GPT usually does not name phrases twice, even if they appear twice in the requirement and therefore in the gold standard. It is also possible that the gold standard does not always contain the correct phrases or does not have all elements, which affects the evaluation results. It stands out that especially *activities* are misclassified. The gold standard is relatively strict and does not include verbs like “support”, “implement”, and “comply” which are extracted by GPT. The element type *state* is sometimes not found, perhaps, because it occurs only 49 times in the gold standard.

Several threats affect the validity of the element type classification and phrase extraction. As described in Section 5.8, there are several types of validity that can be affected: internal, construct, and external validity.

Internal validity is threatened by the fact that the prompts that are used for prompt engineering and fine-tuning were previously tested on a part of the dataset. This was part of the process to improve the prompts. The evaluation results could have been influenced by re-using the prompts. However, the fact that only a small set of requirements were chosen for testing mitigates this risk. For 5-fold cross-validation, the dataset needs to be divided into equally sized folds. As the dataset contains 93 requirements, this is not possible. Instead, the last fold contains three requirements more than the other four. For the first four models, 68 requirements are used for training and 18 for evaluation. The last model is trained on 65 requirements and evaluated with 21 requirements. This could influence the results of the models. To mitigate that threat, the weighted average was calculated over the five models of a cross-validation set to take into account the different sizes of the evaluation sets. However, the difference in size seems to be neglectabe as the weighted average and macro average differ only by 1 to 2% points in all evaluation scenarios. Another possible threat to internal validity is that all coarse-grained requirements were specified twice. This leads to a certain similarity between any two requirements that originate from the same coarse-grained requirement and could affect the fine-tuning results, as these requirements can be split into training and evaluation set. The threat is mitigated by a manual comparison of some of these requirements which reveals that the corresponding requirements are sufficiently different. In addition, in some cases different requirements were not specified in an interview, which is why not all requirements have an equivalent of the same origin.

The gold standard was created only by the author of this thesis, therefore, the risk of misclassification is higher. As the gold standard is used to compare with the GPT results and assess GPT's quality, this poses a major threat to construct validity. That risk is reduced by the transparent description of the gold standard creation and the listing of the errors that occurred during the evaluation. To mitigate this threat it would have been better to create the gold standard with an inter-annotator agreement. However, this was not possible within the scope of this thesis but is intended for further research in this area. Another threat to construct validity is that GPT is non-deterministic. Therefore, its answers to the same question differ slightly. This threat is mitigated by carrying out all evaluations three times and calculating an average value.

In this thesis, only the elicited dataset described in Section 5.7 is used, which poses a threat to external validity. The creation of the dataset itself raises threats to construct validity as discussed in Section 5.8. The dataset is relatively small and unbalanced as the most frequent requirements of the dataset belong to the security category confidentiality. Furthermore, the representativeness of the dataset can be questioned. This risk could be mitigated by using a second dataset for the evaluation of GPT's performance. That is beyond the scope of this thesis. External validity is also threatened by evaluating only a limited number of prompts. There may be better prompts for prompt engineering and fine-tuning that were never tried. The selection of the prompts limits the generalizability of the evaluation results as different prompts could provide other results. However, it is impossible to test all prompt combinations to make a more general statement.

7. Conclusion and Future Work

Verification of security requirements is important. Specifically, with the spreading of electric vehicles, the security requirements of charging stations must be ensured and thus be verified. In this thesis, security requirements for the open-source software EVerest, a framework for electric vehicle charging stations, were elicited and classified. For the elicitation, a questionnaire was first created that covers the security categories of confidentiality, integrity, availability, and authentication. The questionnaire was sent to stakeholders of EVerest. Thereby, 57 security-related requirements were retrieved from seven stakeholders. For traceability to architectural elements of EVerest, the requirements were refined in four interviews with software developers of EVerest. To establish a common understanding of EVerest in the interviews, a component model of EVerest was derived from the publicly available code of EVerest and given to the interviewees. It should help the interviewees formulate the requirements at the design level and also explicitly name components or interfaces. In the end, 93 specified requirements were collected during the interviews.

To assist the requirements verification process, trace links can be created between the requirement and the architecture. A classification into system model elements that can be mapped to elements of the system architecture marks a first step in this direction. Therefore, the elicited security requirements were classified into the system model elements of SecLan[57]. In this thesis, the element types *activity*, *component*, *data*, *entity*, and *state* were classified by using the large language model GPT. Zero-shot prompt engineering on GPT 4 was compared with fine-tunbing on GPT 3.5 turbo. A gold standard was created by the author of this thesis for the 93 requirements. Six different evaluations were chosen. They evaluate GPT's performance concerning the identification of element types, extraction of indicative phrases, and the extraction of the phrase if the element type is correct. These goals were examined both for each pair of element type and indicative phrase as well as on the requirement level. The evaluation results show that fine-tuning performs significantly better than prompt engineering for almost every evaluation. For fine-tuning, the phrase extraction achieves an F1-score of 0.68. With a preliminary check that the element type is correct, an F1-score of 0.87 is even reached for the phrase extraction. Moreover, it can be observed that GPT has difficulties in identifying all element types or phrases correctly per requirement both for prompt engineering and fine-tuning, although the results of the latter are better. For the classification of the element types on the requirement level, prompt engineering only achieves an F1-score of 0.09, whereas fine-tuning reaches up to 0.2. Common mistakes in classification that are made by GPT were too long phrases in prompt engineering and especially for the element type activity too many false positives in fine-tuning.

As a future work, the requirements could be further refined to the code level, so that a mapping, e.g., to the element type *entity* for classes, is possible. To increase the reliability of the results, the gold standard should be improved by using an inter-annotator agreement. Moreover, the classification approach of this thesis should be tested on multiple datasets to be able to make a more generalizable statement of GPT's performance. A two-step classification approach could be tested as well. The division of the identification and extraction tasks into two separate steps could improve the results and limit the false positive results. In the first step, GPT could be asked whether or not an element type is present in the requirement. If GPT found the element type, it could then be asked in a second step to extract the corresponding phrase out of the requirement. Furthermore, different prompts could be tested to improve the classification and extraction results. GPT is one of the largest LLMs, but also smaller models can be tested for prompt engineering and specifically for fine-tuning, as smaller models adapt faster to less training data. An overall aim is the full automation of the requirements verification process. If the indicative phrase and element type would be already correctly identified, a trace link could then be created from the requirement to the architecture element. Therefore, this thesis lays a foundation for further research in the direction of automatic requirement verification.

Bibliography

[1] *5. EVerest Module Configurations — EVerest Documentation.* URL: https://everest.github.io/module-net/general/05_existing_modules.html (visited on 12/05/2023).

[2] Waad Alhoshan, Alessio Ferrari, and Liping Zhao. "Zero-shot learning for requirements classification: An exploratory study". In: *Information and Software Technology* 159 (July 2023), p. 107202. ISSN: 09505849. DOI: 10.1016/j.infsof.2023.107202. URL: https://linkinghub.elsevier.com/retrieve/pii/S0950584923000563 (visited on 11/09/2023).

[3] Claire Ballinger and Christine Davey. "Designing a Questionnaire: An Overview". In: *British Journal of Occupational Therapy* 61.12 (Dec. 1998), pp. 547–550. ISSN: 0308-0226, 1477-6006. DOI: 10.1177/030802269806101204. URL: http://journals.sagepub.com/doi/10.1177/030802269806101204 (visited on 11/15/2023).

[4] Muneera Bano et al. "Teaching requirements elicitation interviews: an empirical study of learning from mistakes". In: *Requirements Engineering* 24.3 (Sept. 1, 2019). Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 3 Publisher: Springer London, pp. 259–289. ISSN: 1432-010X. DOI: 10.1007/s00766-019-00313-0. URL: https://link.springer.com/article/10.1007/s00766-019-00313-0 (visited on 10/25/2023).

[5] Amanda Bolderston. "Conducting a Research Interview". In: *Journal of Medical Imaging and Radiation Sciences* 43.1 (Mar. 2012), pp. 66–76. ISSN: 19398654. DOI: 10.1016/j.jmir.2011.12.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S1939865411001329 (visited on 01/08/2024).

[6] Jaspreet Kaur Boparai, Surjit Singh, and Priyanka Kathuria. "How to Design and Validate A Questionnaire: A Guide". In: *Current Clinical Pharmacology* 13.4 (Jan. 14, 2019), pp. 210–215. ISSN: 15748847. DOI: 10.2174/1574884713666180807151328. URL: http://www.eurekaselect.com/164433/article (visited on 11/16/2023).

[7] Petra M Boynton and Trisha Greenhalgh. "Selecting, designing, and developing your questionnaire". In: *BMJ* 328.7451 (May 29, 2004), pp. 1312–1315. ISSN: 0959-8138, 1468-5833. DOI: 10.1136/bmj.328.7451.1312. URL: https://www.bmj.com/lookup/doi/10.1136/bmj.328.7451.1312 (visited on 10/19/2023).

[8] Tom B. Brown et al. *Language Models are Few-Shot Learners.* 2020.

[9] *CardSystems Solutions, Inc., and Solidus Networks, Inc., d/b/a Pay By Touch Solutions, In the Matter Of.* Federal Trade Commission. Feb. 23, 2006. URL: https://www.ftc.gov/legal-library/browse/cases-proceedings/052-3148-cardsystems-solutions-inc-solidus-networks-inc-dba-pay-touch-solutions-matter (visited on 11/15/2023).

[10] Yupeng Chang et al. *A Survey on Evaluation of Large Language Models.* Oct. 17, 2023. DOI: 10.48550/arXiv.2307.03109. arXiv: 2307.03109[cs]. URL: http://arxiv.org/abs/2307.03109 (visited on 11/04/2023).

[11] K. R. Chowdhary. "Natural Language Processing". In: *Fundamentals of Artificial Intelligence.* Ed. by K.R. Chowdhary. New Delhi: Springer India, 2020, pp. 603–649. ISBN: 978-81-322-3972-7. DOI: 10.1007/978-81-322-3972-7_19. URL: https://doi.org/10.1007/978-81-322-3972-7_19 (visited on 11/02/2023).

[12] *Communication Protocols.* Vector Informatik GmbH. URL: https://www.vector.com/int/en/know-how/smart-charging/communication-protocols/ (visited on 01/10/2024).

[13] Moises Danziger and Marcos Silva. *The Importance of Security Requirements Elicitation and How to Do It.* May 2015.

[14] Eric Dash. "Weakness in the Data Chain". In: *The New York Times. Business* (June 30, 2005). ISSN: 0362-4331. URL: https://www.nytimes.com/2005/06/30/technology/weakness-in-the-data-chain.html (visited on 11/15/2023).

[15] A. Davis et al. "Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review". In: *14th IEEE International Requirements Engineering Conference (RE'06).* 14th IEEE International Requirements Engineering Conference. Minneapolis/St. Paul, MN: IEEE, Sept. 2006, pp. 179–188. ISBN: 978-0-7695-2555-6. DOI: 10.1109/RE.2006.17. URL: http://ieeexplore.ieee.org/document/1704061/ (visited on 10/18/2023).

[16] M. José Escalona and Nora Koch. "REQUIREMENTS ENGINEERING FOR WEB APPLICATIONS – A COMPARATIVE STUDY". In: *Journal of Web Engineering* (2003), pp. 193–212. ISSN: 1544-5976. URL: https://journals.riverpublishers.com/index.php/JWE/ (visited on 10/18/2023).

[17] EVerest. *EV Charging Pioneers # 1 - How the EVerest Ecosystem will simplify Charging Use Cases.* 2022. URL: https://www.youtube.com/watch?v=OJ6kjHRPkyY (visited on 12/22/2023).

[18] *Everest-Core.* EVerest, Nov. 27, 2023. URL: https://github.com/EVerest/everest-core (visited on 12/05/2023).

[19] *EVerest/Everest-Framework.* EVerest. URL: https://github.com/EVerest/everest-framework (visited on 03/01/2024).

[20] *EVerest/Libevse-Security.* EVerest, Feb. 11, 2024. URL: https://github.com/EVerest/libevse-security (visited on 03/01/2024).

[21] *EVerest/Libocpp: C++ Implementation of the Open Charge Point Protocol.* EVerest. URL: https://github.com/EVerest/libocpp (visited on 03/01/2024).

[22] Donald Firesmith. "Engineering Security Requirements." In: *The Journal of Object Technology* 2.1 (2003), p. 53. ISSN: 1660-1769. DOI: 10.5381/jot.2003.2.1.c6. URL: http://www.jot.fm/contents/issue_2003_01/column6.html (visited on 10/13/2023).

[23] Donald Firesmith. “Security Use Cases.” In: *Journal of Object Technology* 2 (Jan. 1, 2003), pp. 53–64.

[24] Louie Giray. “Prompt Engineering with ChatGPT: A Guide for Academic Writers”. In: *Annals of Biomedical Engineering* 51.12 (Dec. 1, 2023), pp. 2629–2633. ISSN: 1573-9686. DOI: 10.1007/s10439-023-03272-4. URL: https://doi.org/10.1007/s10439-023-03272-4 (visited on 01/17/2024).

[25] *GitHub - EVerest/EVerest: Main Repository of EVerest - an EV Charging Software Stack. All Main Documentations and Issues Are Stored Here.* EVerest. URL: https://github.com/EVerest/EVerest (visited on 03/01/2024).

[26] Dirk Grossmann and Dr. Heiner Hild. *Smart Charging – A Key to Successful E-Mobility.* 2014.

[27] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. “Semantically Enhanced Software Traceability Using Deep Learning Techniques”. In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE).* May 2017, pp. 3–14. DOI: 10.1109/ICSE.2017.9. arXiv: 1804.02438 [cs]. URL: http://arxiv.org/abs/1804.02438 (visited on 04/04/2024).

[28] Muhammad Usman Hadi et al. *A Survey on Large Language Models: Applications, Challenges, Limitations, and Practical Usage.* preprint. July 10, 2023. DOI: 10.36227/techrxiv.23589741.v1. URL: https://www.techrxiv.org/articles/preprint/A_Survey_on_Large_Language_Models_Applications_Challenges_Limitations_and_Practical_Usage/23589741/1 (visited on 11/11/2023).

[29] C.B. Haley et al. “Security Requirements Engineering: A Framework for Representation and Analysis”. In: *IEEE Transactions on Software Engineering* 34.1 (Jan. 2008), pp. 133–153. ISSN: 0098-5589. DOI: 10.1109/TSE.2007.70754. URL: http://ieeexplore.ieee.org/document/4359475/ (visited on 11/09/2023).

[30] Ridong Han et al. *Is Information Extraction Solved by ChatGPT? An Analysis of Performance, Evaluation Criteria, Robustness and Errors.* May 23, 2023. DOI: 10.48550/arXiv.2305.14450. arXiv: 2305.14450 [cs]. URL: http://arxiv.org/abs/2305.14450 (visited on 04/08/2024). preprint.

[31] Md. Ariful Haque, Md. Abdur Rahman, and Md Saeed Siddik. “Non-Functional Requirements Classification with Feature Extraction and Machine Learning: An Empirical Study”. In: *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT).* 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT). Dhaka, Bangladesh: IEEE, May 2019, pp. 1–5. ISBN: 978-1-72813-445-1. DOI: 10.1109/ICASERT.2019.8934499. URL: https://ieeexplore.ieee.org/document/8934499/ (visited on 10/24/2023).

[32] Xinyi Hou et al. *Large Language Models for Software Engineering: A Systematic Literature Review.* Sept. 12, 2023. DOI: 10.48550/arXiv.2308.10620. arXiv: 2308.10620[cs]. URL: http://arxiv.org/abs/2308.10620 (visited on 11/02/2023).

[33] S.E. Hove and B. Anda. “Experiences from Conducting Semi-Structured Interviews in Empirical Software Engineering Research”. In: *11th IEEE International Software Metrics Symposium (METRICS’05)*. 11th IEEE International Software Metrics Symposium (METRICS’05). Sept. 2005, 10 pp.–23. DOI: 10.1109/METRICS.2005.24. URL: https://ieeexplore.ieee.org/abstract/document/1509301 (visited on 01/17/2024).

[34] Elizabeth Hull, Ken Jackson, and Jeremy Dick. *Requirements Engineering*. London: Springer, 2011. DOI: 10.1007/978-1-84996-405-0. URL: http://link.springer.com/10.1007/978-1-84996-405-0 (visited on 10/20/2023).

[35] “IEEE Standard Glossary of Software Engineering Terminology”. In: *IEEE Std 610.12-1990* (Dec. 1990), pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064. URL: https://ieeexplore.ieee.org/document/159342 (visited on 11/16/2023).

[36] Tabbassum Iqbal and Mohammad Suaib. “Requirement Elicitation Technique: - A Review Paper”. In: *International Journal of Computer and Mathematical Sciences* 3 (Dec. 2014).

[37] *ISO/IEC 27000:2018*. ISO. May 4, 2022. URL: https://www.iso.org/standard/73906.html (visited on 12/06/2023).

[38] “ISO/IEC/IEEE International Standard - Systems and Software Engineering – Life Cycle Processes – Requirements Engineering”. In: *ISO/IEC/IEEE 29148:2018(E)* (Nov. 2018), pp. 1–104. DOI: 10.1109/IEEESTD.2018.8559686. URL: https://ieeexplore.ieee.org/document/8559686 (visited on 12/08/2023).

[39] Stacy Jacob and S. Furgerson. “Writing Interview Protocols and Conducting Interviews: Tips for Students New to the Field of Qualitative Research”. In: *The Qualitative Report* (Jan. 20, 2015). ISSN: 2160-3715, 1052-0147. DOI: 10.46743/2160-3715/2012.1718. URL: https://nsuworks.nova.edu/tqr/vol17/iss42/3/ (visited on 01/06/2024).

[40] Rajni Jindal, Ruchika Malhotra, and Abha Jain. “Automated classification of security requirements”. In: *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI). Jaipur, India: IEEE, Sept. 2016, pp. 2027–2033. ISBN: 978-1-5090-2029-4. DOI: 10.1109/ICACCI.2016.7732349. URL: http://ieeexplore.ieee.org/document/7732349/ (visited on 10/11/2023).

[41] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, Pearson Education International, 2009.

[42] Eleanor Knott et al. “Interviews in the Social Sciences”. In: *Nature Reviews Methods Primers* 2.1 (1 Sept. 15, 2022), pp. 1–15. ISSN: 2662-8449. DOI: 10.1038/s43586-022-00150-6. URL: https://www.nature.com/articles/s43586-022-00150-6 (visited on 01/24/2024).

[43] Heiko Koziolek et al. “Evaluating performance of software architecture models with the Palladio component model”. In: *Software Applications: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2009, pp. 1111–1134.

[44] Raymond S. T. Lee. *Natural Language Processing: A Textbook with Python Implementation.* Singapore: Springer Nature Singapore, 2024. ISBN: 978-981-9919-98-7 978-981-9919-99-4. DOI: 10.1007/978-981-99-1999-4. URL: https://link.springer.com/10.1007/978-981-99-1999-4 (visited on 05/01/2024).

[45] Soo Ling Lim and A. Finkelstein. “StakeRare: Using Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation”. In: *Software Engineering, IEEE Transactions on* 38 (May 1, 2012), pp. 1–1. DOI: 10.1109/TSE.2011.36.

[46] *LimeSurvey Manual.* URL: https://manual.limesurvey.org/LimeSurvey_Manual (visited on 03/13/2024).

[47] Pengfei Liu et al. *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing.* July 28, 2021. DOI: 10.48550/arXiv.2107.13586. arXiv: 2107.13586[cs]. URL: http://arxiv.org/abs/2107.13586 (visited on 11/04/2023).

[48] Gill Marshall. “The purpose, design and administration of a questionnaire for data collection”. In: *Radiography* 11.2 (May 2005), pp. 131–136. ISSN: 10788174. DOI: 10.1016/j.radi.2004.09.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S1078817404001208 (visited on 11/11/2023).

[49] Pablo N Mendes et al. “Evaluating the Impact of Phrase Recognition on Concept Tagging”. In: *European Language Resources Association (ELRA)* (2012).

[50] Roxanne E. Miller. *The Quest for Software Requirements.* Oconomowoc, WI, USA: MavenMark Books, 2009. ISBN: 1595980679.

[51] J.M. Moore and F.M. Shipman. “A comparison of questionnaire-based and GUI-based requirements gathering”. In: *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering.* Proceedings of ASE 2000 15th IEEE International Automated Software Engineering Conference. Grenoble, France: IEEE, 2000, pp. 35–43. ISBN: 978-0-7695-0710-1. DOI: 10.1109/ASE.2000.873648. URL: http://ieeexplore.ieee.org/document/873648/ (visited on 10/18/2023).

[52] *MQTT - The Standard for IoT Messaging.* URL: https://mqtt.org/ (visited on 01/12/2024).

[53] Tony Nasr et al. “Power Jacking Your Station: In-depth Security Analysis of Electric Vehicle Charging Station Management Systems”. In: *Computers & Security* 112 (Jan. 1, 2022), p. 102511. ISSN: 0167-4048. DOI: 10.1016/j.cose.2021.102511. URL: https://www.sciencedirect.com/science/article/pii/S0167404821003357 (visited on 11/08/2023).

[54] OpenAI et al. *GPT-4 Technical Report.* Dec. 18, 2023. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs]. URL: http://arxiv.org/abs/2303.08774 (visited on 01/19/2024). preprint.

[55] A. N. Oppenheim. “Questionnaire design, interviewing and attitude measurement, New ed.” In: 1992. URL: https://api.semanticscholar.org/CorpusID:221992851.

[56] Carla Pacheco, Ivan García, and Miryam Reyes. “Requirements elicitation techniques: a systematic literature review based on the maturity of the techniques”. In: *IET Software* 12.4 (2018). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/iet-sen.2017.0144, pp. 365–378. ISSN: 1751-8814. DOI: 10.1049/iet-sen.2017.0144. URL: https://onlinelibrary.wiley.com/doi/abs/10.1049/iet-sen.2017.0144 (visited on 10/25/2023).

[57] Sven Peldszus et al. *Coupling Design-time Security Models with Implementation-level Security Checks.* to be published.

[58] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques.* 1st. Springer Publishing Company, Incorporated, 2010. ISBN: 3642125778.

[59] Zoya Pourmirza and Sara Walker. “Electric Vehicle Charging Station: Cyber Security Challenges and Perspective”. In: *2021 IEEE 9th International Conference on Smart Energy Grid Engineering (SEGE).* 2021 IEEE 9th International Conference on Smart Energy Grid Engineering (SEGE). Aug. 2021, pp. 111–116. DOI: 10.1109/SEGE52446.2021.9535052. URL: https://ieeexplore.ieee.org/abstract/document/9535052 (visited on 11/04/2023).

[60] Jennifer Rowley. “Conducting Research Interviews”. In: *Management Research Review* 35.3/4 (Mar. 23, 2012), pp. 260–271. ISSN: 2040-8269. DOI: 10.1108/01409171211210154. URL: https://www.emerald.com/insight/content/doi/10.1108/01409171211210154/full/html (visited on 01/06/2024).

[61] Oscar Sainz et al. *GoLLIE: Annotation Guidelines Improve Zero-Shot Information-Extraction.* Mar. 6, 2024. DOI: 10.48550/arXiv.2310.03668. arXiv: 2310.03668 [cs]. URL: http://arxiv.org/abs/2310.03668 (visited on 04/15/2024). preprint.

[62] Sunita Sarawagi. “Information Extraction”. In: *Foundations and Trends in Databases* 1.3 (Mar. 1, 2008), pp. 261–377. ISSN: 1931-7883. DOI: 10.1561/1900000003. URL: https://doi.org/10.1561/1900000003 (visited on 05/01/2024).

[63] Guttorm Sindre and Andreas L. Opdahl. “Eliciting Security Requirements with Misuse Cases”. In: *Requirements Engineering* 10.1 (Jan. 1, 2005), pp. 34–44. ISSN: 1432-010X. DOI: 10.1007/s00766-004-0194-4. URL: https://doi.org/10.1007/s00766-004-0194-4 (visited on 05/09/2020).

[64] Sonit Singh. *Natural Language Processing for Information Extraction.* July 6, 2018. DOI: 10.48550/arXiv.1807.02383. arXiv: 1807.02383 [cs]. URL: http://arxiv.org/abs/1807.02383 (visited on 04/27/2024). preprint.

[65] John Slankas and Laurie Williams. “Automated extraction of non-functional requirements in available documentation”. In: *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE).* 2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE). San Francisco, CA, USA: IEEE, May 2013, pp. 9–16. ISBN: 978-1-4673-6271-9. DOI: 10.1109/NAturaLiSE.2013.6611715. URL: http://ieeexplore.ieee.org/document/6611715/ (visited on 11/08/2023).

[66] Hamed Taherdoost. *International Journal of Academic Research in Management.* 2022.

[67] Vasily Varenov and Aydar Gabdrahmanov. “Security Requirements Classification into Groups Using NLP Transformers”. In: *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*. 2021 IEEE 29th International Requirements Engineering Conference Workshops (REW). Notre Dame, IN, USA: IEEE, Sept. 2021, pp. 444–450. ISBN: 978-1-66541-898-0. DOI: 10.1109/REW53955.2021.9714713. URL: https://ieeexplore.ieee.org/document/9714713/ (visited on 11/09/2023).

[68] Elizabeth M Walker. “Questionnaire Design in Practice”. In: *British Journal of Therapy and Rehabilitation* 3.4 (Apr. 1996), pp. 229–233. ISSN: 1354-8581, 2059-9331. DOI: 10.12968/bjtr.1996.3.4.14847. URL: http://www.magonlinelibrary.com/doi/10.12968/bjtr.1996.3.4.14847 (visited on 11/17/2023).

[69] Xiang Wei et al. *Zero-Shot Information Extraction via Chatting with ChatGPT*. Feb. 20, 2023. DOI: 10.48550/arXiv.2302.10205. arXiv: 2302.10205 [cs]. URL: http://arxiv.org/abs/2302.10205 (visited on 05/01/2024). preprint.

[70] *What Is EVerest — EVerest documentation*. URL: https://everest.github.io/nightly/ (visited on 11/02/2023).

[71] Jules White et al. *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*. Feb. 21, 2023. DOI: 10.48550/arXiv.2302.11382. arXiv: 2302.11382[cs]. URL: http://arxiv.org/abs/2302.11382 (visited on 11/02/2023).

[72] Wikipedia contributors. *Open Charge Point Protocol — Wikipedia, The Free Encyclopedia*. [Online; accessed 10-January-2024]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Open_Charge_Point_Protocol&oldid=1185865637.

[73] Jonas Winkler and Andreas Vogelsang. “Automatic Classification of Requirements Based on Convolutional Neural Networks”. In: *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW). Beijing, China: IEEE, Sept. 2016, pp. 39–45. ISBN: 978-1-5090-3694-3. DOI: 10.1109/REW.2016.021. URL: http://ieeexplore.ieee.org/document/7815604/ (visited on 10/11/2023).

[74] Masooma Yousuf and Mohammad Asger. “Comparison of Various Requirements Elicitation Techniques”. In: *International Journal of Computer Applications* 116.4 (Apr. 22, 2015), pp. 8–15. ISSN: 09758887. DOI: 10.5120/20322-2408. URL: http://research.ijcaonline.org/volume116/number4/pxc3902408.pdf (visited on 10/18/2023).

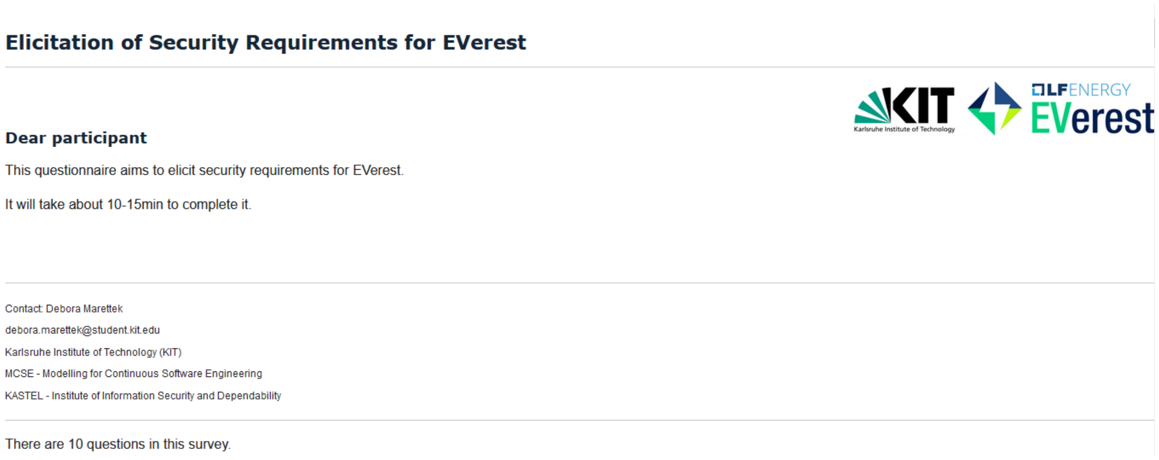
[75] Jianzhang Zhang et al. *Empirical Evaluation of ChatGPT on Requirements Information Retrieval Under Zero-Shot Setting*. July 19, 2023. DOI: 10.48550/arXiv.2304.12562. arXiv: 2304.12562 [cs]. URL: http://arxiv.org/abs/2304.12562 (visited on 04/03/2024). preprint.

[76] Yutao Zhu et al. *Large Language Models for Information Retrieval: A Survey*. Jan. 19, 2024. DOI: 10.48550/arXiv.2308.07107. arXiv: 2308.07107 [cs]. URL: http://arxiv.org/abs/2308.07107 (visited on 01/26/2024). preprint.

[77] Didar Zowghi and Chad Coulin. “Requirements Elicitation: A Survey of Technique, Approaches and Tools”. In: Springer, Jan. 1, 2005, 19–46 %U http://dx.doi.org/10.1007/3. ISBN: 978-3-540-25043-2. DOI: 10.1007/3-540-28244-0_2.

A. Appendix

A.1. Questionnaire



Elicitation of Security Requirements for EVerest

Dear participant

This questionnaire aims to elicit security requirements for EVerest.

It will take about 10-15min to complete it.

Contact: Debora Marettek
debora.marettek@student.kit.edu
Karlsruhe Institute of Technology (KIT)
MCSE - Modelling for Continuous Software Engineering
KASTEL - Institute of Information Security and Dependability

There are 10 questions in this survey.

Figure A.1.: Welcome screen of the survey

Dear participant

This questionnaire aims to elicit security requirements for EVerest.

It will take about 10-15min to complete it.

Contact: Debora Marettek debora.marettek@student.kit.edu Karlsruhe Institute of Technology (KIT) MCSE - Modelling for Continuous Software Engineering KASTEL - Institute of Information Security and Dependability

Section A: Introduction to survey

Structure: After a brief introduction to requirements elicitation, you will be asked about the security requirements of EVerest. Thus, the security categories confidentiality, integrity, availability, and authentication are presented to you. To each category, you are asked to write as many security requirements for EVerest as possible. When entering a requirement, more text fields will show up dynamically. Even though there are up to ten text fields available, you do not have to fill them all!

Definition:

In this questionnaire, we define "requirement" as follows:

A requirement describes the capabilities, the system should have to satisfy the needs of stakeholders. This includes e.g., considering how measures are to be implemented, what possible attacks should be avoided, and what data is critical. Example: *"The application shall not allow customer service agents to access the credit card information of customers."*

Writing Recommendations:

Please write the requirements in a complete sentence and keep in mind the following guidelines for writing good requirements. Requirements should be:

singular: write one requirement per sentence and keep your sentences short unambiguous: try to use active voice instead of passive voice unambiguous: avoid fuzzy words like effective, easy, quickly, may be, possibly, best, ... verifiable: it can be tested if the requirement is satisfied complete: the sentence contains all information to understand it

A1. I have read the text above

Yes ☐

No ☐

Section B: General

Before getting to the questions about the security requirements, some information about you helps to categorize the responses.

B1. What is your relation to EVerest?

I'm a developer at PIONIX ☐

I'm from a company that uses EVerest ☐

I'm from a company that wants to use EVerest ☐

None of the above ☐

B2. What is your interest in EVerest?

B3. **What is your profession?**

Software architect	☐
Software developer	☐
Requirements engineer	☐
Project manager	☐
Other	☐

Other

B4. **How confident are you regarding software security?**

	1	2	3	4	5
low | high	☐	☐	☐	☐	☐

Section C: Security Goal: Availability

Availability is the property of being accessible and usable on demand by an authorized entity.

This includes among others:

downtime impact on the business partial availability impact on the business transparent unavailability minimizing unavailability ...

An example availability requirement is:

"The user database shall be available 90% of the time."

ISO/IEC 27000:2018. Miller, Roxanne E.. "The Quest for Software Requirements." (2009).

C1. **What are security requirements for availability in EVerest?**

Up to nine additional text fields will appear dynamically. You don't have to fill them all.

Requirement 1

Requirement 2

Requirement 3

Requirement 4

Requirement 5

Requirement 6

Requirement 7	
Requirement 8	
Requirement 9	
Requirement 10	

Section D: Security Goal: Integrity

Integrity is the property of accuracy and completeness.

This includes the following aspects among others:

regular and consistent backups of the system's data prevent data loss backing up to other hard-drive tests for data restore procedures data authenticity and precision ...

An example integrity requirement is:

"The application shall prevent the unauthorized corruption of data collected from customers and other external users."

ISO/IEC 27000:2018. Miller, Roxanne E.. “The Quest for Software Requirements.” (2009).

D1. What are security requirements for integrity in EVerest?

Up to nine additional text fields will appear dynamically. You don't have to fill them all.

Requirement 1	
Requirement 2	
Requirement 3	
Requirement 4	
Requirement 5	
Requirement 6	
Requirement 7	
Requirement 8	
Requirement 9	
Requirement 10	

Section E: Security Goal: Confidentiality

Confidentiality is the property that information is not made available or disclosed to unauthorized individuals, entities, or processes.

This includes the following aspects among others:

data encryption methods user registration user authorization access control ...

An example confidentiality requirement is:

"The communication between the account management component and the payment management component shall be encrypted with AES-256."

ISO/IEC 27000:2018. Miller, Roxanne E.. “The Quest for Software Requirements.” (2009).

E1. What are security requirements for confidentiality in EVerest?

Up to nine additional text fields will appear dynamically. You don't have to fill them all.

Requirement 1	
Requirement 2	
Requirement 3	
Requirement 4	
Requirement 5	
Requirement 6	
Requirement 7	
Requirement 8	
Requirement 9	
Requirement 10	

LimeSurvey

Section F: Security Goals: Authentication

Authentication is the provision of assurance that a claimed characteristic of an entity is correct.

This includes the following aspects among others:

user registration user authentication user authorization password management ...

An example authentication requirement is:

"The application shall verify the identity of all of its client applications before allowing them to use its capabilities."

ISO/IEC 27000:2018. Miller, Roxanne E.. "The Quest for Software Requirements." (2009).

F1. What are security requirements for authentication and access control in EVerest?

Up to nine additional text fields will appear dynamically. You don't have to fill them all.

Requirement 1	
Requirement 2	
Requirement 3	
Requirement 4	
Requirement 5	
Requirement 6	
Requirement 7	
Requirement 8	
Requirement 9	
Requirement 10	

Section G: Other security requirements

In the previous questions, you were asked about security requirements of certain categories. If you are unsure about the categories of some requirements, or the categories didn't fit, please share them in the following.

G1. Please write down any other security requirements for EVerest that come to your mind.

Up to nine additional text fields will appear dynamically. You don't have to fill them all.

Requirement 1	
Requirement 2	

Requirement 3										
Requirement 4										
Requirement 5										
Requirement 6										
Requirement 7										
Requirement 8										
Requirement 9										
Requirement 10										

Thank you very much for your time and expertise!

If you are interested in the results, please send an e-mail to the address below.

Contact: Debora Marettek debora.marettek@student.kit.edu Karlsruhe Institute of Technology (KIT) MCSE - Modelling for Continuous Software Engineering KASTEL - Institute of Information Security and Dependability

A.2. Pilot study

Supplementary Material for Pilot study: Elicitation of Security Requirements for EVerest

Dear participant,

thank you for your participation in the pilot study for Elicitation of Security Requirements for EVerest!

The participants of the later study will be stakeholders of EVerest, a software for electric vehicle (EV) charging stations. Background knowledge of EVerest is necessary to be able to answer the questions of the study, therefore, EVerest is explained below using a scenario. The scenario, the architecture diagram and the subsequent feedback form are not part of the later study, and are only provided for the participants of the pilot study. Only the online questionnaire will be given to the participants of the later study.

Procedure of the pilot study:

1. Read the introduction and the scenario, and look at the architecture diagram
2. Start a timer to see how long the questionnaire takes
3. Answer the online questionnaire on LimeSurvey
4. Stop the timer
5. Fill in the feedback form at the end of the document

About EVerest:

EVerest is a software that provides an open-source full-stack implementation for electric vehicle (EV) charging stations. EVerest is a framework that offers customers the option to choose between modules or implement their own modules to configure the respective charging scenario.

EVerest is divided into different layers. At the bottom is the hardware layer, followed by a protocol layer for communication between the car and the charging station. The charge point is on layer 3. The associated API is located above the charge point, followed by the authentication layer. The energy management is located at the top.

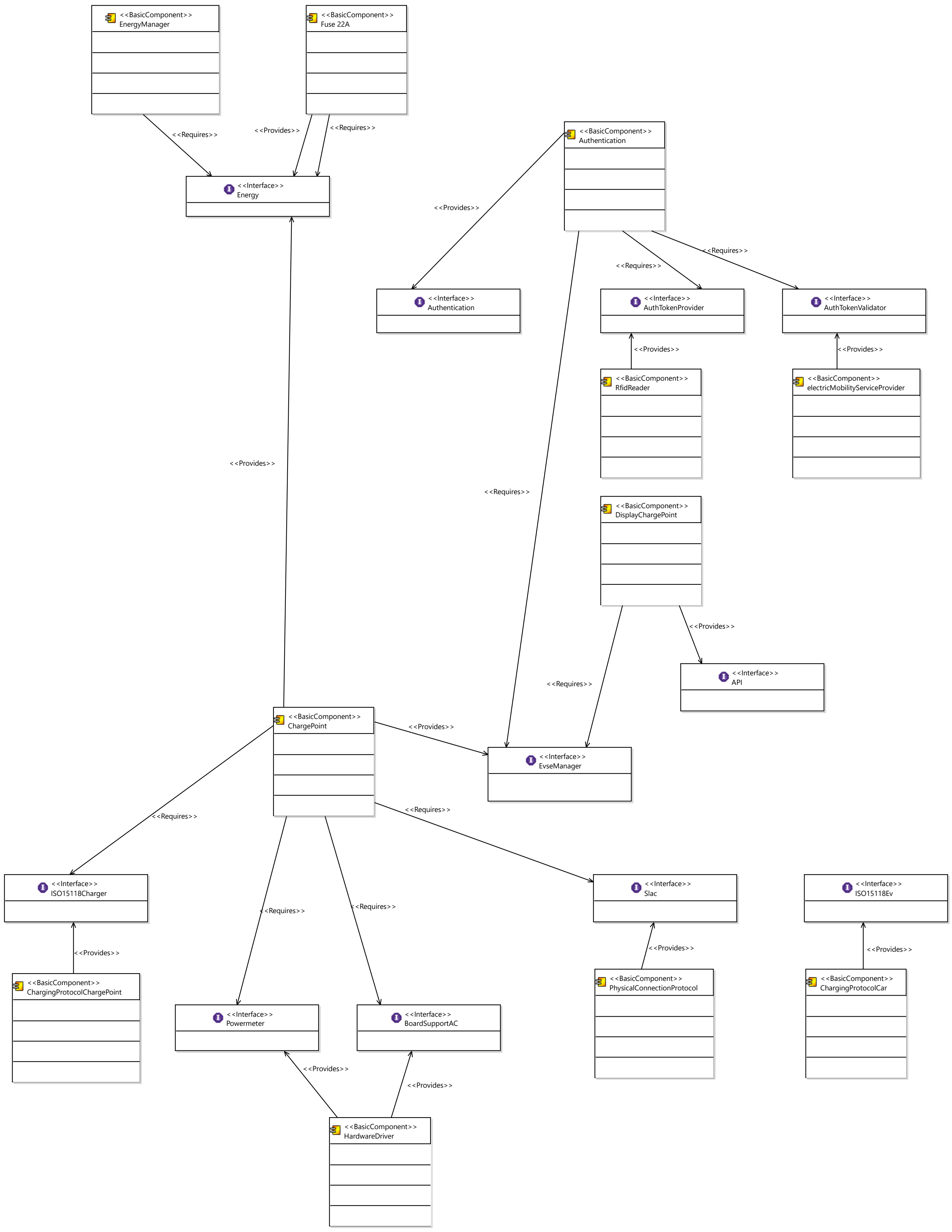
Scenario:

To be able to charge a car, the customer has to plug in his car and connect it to the charge point. The protocol that checks whether this connection exists is called SLAC. The *PhysicalConnectionProtocol* component with its interface is located at the right bottom of the diagram in layer 2. If both the car and the charging station implement their interfaces for the standard ISO15118, they can communicate over this protocol. ISO15118 defines the communication between the vehicle and the grid for bidirectional charging and provides also use cases like Plug & Charge. The components *ChargingProtocolChargePoint* and *ChargingProtocolCar* for ISO15118 as well as their interfaces both for the charger-side and the car-side are on the left and right side of layer 2.

The *EvseManager* interface manages one charge point. Hardware is required for the charge point to function properly. The *HardwareDriver* component is located at the bottom of the diagram and provides board support for AC charging as well as a powermeter to measure the energy flowing through the *ChargePoint*.

The display of the charging station shows information to the customer, like the amount of energy the car has already been charged with. The *DisplayChargePoint* component is located on layer 4, above the *ChargePoint* component on the right side.
To start the charging session, the customer has to authenticate himself with an RFID card. He swipes the card through the RFID reader that transmits the token to the *Authentication* component located at the upper right side of the diagram. The *Authentication* component sends the data to the electric mobility service provider where the RFID card is registered, who can check if the token is valid. If authentication is successful, the *Authentication* component authorizes the energy to flow through the *ChargePoint*. The power that charges the car comes from an energy node, e.g., a *Fuse with 22A* component that delivers the energy to the charge point. An *EnergyManager* can be used for load balancing if there are multiple charge points. Both energy components are located at the top left of the diagram.

The architecture diagram can be found on the next page.


<<BasicComponent>> EnergyManager
<<BasicComponent>> Fuse 22A
<<Requires>>
<<Provides>>
<<Requires>>
<<Interface>> Energy
<<BasicComponent>> Authentication
<<Provides>>
<<Requires>>
<<Requires>>
<<Interface>> Authentication
<<Interface>> AuthTokenProvider
<<Interface>> AuthTokenValidator
<<Provides>>
<<Provides>>
<<BasicComponent>> RfidReader
<<BasicComponent>> electricMobilityServiceProvider
<<Provides>>
<<Requires>>
<<BasicComponent>> DisplayChargePoint
<<Provides>>
<<Requires>>
<<Interface>> API
<<BasicComponent>> ChargePoint
<<Provides>>
<<Interface>> EvseManager
<<Requires>>
<<Requires>>
<<Interface>> ISO15118Charger
<<Requires>>
<<Requires>>
<<Interface>> Slac
<<Interface>> ISO15118Ev
<<Provides>>
<<Provides>>
<<Provides>>
<<BasicComponent>> ChargingProtocolChargePoint
<<Interface>> Powermeter
<<Interface>> BoardSupportAC
<<BasicComponent>> PhysicalConnectionProtocol
<<BasicComponent>> ChargingProtocolCar
<<Provides>>
<<Provides>>
<<BasicComponent>> HardwareDriver

Feedback on the questionnaire

Write down here the time you needed to answer the questionnaire (in minutes):

Please give also feedback on the questionnaire on how comprehensible and clear the instructions and questions are. The given instructions and questions were clear (please select):

○ Yes

○ No

○ Partially

If you selected *partially* or *no*, please specify, what was unclear:

If you have suggestions for improvement, please write it down here:

Thank you for your time and feedback on the questionnaire!

A.3. Interview Protocol

Introduction: (7-10min)

- Greeting
- Hints in case of technical issues
- Short presentation of interviewer and research (security requirements elicitation)
- Informed consent

Initial questions:
- *Would you describe yourself as a software developer?*
- *How long have you been working for PIONIX/Chargebyte?* (in months/years)

Explanation of the structure and notation of EVerest component diagram (PCM)
- *Which components and interfaces are you working on?* (show in PCM)

- *How confident are you regarding software security on a scale of 1 to 5 with 5 being the best?*

Explanation main task:
The main task is to further specify elicited security requirements. At this point we switch to a Google Docs document with about 30 requirements. The aim is to automatically verify security requirements. For the specification, information is needed about which parts of the software architecture the requirement relates to.

- Two examples are given, each includes sentences before and after the specification
- Think aloud about possible specifications. You can think about the deployment, involved components, data or data transfer.
- We go through the requirements sorted by security categories and you write down the specified requirement below the corresponding sentence.
- There is the possibility to add security requirements at the end of each category

Main part: (30-45min)

[For each of the four security categories confidentiality, integrity, availability, and authentication]
Provide short definition of the category to participant

Placeholder for the security requirements of the category

[ask probing questions if the new requirement is not specific enough, like: *Is there enough information for you to verify the requirement? What kind of data is important for this requirement? What parts of the software architecture are involved in this requirement?*]

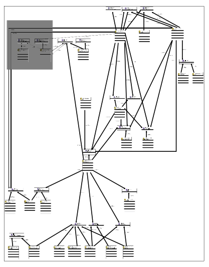
Are there any other security requirements for that category that are not yet mentioned?
You can write them down additionally in the document.

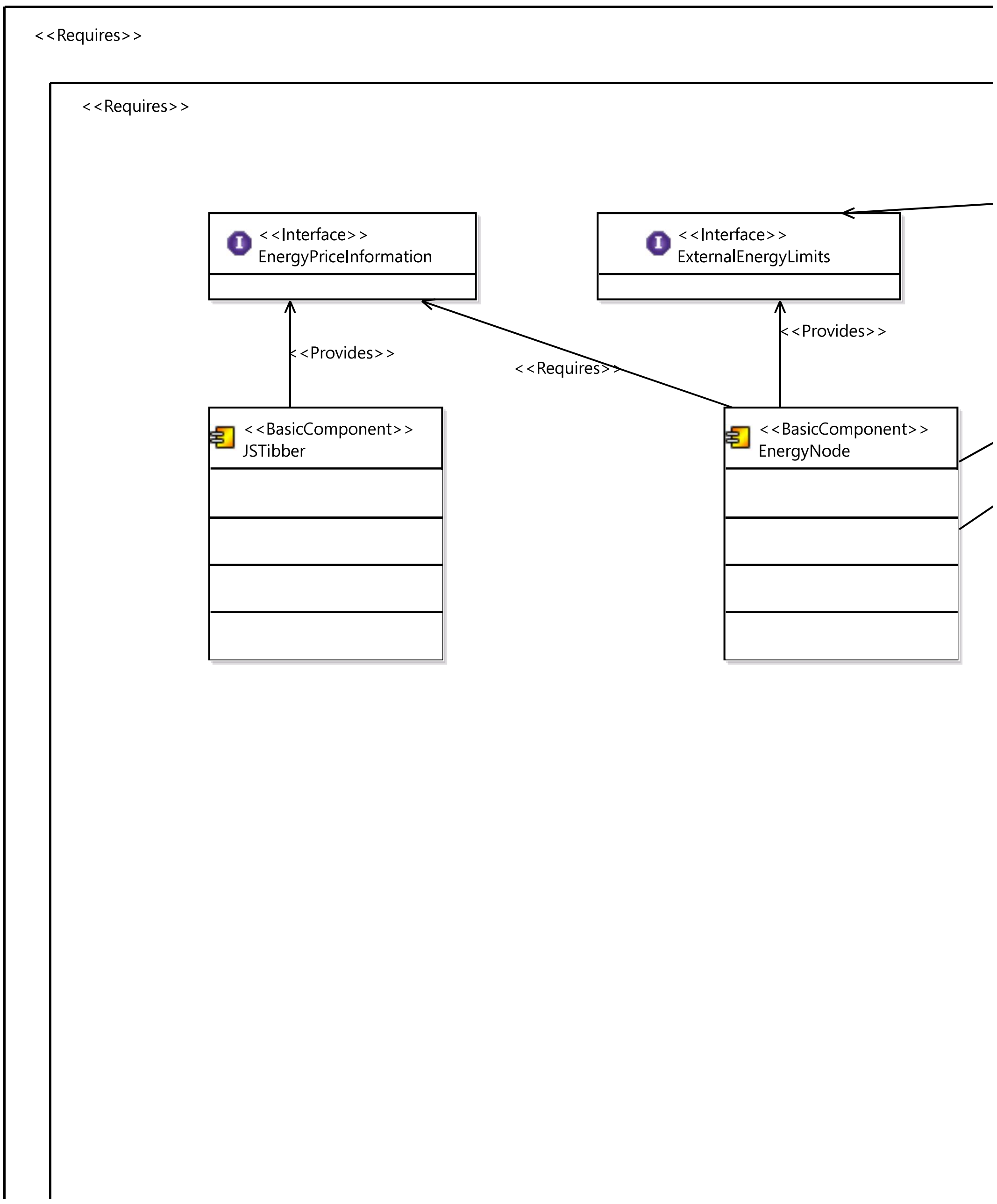
Is there anything else you would like to tell me?

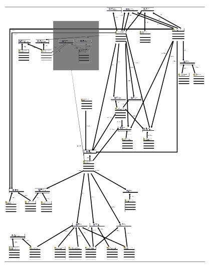
Ending: (3min)

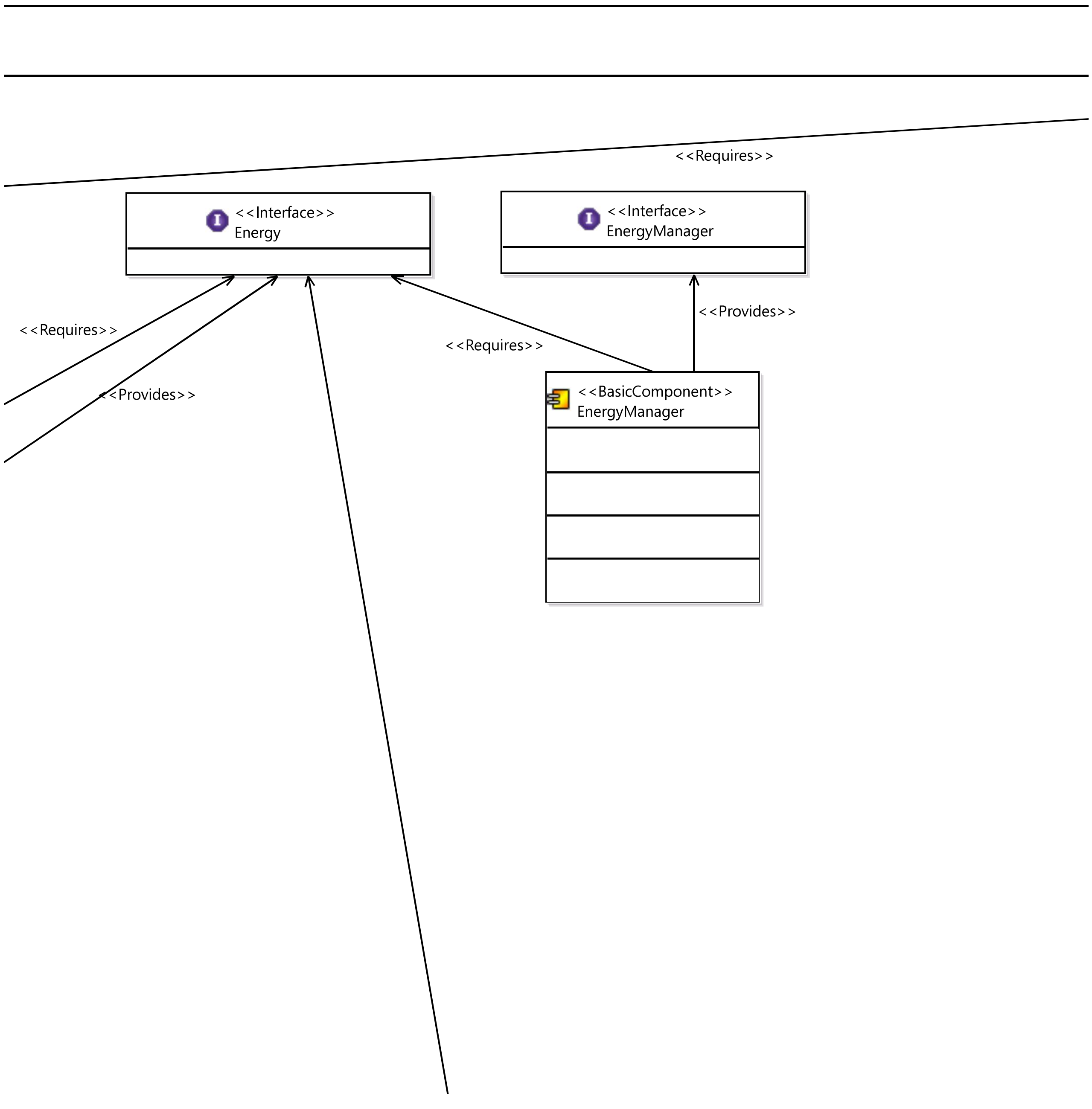
- Thank the participant
- Ask the participant if he is available for further questions

A.4. EVerest Full Component Model

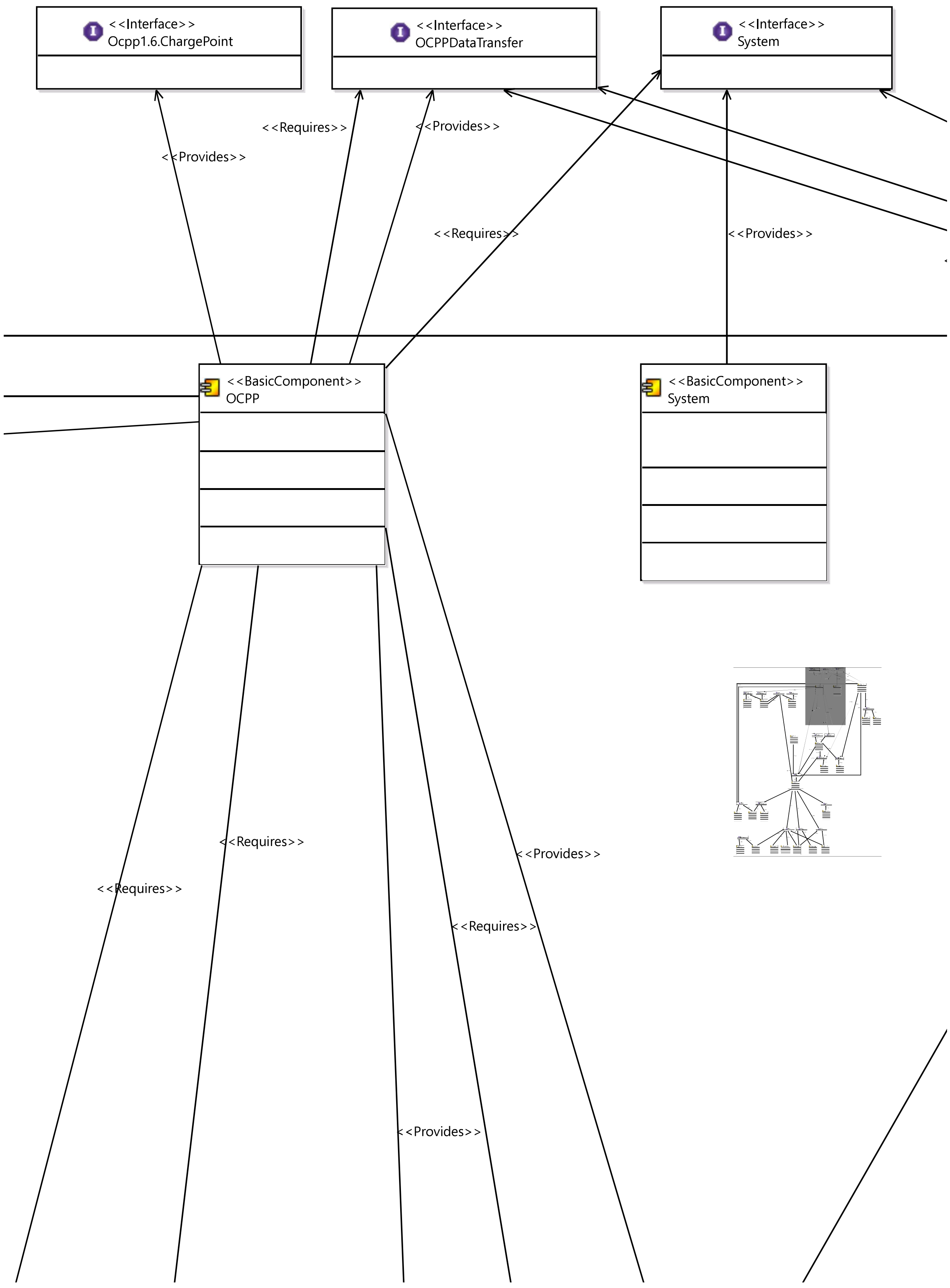


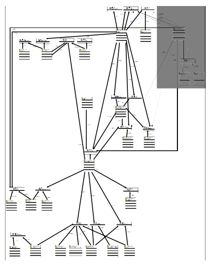


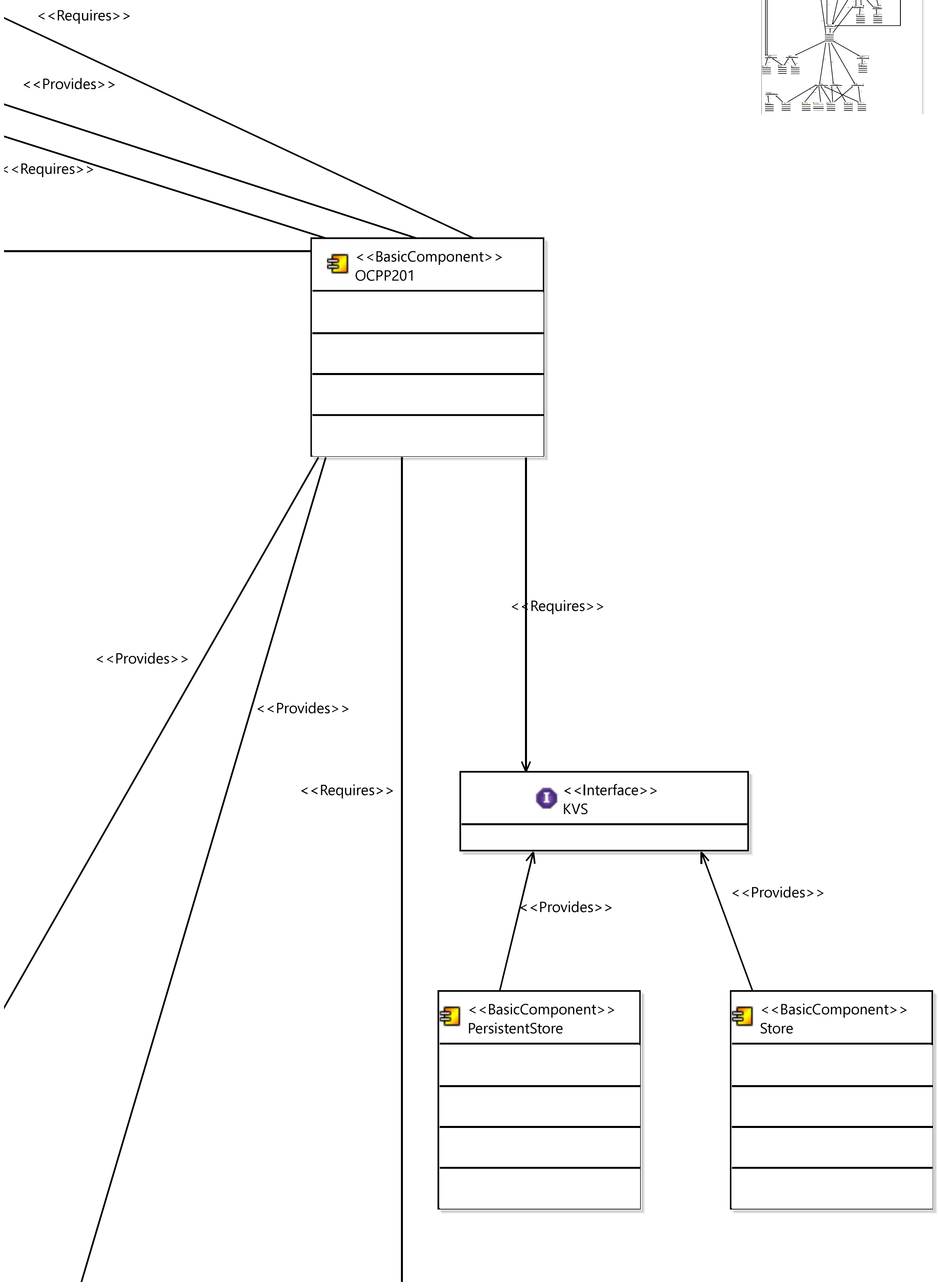


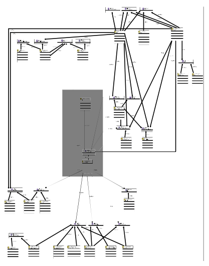


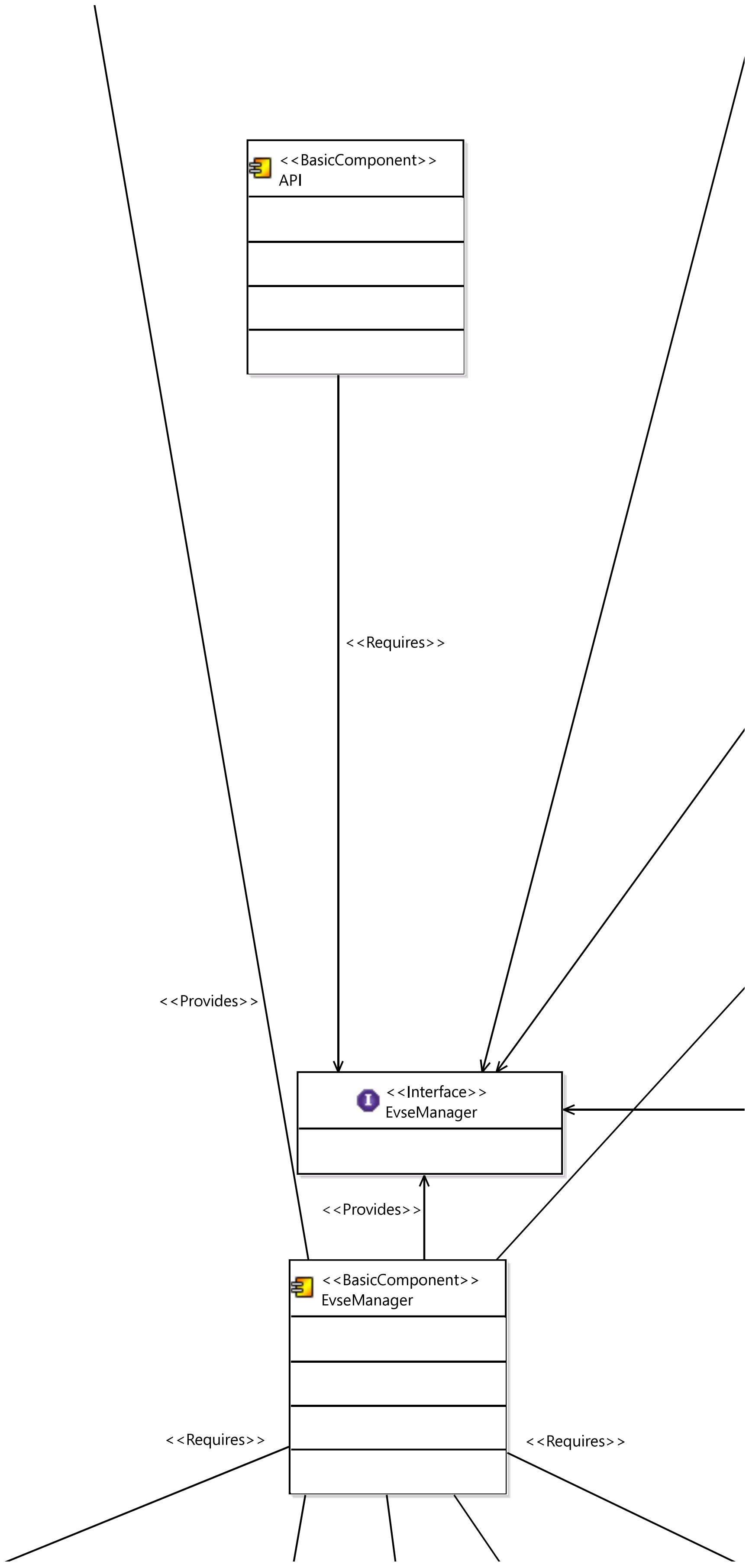
<<Requires>>
<<Interface>>
Energy
<<Interface>>
EnergyManager
<<Requires>>
<<Provides>>
<<Requires>>
<<Provides>>
<<BasicComponent>>
EnergyManager

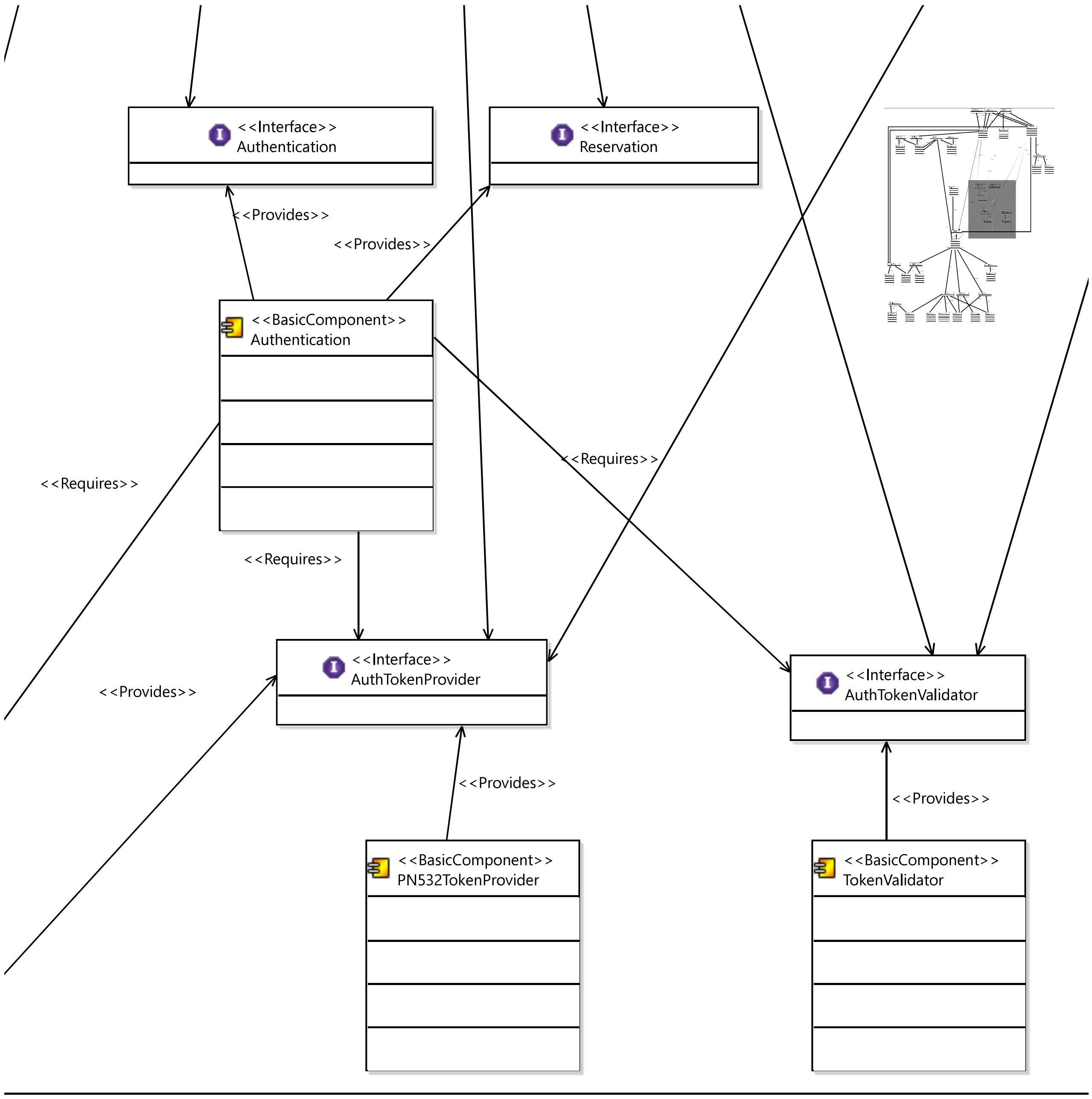

<<Interface>>
Ocpp1.6.ChargePoint
<<Interface>>
OCPPDataTransfer
<<Interface>>
System
<<Requires>>
<<Provides>>
<<Provides>>
<<Requires>>
<<Provides>>
<<BasicComponent>>
OCPP
<<BasicComponent>>
System
<<Requires>>
<<Requires>>
<<Provides>>
<<Requires>>
<<Provides>>

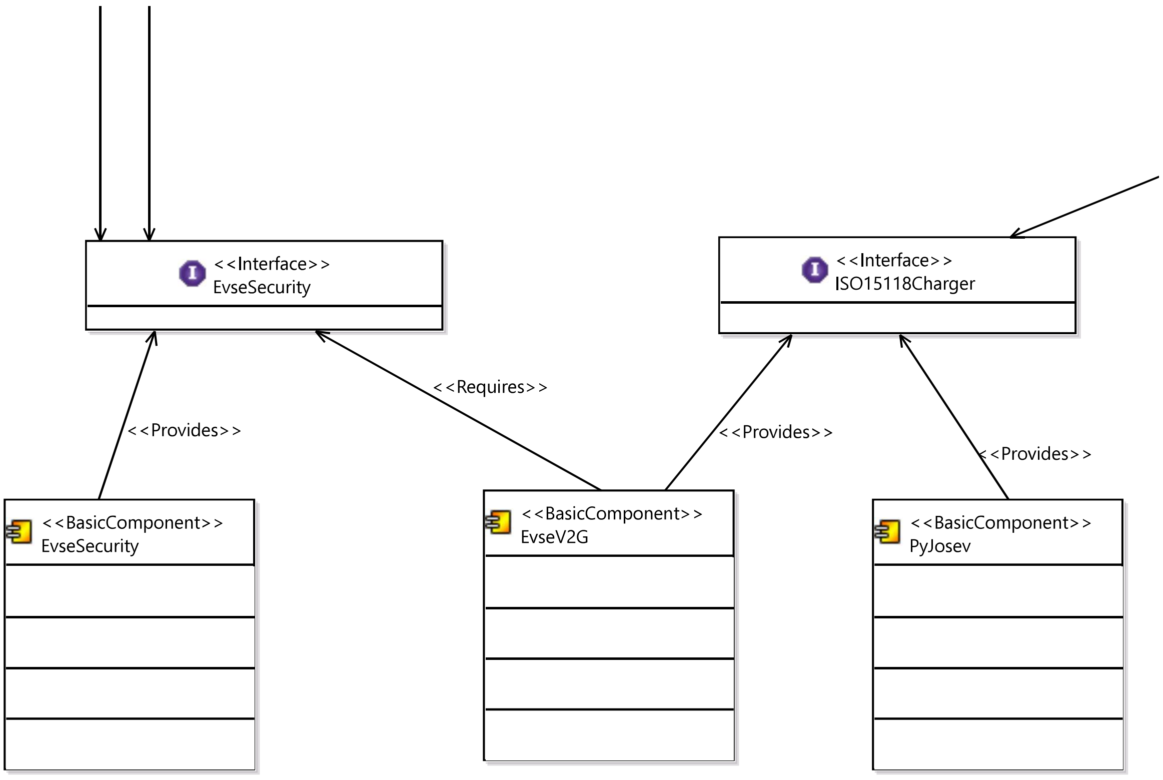


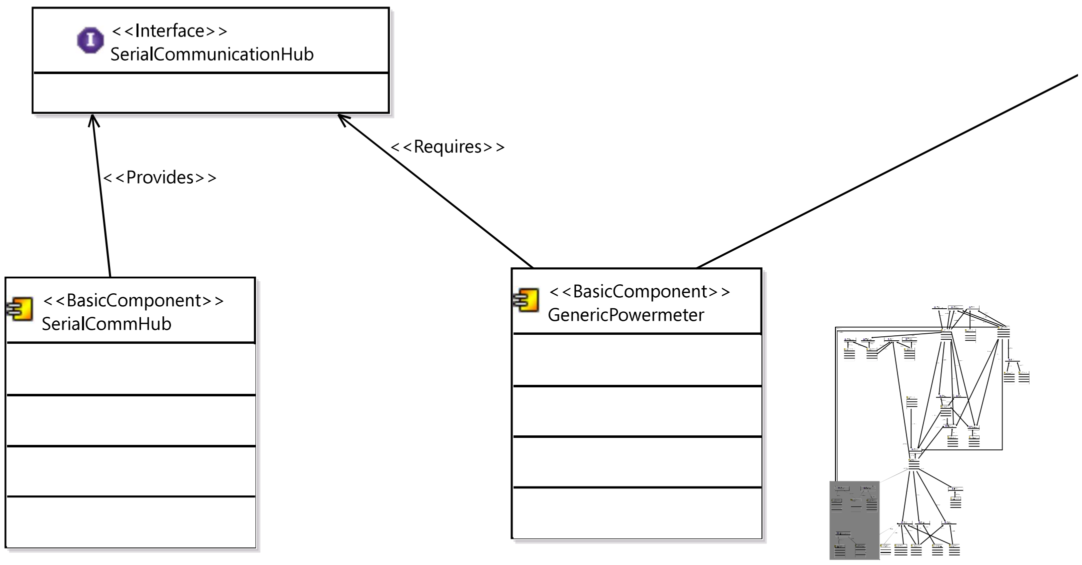

<<Requires>>
<<Provides>>
<<Requires>>
<<BasicComponent>> OCPP201
<<Requires>>
<<Provides>>
<<Provides>>
<<Requires>>
<<Interface>> KVS
<<Provides>>
<<Provides>>
<<BasicComponent>> PersistentStore
<<BasicComponent>> Store

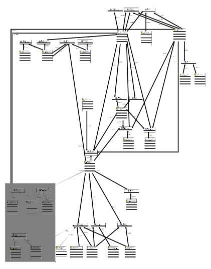


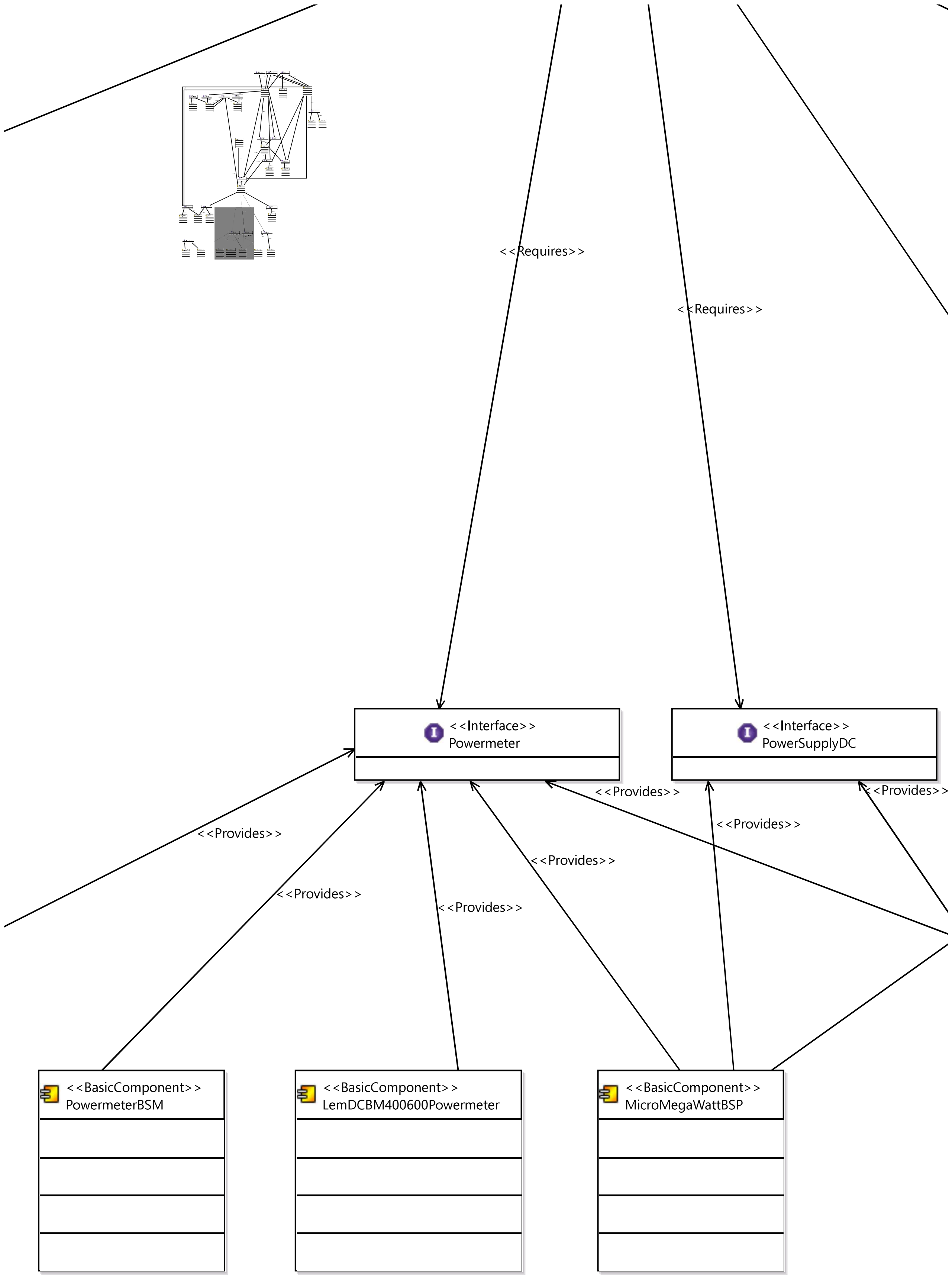

<<BasicComponent>>
API
<<Requires>>
<<Provides>>
<<Interface>>
EvseManager
<<Provides>>
<<BasicComponent>>
EvseManager
<<Requires>>
<<Requires>>

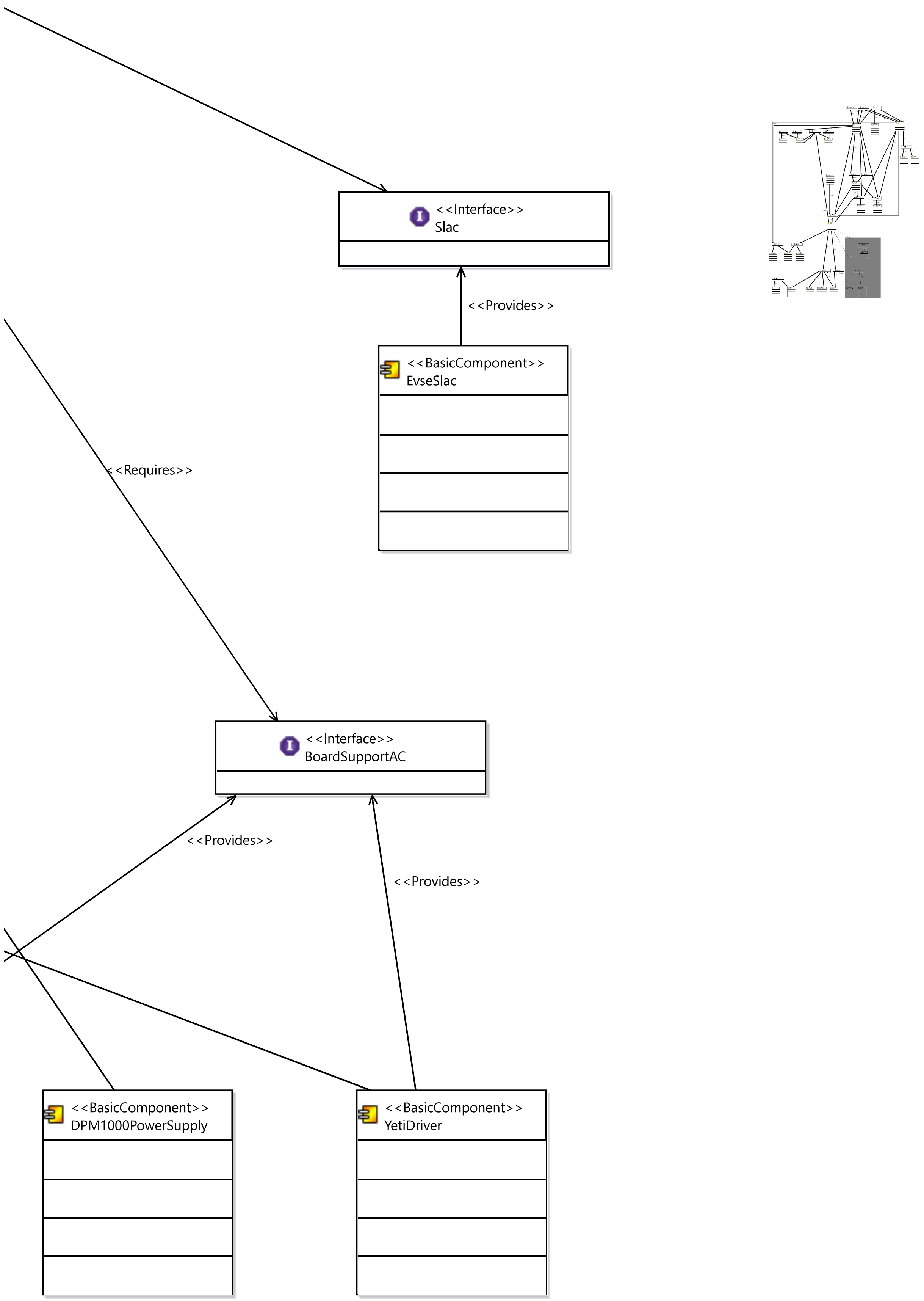

<<Interface>>
Authentication
<<Interface>>
Reservation
<<Provides>>
<<Provides>>
<<BasicComponent>>
Authentication
<<Requires>>
<<Requires>>
<<Requires>>
<<Interface>>
AuthTokenProvider
<<Provides>>
<<Interface>>
AuthTokenValidator
<<Provides>>
<<Provides>>
<<BasicComponent>>
PN532TokenProvider
<<BasicComponent>>
TokenValidator


<<Interface>>
EvseSecurity
<<Interface>>
ISO15118Charger
<<Requires>>
<<Provides>>
<<Provides>>
<<Provides>>
<<BasicComponent>>
EvseSecurity
<<BasicComponent>>
EvseV2G
<<BasicComponent>>
PyJosev


<<Interface>>
SerialCommunicationHub
<<Requires>>
<<Provides>>
<<BasicComponent>>
SerialCommHub
<<BasicComponent>>
GenericPowermeter




<<Requires>>
<<Requires>>
<<Interface>>
Powermeter
<<Interface>>
PowerSupplyDC
<<Provides>>
<<Provides>>
<<Provides>>
<<Provides>>
<<Provides>>
<<Provides>>
<<Provides>>
<<Provides>>
<<BasicComponent>>
PowermeterBSM
<<BasicComponent>>
LemDCBM400600Powermeter
<<BasicComponent>>
MicroMegaWattBSP


<<Interface>>
Slac
<<Provides>>
<<BasicComponent>>
EvseSlac
<<Requires>>
<<Interface>>
BoardSupportAC
<<Provides>>
<<Provides>>
<<BasicComponent>>
DPM1000PowerSupply
<<BasicComponent>>
YetiDriver

A.5. Evaluation Results

		Precision	Recall	F1-score
Activity	**Prompt Engineering**			
	without system mode	0.52	0.95	0.67
	with system mode	0.51	0.95	0.66
	Fine-tuning			
	without system mode	0.62	0.76	0.67
	with system mode	0.68	0.64	0.63
Component	**Prompt Engineering**			
	without system mode	0.71	0.82	0.76
	with system mode	0.72	0.85	0.78
	Fine-tuning			
	without system mode	0.81	0.85	0.82
	with system mode	0.82	0.86	0.84
Data	**Prompt Engineering**			
	without system mode	0.74	0.78	0.76
	with system mode	0.8	0.77	0.79
	Fine-tuning			
	without system mode	0.74	0.78	0.75
	with system mode	0.8	0.79	0.79
Entity	**Prompt Engineering**			
	without system mode	0.67	0.63	0.65
	with system mode	0.65	0.6	0.62
	Fine-tuning			
	without system mode	0.64	0.63	0.62
	with system mode	0.71	0.6	0.64
State	**Prompt Engineering**			
	without system mode	0.48	0.64	0.55
	with system mode	0.52	0.62	0.57
	Fine-tuning			
	without system mode	0.64	0.71	0.65
	with system mode	0.57	0.64	0.58

Table A.1.: Evaluation of the element type classification: Macro averages for all element types for $\text{Prompt}_{\text{identify\_all}}$, n=517

		Precision	Recall	F1-score
Activity	**Prompt Engineering**			
	without system mode	0.53	0.9	0.6
	with system mode	0.56	0.86	0.68
	Fine-tuning			
	without system mode	0.66	0.67	0.65
	with system mode	0.69	0.6	0.63
Component	**Prompt Engineering**			
	without system mode	0.76	0.73	0.75
	with system mode	0.76	0.83	0.8
	Fine-tuning			
	without system mode	0.8	0.85	0.82
	with system mode	0.77	0.87	0.81
Data	**Prompt Engineering**			
	without system mode	0.78	0.69	0.73
	with system mode	0.75	0.7	0.73
	Fine-tuning			
	without system mode	0.72	0.8	0.76
	with system mode	0.73	0.8	0.76
Entity	**Prompt Engineering**			
	without system mode	0.64	0.55	0.6
	with system mode	0.69	0.48	0.56
	Fine-tuning			
	without system mode	0.68	0.58	0.61
	with system mode	0.68	0.5	0.55
State	**Prompt Engineering**			
	without system mode	0.49	0.6	0.54
	with system mode	0.54	0.54	0.54
	Fine-tuning			
	without system mode	0.63	0.67	0.63
	with system mode	0.58	0.61	0.58

Table A.2.: Evaluation of the element type classification: Macro averages for all element types for Prompt$_{\text{classify\_all}}$, n=517

		Precision	Recall	F1-score	Sample Size
Activity	**Prompt Engineering**				
	without system mode	0.26	0.26	0.26	93, 92, 94
	with system mode	0.28	0.28	0.28	93, 92, 93
	Fine-tuning				
	without system mode	0.76	0.76	0.76	74, 78, 71
	with system mode	0.75	0.75	0.75	57, 58, 69
Component	**Prompt Engineering**				
	without system mode	0.75	0.75	0.75	118, 120, 119
	with system mode	0.74	0.74	0.74	124, 122, 124
	Fine-tuning				
	without system mode	0.89	0.89	0.89	119, 123, 119
	with system mode	0.87	0.87	0.87	126, 122, 128
Data	**Prompt Engineering**				
	without system mode	0.77	0.77	0.77	87, 86, 89
	with system mode	0.84	0.84	0.84	86, 87, 85
	Fine-tuning				
	without system mode	0.85	0.85	0.85	83, 93, 87
	with system mode	0.89	0.89	0.89	91, 88, 89
Entity	**Prompt Engineering**				
	without system mode	0.64	0.64	0.64	70, 71, 71
	with system mode	0.67	0.67	0.67	68, 65, 68
	Fine-tuning				
	without system mode	0.88	0.88	0.88	77, 78, 59
	with system mode	0.86	0.86	0.86	72, 68, 66
State	**Prompt Engineering**				
	without system mode	0.25	0.25	0.25	32, 32, 31
	with system mode	0.25	0.25	0.25	30, 31, 31
	Fine-tuning				
	without system mode	0.72	0.72	0.72	34, 36, 31
	with system mode	0.78	0.78	0.78	25, 36, 31

Table A.3.: Evaluation of the element type extraction regarding the element type: Macro averages for all element types for Prompt$_{\text{identify\_all}}$ with their respective sample sizes

		Precision	Recall	F1-score	Sample Size
Activity	**Prompt Engineering**				
	without system mode	0.27	0.27	0.27	88, 86, 89
	with system mode	0.27	0.27	0.27	85, 86, 82
	Fine-tuning				
	without system mode	0.8	0.8	0.8	69, 66, 63
	with system mode	0.8	0.8	0.8	60, 62, 52
Component	**Prompt Engineering**				
	without system mode	0.67	0.67	0.67	106, 108, 106
	with system mode	0.7	0.7	0.7	122, 119, 120
	Fine-tuning				
	without system mode	0.89	0.89	0.89	129, 125, 120
	with system mode	0.89	0.89	0.89	128, 124, 128
Data	**Prompt Engineering**				
	without system mode	0.66	0.66	0.66	80, 77, 74
	with system mode	0.68	0.68	0.68	78, 82, 75
	Fine-tuning				
	without system mode	0.86	0.86	0.86	91, 91, 88
	with system mode	0.91	0.91	0.91	86, 104, 94
Entity	**Prompt Engineering**				
	without system mode	0.6	0.6	0.6	62, 63, 61
	with system mode	0.66	0.66	0.66	56, 54, 51
	Fine-tuning				
	without system mode	0.85	0.85	0.85	65, 57, 67
	with system mode	0.9	0.9	0.9	48, 60, 52
State	**Prompt Engineering**				
	without system mode	0.24	0.24	0.24	30, 30, 38
	with system mode	0.19	0.19	0.19	29, 24, 26
	Fine-tuning				
	without system mode	0.82	0.82	0.82	34, 30, 32
	with system mode	0.77	0.77	0.77	30, 29, 27

Table A.4.: Evaluation of the element type extraction regarding the element type: Macro averages for all element types for Prompt$_{\text{classify\_all}}$ with their respective sample sizes