An Application of Natural Language Processing to Domain Modelling – Two Case Studies

Leonid Kof
Fakultät für Informatik, Technische Universität München,
Boltzmannstr. 3, D-85748 Garching bei München, Germany
kof@informatik.tu-muenchen.de

October 29, 2004

Abstract

This paper describes an approach for analysis of natural language requirements documents and two case studies conducted to prove the feasibility of the approach. The goal of the analysis is to define a common language understood both by the domain expert and the software engineer. To define such a common language, it is necessary to extract terms from the text written by domain expert. The extracted terms must be classified to build a taxonomy. This taxonomy is augmented by associations between terms to build a domain ontology.

The paper introduces two case studies that illustrate feasibility of the approach. The first case study was conducted to test the approach itself and showed that the approach works, but certain interaction with human analyst is necessary. The second case study showed that this interaction does not consume too much time so that the approach scales to larger documents.

1 Introduction

Precise specification is a key success factor for a software project. Formal specification is ideal for the software developer, but it is not reasonable to require the author of the requirements document, who is seldom familiar with formal methods or even with the concept of specification, to provide a formal description.

According to Berry [4] and Mich et al. [17], the overwhelming majority of requirements are written in natural language. For the reader of such a requirements document it is very important to identify the concepts used by the writer (domain expert) and relations between them. These concepts and relations build the basis of the common language used both by the requirements engineer and the domain expert. On the basis of these terms it is also possible to construct a domain ontology. Domain ontology is itself a valuable requirements engineering product, as stated by Breitman et al. [6].

The necessity to extract information from natural language documents motivated a lot of research on application of text analysis in requirements engineering.

1.1 Related Work

Although Kevin Ryan claimed in 1992 [26] that natural language processing (NLP) is not ripe enough to be used in requirements engineering, it is nevertheless possible to

identify several useful applications:

- According to Ryan [26], NLP cannot understand the text. But it is not the goal of most NLP-using approaches to understand the text, the goal is often to extract concepts contained in the document.

- Kevin Ryan [26] claims also that the domain model produced by NLP means may be incomplete because some information that is thought to be common domain knowledge is omitted in the requirements text. But this is exactly one of the tasks of the requirements engineer to detect such omissions. So, an incomplete extracted model would be an indicator for some omissions in text.

To put it in a nutshell, there are some sensible NLP applications in requirements engineering even though NLP is not yet capable of proper text understanding.

Ben Achour [3] classifies the linguistic methods as either lexical or syntactic or semantic. Lexical methods, as for example AbstFinder [14] are the most robust ones. AbstFinder extracts the terms (lexica) that occur repetitively in the specification text. This method is extremely robust because it does not rely on part–of–speech analysis, parsing or something like that. It just considers sentences as character sequences and searches for common subsequences in different sentences. It does not perform any term classification.

In the syntactical approaches, like in those of Abbott [1], Chen [8] and Saeki et al. [27], one analyzes either parts of speech (substantives, verbs, etc.) or looks for special sentence constructions. Abbott [1] analyzes used substantives to produce the data types for the construction of the program. Saeki et al. [27] additionally analyze verbs and declare them to operations. Saeki et al. also introduce a tool helping to copy extracted concepts to the formal specification. However, part–of–speech identification and concept extraction are still performed manually. Chen [8] goes further than Abbott and produces an E/R–diagram as analysis result. To construct the E/R–diagram, Chen searches for special sentence constructions that become the basis for certain relations. The set of constructions that can be analyzed is finite and the sentences that do not fit into one of the predefined types cannot be analyzed.

Semantic approaches, like those of Fuchs [12], Rolland and Ben Achour [24], Gervasi and Nuseibeh [13] and Vadera and Meziane [29] promise more than the other two classes: They interpret each sentence as a logical formula. The syntax of the formula can vary: it can be a kind of first–order logic, like in ACE (Attempto Controlled English [12]) or based on verbs and their arguments, like in the approaches by Rolland and Ben Achour [24] and Gervasi and Nuseibeh [13], but the principle remains the same: each sentence of the specification text is translated into a formula. As this goal is difficult to achieve, such approaches require that sentences follow a certain pattern for the analysis to function automatically. They require also that all the words used in the specification text be previously explicitly defined. Some of the approaches (ACE [12]) use purely manual lexicon construction, others (the one by Vadera and Meziane [29]) use manual lexicon extraction of similar type as in syntactical approaches.

There are also some other related approaches, as for example approaches by Chris Rupp [25], Natt och Dag et al. [20] and Fabbrini et al. [10]. They are related in the sense that they analyze also requirements documents written in natural language, but they do not fit into the above classification scheme because their goal is not to automate the extraction of the information from text.

Chris Rupp [25] defines a set of writing rules (writing templates). The templates define which arguments are necessary for which verbs. For the sentences written ac-

cording to templates, one can manually extract the actors, actions and objects. The goal of the approach is to unify the writing, which allows to produce better requirements documents. The approach does not offer automated support for text analysis. Fabbrini et al. [10] introduce categories of key words that are undesirable in requirements documents (e.g., "if needed", "as ... as possible", etc.) and measures document quality by counting the undesirable constructions. Natt och Dag et al. [20] search for similar and related requirements by finding common concepts (words) in different requirement phrasings. These approaches are interesting for producing qualitative requirements, but they are not comparable with the approach presented in this paper, whose primary goal is to extract domain information from the document.

1.2 Goals of the Presented Work

Daniel Berry addressed in his talk "Natural language and requirements engineering – nu?" [4], what could help to reduce the disadvantages of natural language specifications:

1. Learn to write less ambiguously and less imprecisely
2. Learn to detect ambiguity and imprecision
3. Use a restricted natural language which is inherently unambiguous and more precise

Additionally to the extraction of the domain ontology, the goal of the approach presented in this paper is to address the first two points. "Learn to write less ambiguously and less imprecisely" means that a set of writing rules is introduced. These rules make the text less ambiguous from the point of view of the human reader and at the same time make computer–based analysis better applicable.

The approach detects also ambiguities in the specification text and eliminates them. Ambiguities mean in this context inconsistencies in term usage. When the specification text is good enough, it is possible to extract a domain ontology from the text. A text is good enough for the analysis, when it is consistent in term usage and grammatically correct.

The goal of the presented work is also to further develop syntactical approaches mentioned in the previous section. The objective is to support the extraction both of the terms and the relations between them, aiming at building an application domain ontology, thus providing more than purely syntactical approaches. Nevertheless, the presented approach does not require firm sentence structure, like semantical approaches.

This paper describes a set of techniques to extract terms and relations between them. It introduces also two case studies conducted to prove the applicability of these extraction techniques.

The paper is organized in the following way: Section 2 introduces theoretical concepts and the tools implementing these concepts, Sections 3 and 4 describe the case studies, and Section 5 sums up the results.

2 Steps of Ontology Extraction

Extraction of domain knowledge consists of three basic steps:

1. term extraction

2. term clustering and taxonomy building

3. finding associations between extracted terms

The domain model to be extracted is built of the terms with the associations between them.

Term Extraction: The aim of this step is to extract predicates with their arguments (subjects and objects). Subjects and objects are the terms to extract, predicates are used to classify them. See Subsection 2.2 for details of term clustering and classification process.

In the very first step of term extraction, each sentence is properly parsed. Proper parsing means that a parse tree is built, as opposed to Part–of–Speech (POS) tagging, which just marks every word as a substantive / adjective / verb / ... POS tagging would suffice for automation of the approaches of Abbott [1] and Saeki et al. [27]. Proper parsing eases the extraction of predicates and their arguments, which, in turn, is used for classification of the extracted terms. See Subsection 2.1 for details.

Term Clustering: Extracted terms are clustered according to grammatical contexts (verb subcategorization frames) they are used in. Primary clusters are built by either subjects or objects of the same predicate. Cluster overlaps are used to find related clusters and to join them. The tool ASIUM [11] is used for term classification. The result of this classification is an initial domain taxonomy. See Subsection 2.2 for details of term clustering.

Association Mining: This step takes the taxonomy generated in the second step as input and enriches it by general associations between extracted terms. The idea is borrowed from data mining, as described by Maedche and Staab [16]. Text is considered as a set of database transactions and the terms occurring often in the same transaction are assumed to be related. Additionally, the relations are lifted to the right ontology level, as described by Srikant and Agrawal [28]. Details of association mining are presented in Subsection 2.3

After the last step an initial application domain model is built, represented as a set of terms and binary relations between these terms.

The rest of this section describes the ontology building procedure in more detail.

2.1 Subcategorization Frame Extraction

A verb subcategorization frame is a predicate with its arguments (subject and objects). Subcategorization frames are used by the tool ASIUM [11] for term classification and clustering. Extraction of subcategorization frames uses as input parse trees produced by the parser. The parser by Michael Collins [9] is used to produce parse trees. The parser provides also information about the head child of every tree node. The head child of a parse tree node is the most important child, capturing the actual meaning of the node. The following oversimplified grammar illustrates the idea of head child: Head children are marked bold.

$$\begin{aligned} S &\rightarrow NP\ \mathbf{VP} \\ NP &\rightarrow DT\ \mathbf{NN} \\ VP &\rightarrow \mathbf{VB}\ NP \end{aligned}$$

Tag meanings are introduced in the "Bracketing Guidelines for Treebank II Style Penn Treebank Project" by Bies et al. [5]. In a nutshell, S and similar tags like S-A, S-B mark complete sentences, VP and similar tags mark verb phrases, VB marks verbs, MD modal verbs, NP marks noun phrases, PP marks prepositional phrases and NN marks nouns.

For the usage of ASIUM the predicate and its arguments (subject and objects) are extracted from each sentence. This process is illustrated on the tree shown in Figure 1.

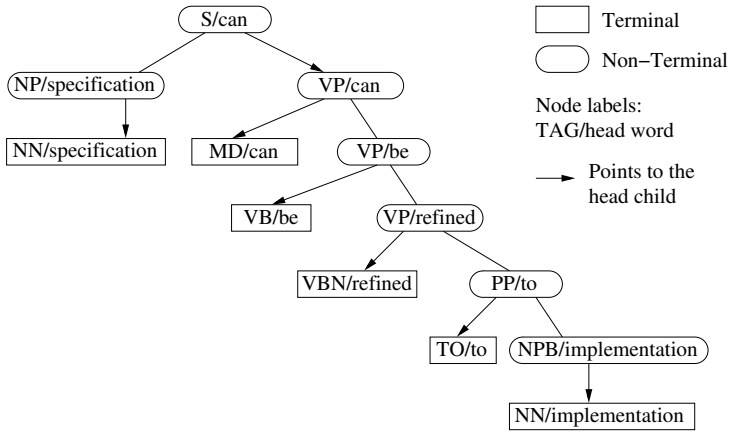

Figure 1: Parse tree for "`Specification can be refined to implementation.`"

To extract the predicate, the extraction algorithm descends from the root node to the head leaf. That is, it descends to the root's head child, then to its head child and so on. This descend process yields the main verb of the sentence. For example, this method extracts "can" from "can be refined ...". "Can" is not really interesting for term classification, so it is necessary to correct the extracted predicate.

The correction algorithm works in the following way: It starts with the verb node extracted initially, i.e. "MD/can" in the case of Figure 1, and looks for sibling verb or verb phrase nodes. In the case of Figure 1 the algorithm finds "VP/be". It descends from "VP/be" to its head child node "VB/be" and looks for the sibling verb or verb phrase nodes again. In such fashion it reaches "VBN/refined". This node does not have any sibling verb or verb phrase nodes, so "VBN/refined" is the verb node that is interesting for term classification.

To extract the subject, the extraction algorithm starts with the main predicate node, e.g., "VP/can" in Figure 1 and traverses the parse tree to the left until it finds a noun phrase. In Figure 1 it finds "NP/specification". Then it descends to the head child of the noun phrase, which is "NN/specification".

The direct object is extracted in a similar way: The extraction algorithm starts with the significant predicate node, i.e., "VBN/refined" in Figure 1 and traverses the parse tree to the right, looking for the *last* noun phrase. (See also "Bracketing Guidelines for Treebank II Style Penn Treebank Project" [5], page 12.) In the case of Figure 1 there is no direct object. The indirect object is the noun or prepositional phrase situated between the verb and the direct object. If the algorithm finds a direct object, it looks for such a phrase between the significant verb node "VBN/refined" and the direct object,

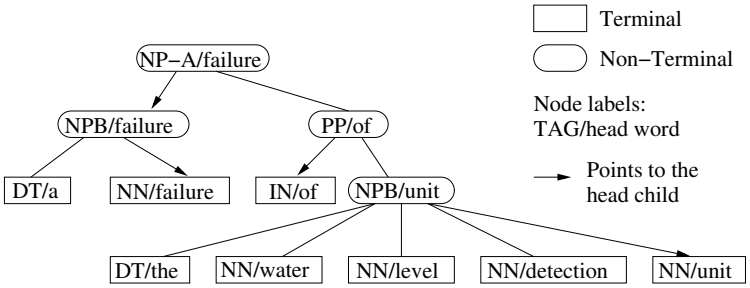

Figure 2: Parse subtree for "failure of . . . "–construction

otherwise it looks for a prepositional phrase closest to the significant verb node. In the case of Figure 1 it extracts "PP/to". The prepositional phrase is then split into the preposition "TO/to" and the noun phrase "NPB/specification". Then it descends to the head child of the noun phrase, as in the case of subject extraction.

2.1.1 Extraction of Compound Concepts

The procedure described above extracts just concepts consisting of a single word. As many examples in case studies show, most concepts are compound ones. To extract compound concepts the extraction algorithm ascends from the leaf noun to the next noun phrase. For example, in the case of parse tree shown in Figure 2 it ascends from "NN/failure" to "NPB/failure". For the example in Figure 2 it results in extracting "a failure" instead of "failure". If the node "NPB/failure" had more children, it were also possible to extract, for example, "a fatal failure" using the same ascending idea.

During the experiments with this extraction heuristics it was found out that there are many compound concepts of the form $\langle PROPERTY \rangle\ of\ \langle OBJECT \rangle$, like "failure of control unit". To extract concepts of this type, the extraction algorithm checks the node directly to the right from the noun phrase node. If this node contains "of" as its head word, the algorithm ascends one level up. For example, for the parse tree in Figure 2 it starts with the node "NPB/failure" and checks the node "PP/of". The head word of "PP/of" is "of", so the extraction algorithm ascends to "NP-A/failure" and extracts the whole subtree "a failure of the water level detection unit".

2.1.2 Stemmer

To properly classify the concepts, it is also necessary to make them grammatically uniform. I.e., it is undesirable to differ between different forms of the same word, like "use", "uses" and "using". Martin Porter's stemmer [22] was used for this purpose. The stemmer was applied after the extraction of subcategorization frames to verbs only. In principle it is possible to use the stemmer also for substantives. The stemmer was not applied to substantives to ease the reading of the resulting subcategorization frames. To unify the spelling of substantives, the ASIUM's orthography correction feature was used.

2.2 Taxonomy Building

The tool ASIUM [11] is based on the assumption that concepts used in the same grammatical context are usually related. It builds clusters of nouns occurring in the same context and looks for common words in different cluster pairs. From ASIUM's point of view, grammatical context is a verb subcategorization frame. If the intersection of two clusters is not empty, the user may join them to a larger cluster. When joining clusters, the user may introduce a generic term describing both joined clusters. In this way the user builds a tree of "is-a" relations, which is the domain taxonomy.

For example, in one of the case studies following clusters were extracted:

$$\{waiting\ state,\ emergency\ stop\ mode,\ normal\ mode,\ degraded\ mode\ \}$$

(direct objects of "enter")
and

$$\{initialization\ mode,\ rescue\ mode,\ emergency\ stop\ mode\}$$

(prepositional objects of "goes into").

The intersection of these two clusters is not empty, so they may be joined to a larger cluster. The user may freely choose the name for the new cluster. In the above example, it would be "operation modes". This new larger cluster can be also used in further analysis of cluster intersections.

2.2.1 Other Clustering Approaches

ASIUM clustering is based on grammatical contexts (subcategorization frames). There are also other clustering approaches, described by Nenadić et al. [21]. Nenadić's term clustering approach is based on three similarity measures:

Contextual Similarity of two terms measures the number of common and different contexts for these two terms. For this measure, a context is defined as a sequence of Part–of–Speech tags occurring in the sentence before and after the term.

Lexical Similarity of two terms measures the presence of common lexical heads (e.g., "message" in "start message" and "stop message") and the number of common modifiers. For example, "first start message" and "second start message" are more similar according to this measure than "start message" and "stop message".

Syntactical Similarity checks for the presence of certain standard constructions. For example, in the construction "Xs, such as A, B, and C", X, A, B and C are seen as similar. The syntactical similarity measure is discrete: It can be either 0, if terms are not similar, or 1, if terms are similar.

To decide whether two terms are similar, one calculates a linear combination of the three above measures. Terms with high similarity can be grouped to clusters.

Unfortunately, it was not possible to compare whether the above approach or the ASIUM–approach, based on grammatical contexts, is more suitable for taxonomy building because the tool ATRACT [18], implementing the approach by Nenadić et al., was developed in an industrial project and is not available for research purposes.

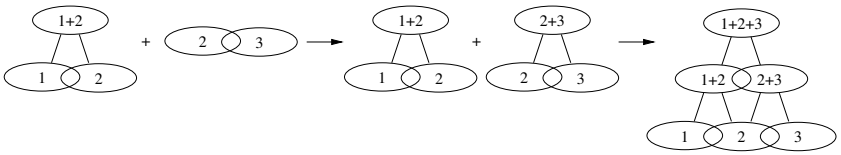

Figure 3: Default ASIUM tree building

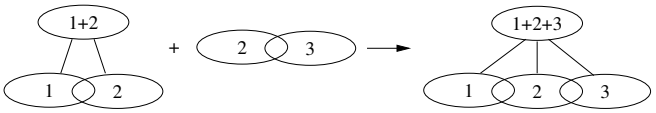

Figure 4: Building a less deep taxonomy tree

2.2.2 Improvements of ASIUM Tree Building Algorithm

During the case studies it turned out that some improvements to the basic tree building algorithm implemented in ASIUM are necessary.

First of all, the standard ASIUM tree building algorithm produces taxonomy trees that are deeper than really necessary. This is due to the fact that ASIUM can join clusters pairwise only. Pairwise joining of clusters is illustrated in Figure 3. Sometimes several clusters belong to the same general concept. In this case it is better to join these clusters in the way shown in Figure 4, without producing intermediate tree levels.

Sometimes it is also the case that one of the intersecting clusters contains more general concepts and the other cluster both more general and less general concepts. For example, in Figure 5 cluster 2 contains more general and cluster 1 less general concepts. In this case it is better to join the clusters and to separate more general concepts from less general concepts. To do so, the clusters are joined, but a set difference for the mixed up cluster is built, as shown in Figure 5.

2.3 Association Mining

The tool KAON [16] borrows its main idea from data mining: It considers the input texts as database transactions and counts how often certain concepts occur in the same transaction. Each transaction is considered just as a set of items. In the case of data mining, it is usually the set of items bought by a single customer.

To decide if an association is important, the tool computes two metrics. For an item set A, let $trans(A)$ be the set of transactions containing A and let N be the total number of transactions. The *support* of the association $A \Rightarrow B$ is defined as $\frac{|trans(A \cup B)|}{N}$. The *confidence* of the association $A \Rightarrow B$ is defined as $\frac{|trans(A \cup B)|}{|trans(A)|}$. An

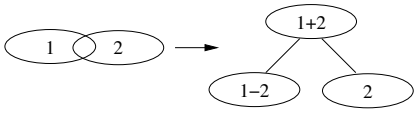

Figure 5: Building cluster difference when joining clusters

association is considered as important if its support and confidence exceed a certain user–defined threshold. Given a taxonomy, KAON can also generalize the association rules as described by Srikant and Agrawal [28].

In order that the data mining idea can be used for association mining, the text is split into single sentences and each sentence is declared to a transaction. The tool KAON offers an own concept extraction facility, based on Part–of–Speech tagging and extraction of tag patterns, so it is possible to extract the set of items (concepts) from each transaction (sentence).

For example, in one of the case studies the association

$$\text{“}transmission\ failure \Rightarrow emergency\ stop\ mode''.$$

got the support of 0.056 and confidence of 0.667. The confidence of 0.667 means that $2/3$ of the sentences containing “$transmissionfailure$” contain also “$emergency\ stop\ mode$”. This association arises from the sentence

“A transmission failure puts the program into the emergency stop mode”,

which is repeated several times in the specification text. For every found potential association the user may decide if it should be included in the ontology. The confidence of 0.667 was considered high enough to include the association “$transmission\ failure$ causes $emergency\ stop\ mode$” into the case study model.

3 First Case Study: The Steam Boiler

In this section the first case study conducted to evaluate applicability of the methods and tools described above is presented.

The steam boiler specification [2] was chosen for the case study because this specification was used also as a benchmark for different formal specification techniques. The specification describes a system consisting of the boiler itself, a valve, four pumps and a couple of measuring devices. The goal of the control program is to maintain certain level of water in the boiler in order to avoid the damage of the boiler. The control program should be fault tolerant and maintain the proper water level despite failures of some hardware units. The goal of the case study was to build a domain model using the techniques described above. Subsection 3.1 presents the case study itself and Subsection 3.2 the lessons learned from this case study.

3.1 Steam Boiler: The Case Study Itself

3.1.1 Overview of the Case Study

In the very first run of the case study the text was analyzed as is, without eliminating inconsistencies. The results of the very first analysis run did not allow to build a *sensible* domain model. Unrelated concepts were put into the same cluster during the taxonomy building. For example, one of the clusters contained both “program” and hardware components:

$$\{program,\ steam\ boiler,\ water\ level\ measuring\ unit,\ pump\},$$

(subjects of “work”)
and another contained completely unrelated terms:

$$\{level,\ mode,\ program\}$$

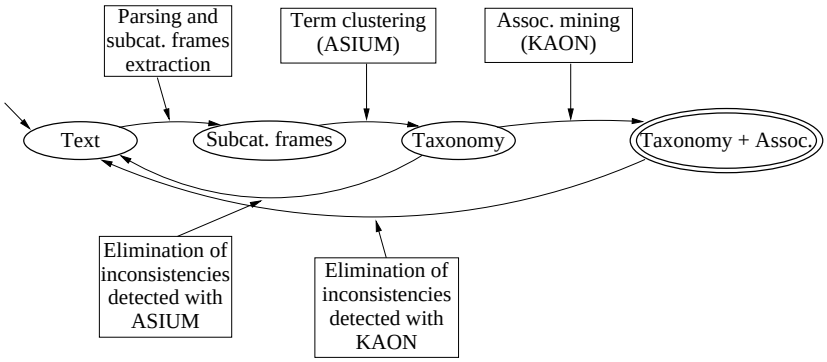

Figure 6: Run of the ontology building case study

(subjects of “reach”).

Association mining was not performed with this original text, as it was necessary to build a sensible taxonomy first.

Manual analysis of the input text, driven by the clustering results, showed that it contained many inconsistencies preventing from direct building of domain ontology. So the case study turned into interleaving steps of ontology building, inconsistence detection, its elimination and rebuilding of the ontology. At the bottom line two ontology building iterations were necessary, each iteration consisting of text parsing, subcategorization frame extraction, taxonomy building with ASIUM and association mining with KAON.

Figure 6 clarifies the idea: The predicates and their arguments were extracted, then they were used for term clustering. During clustering inconsistencies were detected and then eliminated, until there were no more inconsistencies detectable with ASIUM. Then association mining was performed, which discovered further inconsistencies. So the newly detected inconsistencies were eliminated in the specification text and then the same iteration again was performed again: The subcategorization frames were extracted, the terms were clustered and classified, etc.

3.1.2 First Case Study Iteration: Detection and Elimination of Inconsistencies

The first manual analysis of the extracted subcategorization frames showed that some parts of the specification text are unsuitable for sentence–based analysis. I.e., they lose their meaning if considered outside of their original context. For example, the text part describing possible failures looks like this:

> Detection of equipment failures
>
> TRANSMISSION: (1) The program receives a message whose presence is aberrant. (2) The program does not receive a message whose presence is indispensable.

The first sentence of this example causes a wrong parse, as the parser considers “TRANSMISSION” to be a part of the sentence. From the second sentence “program” is extracted as the subject, “receive” as the predicate and “message” as the object. Although this extraction is absolutely correct, it is not what is really necessary: It is necessary to relate failure detection and message reception or non-reception.

The first consequence of this analysis was to replace such construction as cited above by fully-fledged sentences like

> The program detects transmission failure if it receives a message whose presence is aberrant. The program detects transmission failure if it does not receive a message whose presence is indispensable.

This first correction step replaced all the enumeration–like constructions by fully-fledged sentences. This was mostly necessary in constructions like

> Message X: This message is sent ...
>
> Failure Y: This failure is detected when ...

The text without enumeration–like constructions was suitable for analysis with ASIUM. Clustering of the extracted concepts using ASIUM discovered further problems: For example, one of the clusters (direct objects of "enter") consisted of

$$\{state, emergency\ stop\ mode,\ mode,\ mode\ emrgency\ stop\}$$

First of all, this cluster showed that at least one operation mode had several names. Replacing the different names by "emergency stop mode" was easy. The second problem was more interesting: neither "state" nor "mode" is suitable for classification without further specification, which state or mode is meant. Text search showed that "state" comes from the sentence

> The program enters a state in which it waits for the message STEAM-BOILER-WAITING to come from the physical units.

To make the specification more precise, "a state" was replaced by "the waiting state".

The origin of the orphan "mode" was even more interesting: it arose from the sentence

> As soon as this signal has been received, the program enters either the mode normal if all the physical units operate correctly or the mode degraded if any physical unit is defective.

The two mode mentions in this sentence are grammatically incorrect. The parser cannot know that "normal" and "degraded" are mode names and parses them as ordinary adjectives. This puts the words "mode" and "normal" into disjoint subtrees and makes the extraction of compound concepts "normal mode" and "degraded mode" impossible. An additional difficulty arises from the "either ... or"–construction in this sentence. This difficulty is a deficiency of the current heuristics for subcategorization frame extraction, but not an inherent problem. The current extraction heuristics just ignores the conjunctions like "and", "or", "either ... or" altogether. Although the conjunctions are vital for semantics capturing, they are not that important for subcategorization frame extraction. To overcome all these difficulties, the original sentence was replaced by

> As soon as this signal has been received, the program enters either the normal mode or the degraded mode. If all the physical units operate correctly it enters the normal mode. If any physical unit is defective it enters the degraded mode.

A similar problem was detected in other clusters: there were orphan "unit" and "device" terms. Text search discovered "unit which measures the quantity of steam", "unit which measures the outcome of steam", "physical unit which measures the outcome of steam", "device to measure the quantity of steam" and "steam measurement device". All these constructions were replaced by "steam measurement unit". The same name unification was necessary for "pump controller" and "water level measuring unit".

The last curiosity discovered with ASIUM in the first iteration was the cluster

$$\{program,\ physical\ unit\}$$

containing prepositional objects of "emitted by" and "received by". This is an example of metonymy, whereby one object is used to stand for another. The program itself does not send or receive messages, whereas the control unit running the program does. Every human reader understands this substitution, but it provokes senseless clusters. For this reason "program" was replaced by "central control unit" everywhere in the sending or receiving context.

The purified text allowed for building of this simple taxonomy:

- Message sources (prepositional objects of "comes from"):

  $$\{water\ level\ measuring\ unit,\ steam\ measurement\ unit,\ pump\ controller\}$$

- Potentially failing hardware (subjects of "is repaired" (passive form), subjects of "working")

  $$\{water\ level\ measuring\ unit,\ steam\ measurement\ unit,\ pump,\\ physical\ control\ unit,\ pump\ controller\}$$

- Operation modes (direct objects of "enter", prepositional objects of "goes into")

  $$\{waiting\ state,\ emergency\ stop\ mode,\ normal\ mode,\ degraded\ mode,\\ initialization\ mode,\ rescue\ mode\}$$

- Messages (direct objects of "receive", subjects of "indicate", subjects of "is received" (passive form), subjects of "is sent" (passive form))[1]

  $$\{message,\ message\ pump{-}state,\ message\ steam{-}boiler{-}waiting,\\ message\ stop,\ message\ valve,\ message\ open{-}pump,\ \ldots\}$$

- Actuators (direct objects of "activate")

  $$\{valve,\ pump\}$$

- Failures (subjects of "is detected" (passive form), direct objects of "detect", subjects of "put" (In the context "... puts the program into mode XY"))[2]

  $$\{failure,\ pump\ failure,\ transmission\ failure,\ pump\ controller\ failure,\\ water{-}level{-}measuring{-}unit\ failure,\\ steam{-}level{-}measuring{-}unit\ failure\}$$

[1] For this cluster it was necessary to use the improved version of the clustering algorithm, as described in Subsection 2.2.2

[2] For this cluster it was necessary to use the improved version of the clustering algorithm as well.

This taxonomy was transferred to KAON. When extracting the terms with the KAON concept extraction facility, it was discovered that the concept extraction by ASIUM was incomplete. For example, “message level”, “message mode” and many more other messages were not discovered by ASIUM. Recognition of this problem made a second iteration necessary.

3.1.3 Inconsistency Elimination and Ontology Building: Second Iteration

First of all, all the new messages discovered in KAON were marked as compound concepts. It is sufficient to write “message-level” instead of “message level” for the parser to consider it as a single compound concept. During this marking the expression “start or stop message” was discovered , which was replaced by “message-start or message-stop”.

In a similar way “acknowledgement message” and “detection message” were discovered. As neither “acknowledgement message” nor “detection message” is a real message used for communication, the corresponding sentences were rephrased more precisely, so that they specify *which* acknowledgement or detection message is used in every particular case.

Analysis of term clusters extracted from the corrected text showed following problems:

- there were large clusters produced by the verbs “be” and “have”, containing different unrelated concepts
- there were orphan “mode” and “failure” concepts.

The first problem was solved by rephrasing all the sentences containing “be” or “have”. For example, “has a failure” and “is defective” were replaced by “fails”; “is really zero” was replaced by “really equals zero”, etc.

The orphan “mode” arose from “this mode”, where the actual mode was specified in the previous sentence. The current text analysis approach cannot establish relations between sentences, so the only solution was to replace “this mode” by the actual mode name that is meant.

After the purification of the text it was possible to build a *sensible* ontology using the text analysis techniques.

The steam boiler case study showed that the amount of manual work necessary to process the document is not negligible. However, this manual work is not in vain: Detection and correction of inconsistencies is a part of document validation. This part of validation is eased by the tool that fails to extract a consistent ontology from an inconsistent document. The amount of manual work was not measured during the first case study because the goal of this case study was to evaluate the feasibility of the approach itself. Applicability to larger documents was addressed in the second case study, presented in Section 4.

3.2 Steam Boiler Case Study: Results and Lessons Learned

3.2.1 Results: The Domain Ontology

Figure 7 shows a part of the produced ontology. The diagram shows the ontology root (`kaon:Root`), three top-level concepts (`operation mode`, `failure` and `message`) with some of their subordinate concepts and relations between them. Colors and arrows have the following meaning: Variable–width lines denote “is-a”–relations,

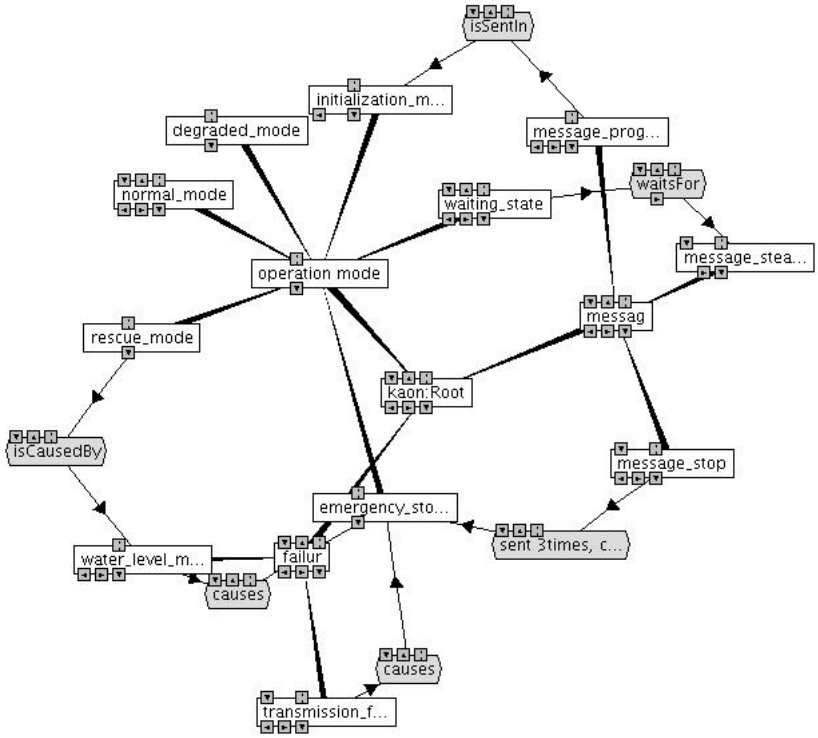

Figure 7: Steam Boiler: part of the produced ontology

the thin end pointing to the more general concept and the thick end pointing to the more special one. White boxes denote concepts and classes of concepts, dark boxes are properties (associations) connecting concepts. Arrow directions and property names are chosen in such a way that they produce sensible sentences. One gets the sentence in reading a concept name, following the arrow to the property and subsequent reading the property and the second concept. For example, one can read "Transmission_failure causes emergency_stop_mode" and "Rescue_mode is_caused_by water_level_measuring_unit_failure".

3.2.2 Writing Rules

The case study showed that extraction results become better if the text follows certain rules. Some of these rules are natural and just require consistency in concept names and in expressions, while one rule is caused by deficiencies of currently available tools. Here it is important to emphasize that the approach *does work* even if the text does not follow the rules, as it worked for the initial version of the requirements document. However, the produced results are less sensible if the document does not follow the guidelines, just confirming the known "garbage in – garbage out" rule.

Here is the summary of writing rules:

- Rules that make sense anyway:
  - Always use the same name for the same concept
  - In the case of compound names, either use names that, put in the sentence, remain grammatically correct (e.g., "normal mode" instead of "mode normal") or mark the compound names as such (i.e., "mode-normal").
  - Always use the complete form in the case of compound names: i.e., "stop message or start message" instead of "stop or start message".
  - Do not use the verbs "be" and "have". They do not provide much information even for the human reader. For the computer–based analysis they produce large clusters of unrelated concepts. Nevertheless, it is allowed to use these verbs to build passive form or perfect tenses. In this case they are easy to filter out.
- Rule due to technical deficiencies:
  - Do not use cross–sentence references like "Message X is sent by unit Y. This message indicates ..." In future versions of the approach it may be sensible to try to use anaphora resolution algorithms, like those developed by Lappin and Leass [15] and Mitkov [19]. Such an algorithm could match "Message X" in the first sentence and "This message" in the second one.

3.2.3 Limitations of the Approach

The approach in its current form has an inherent limitation that further constrain the writing rules. This limitation is due to the clustering algorithm. The algorithm clusters the concepts used with the same verb and looks for cluster intersections. It cannot relate disjoint clusters containing similar concepts. It is possible to define similarity of concepts by means of their main noun. This definition would relate "stop message" with "start message" and "pump state message" and so on. This kind of similarity (lexical

similarity) is taken into account in the ATRACT approach [18]. See Subsection 2.2.1 for the details of similarity measures used by Nenadić et al. [18].

At a first glance, building basic clusters solely on the basis of grammatical contexts does not look like a limitation, but it caused, for example, that "message-pump-control-state" was completely ignored in the first run of the analysis. "Message-pump-control-state" occurs only in the sentence

> Message pump-control-state(n; b) gives the information which comes from the pump controller of pump n (there is flow of water or there is no flow of water).

The verb "give" is also used solely in this sentence, which causes a stand alone concept.

This problem could be solved in two ways:

- Another principle to construct basic clusters can be used. Additionally to similarity of grammatical contexts, which is used now for clustering, it is possible to use measures introduced in Subsection 2.2.1: lexical and syntactical similarity. Lexical similarity would solve the above problem because "message" is the head word of both "Message pump-control-state(n; b)" and other message names.

- The same verbs must be systematically used with related concepts. For example, "indicate" is used with other messages, so "Message pump-control-state(n; b) indicates ..." would solve the problem. In this case it is sufficient stick to basic clusters built on the basis of grammatical contexts.

4 Second Case Study: Instrument Cluster

The goal of the first case study was to test principal applicability of the approach and to experiment with available tools. This case study showed that the approach works, but a certain amount of manual work is necessary. Although this manual work may be perceived as bothersome, this work is necessary to validate the document: Apart from allowing the tool to extract the ontology, it produces a consistent document.

Necessity of manual work gives raise to the question whether the approach scale. A second case study was conducted to prove the scalability. The second case study was based on the DaimlerChrysler Demonstrator [7]. This document is much larger than the steam boiler specification (approx. 80 pages vs. 6 pages for the steam boiler), what makes it suitable for a scalability case study.

The document [7] describes a car instrument cluster, showing the current speed, RPM (motor revolutions per minute), outside temperature and so on. The instrument cluster communicates via CAN bus with other ECUs (electronic control units).

As in the first case study, the goal was to extract the application domain ontology from the document. In the scalability case study the time that was necessary for different process steps was also documented. This made the identification of time consuming steps that potentially do not scale possible.

The rest of this section describes single steps of the case study. Subsection 4.1 describes document preparation, which was necessary for a large document, Subsection 4.2 introduces the results of the first parsing and Subsection 4.3 explains why rephrasing of some text parts were necessary. Subsections 4.4 and 4.5 describe the results of taxonomy building and association mining respectively. Subsection 4.6 summarizes the lessons learned from this case study.

4.1 Document Preparation

Text analysis starts with document preparation. There is a set of purely technical issues that are unimportant for smaller documents, but can become time consuming for larger ones. For the analysis it is necessary to convert the text in one-sentence-per-line format. There are tools that recognize sentence boundaries, as for example the one by Ratnaparkhi [23]. However, it turned out that this approach does not work well if the text contains also incomplete sentences.

So, in the first step of text preparation, the text was manually transformed into one-sentence-per-line format. The formatting and the first reading of the specification text took one working day.

At this stage, grammatically wrong sentences were not reformulated and item lists and tables were not converted to fully-fledged sentences. Although the subcategorization frame extraction in its current form (see Subsection 2.1) works for grammatically correct sentences only, the goal was to see how much noise data is produced in such a way and whether it is really necessary to rephrase incorrect sentences manually.

4.2 Parsing and Information Extraction

After reformatting the text it was possible to parse it and to extract syntax information. The predicate, the subject and objects were extracted from each sentence. Extraction results showed that rephrasing of incorrect sentences was necessary.

By analyzing the extracted predicates and their arguments, a lot of wrong verbs and objects were discovered. For example, the operations "=", "$<$" and "$>$" were classified as verbs, as they often occurred in the specification text in the verb position:

- If Ig-Lock = 1 then the ignition key is in position ignition on.
- If Current-Speed-V $<$ 30 km/h and the Internal-Temp values are sinking, then Outside-Temp-Shown = Internal-Temp.
- If Current-Speed-V $>=$ 50 km/h the rising Internal-Temp values are ignored for 1,5 minutes.

There was an additional problem with the text containing incomplete and grammatically incorrect sentences: The term extraction looks for the sentence predicate and then extracts predicate's arguments (terms). For grammatically incorrect sentences this is not always possible, so incorrect sentences are just ignored during term extraction. If the requirements document contains incorrect sentences, it is not possible to guarantee that all the relevant concepts are extracted. It could happen that some concepts occur in incomplete sentences only, so that they are completely ignored.

For these reasons the next step was to rewrite incomplete sentences into grammatically correct ones.

4.3 Lists and Tables: Proper Phrasing

It turned out that lists and tables were the main source of incomplete sentences. For example, input signals of the instrument cluster were described like this:

Ig-Lock: Describes the position of the ignition key. If Ig-Lock = 1 then the ignition key is in position ignition on. Sent by the ignition lock control unit. Scope: $\{0,1\}$. Received every 100 ms. Transferred by the CAN bus.

Ig-LockR: Describes the position of the ignition key. If Ig-LockR = 1 then the ignition key is in position radio. Sent by the ignition lock control unit. Scope: {0,1}. Received every 100 ms. Transferred by the CAN bus.

Status-Door-dd: Describes the status of the driver's door. Scope: {open (= 1), closed (= 0)}. Sent by the door control unit. Received every 100 ms. Transferred by the CAN bus.

Each phrase of such constructions was completed so that it became a grammatically correct sentence. In most cases it could be done schematically, but the rephrasing still required manual work. For example, the above list was transformed into

- Ig-Lock describes the position of the ignition key. If Ig-Lock equals 1 then the ignition key is in ignition-on-position. Ig-Lock is sent by the ignition lock control unit. Ig-Lock can equal 0 or 1. Ig-Lock is received every 100 ms. Ig-Lock is transferred by the CAN bus.
- Ig-LockR describes the position of the ignition key. If Ig-LockR equals 1 then the ignition key is in radio-position. Ig-LockR is sent by the ignition lock control unit. Ig-LockR can equal 0 or 1 Ig-LockR is received every 100 ms. Ig-LockR is transferred by the CAN bus.
- Status-Door-dd describes the status of the driver's door. Status-Door-dd can equal 0 or 1. If Status-Door-dd equals 1, the driver's door is open. If Status-Door-dd equals 0, the driver's door is closed. Status-Door-dd is sent by the door control unit. Status-Door-dd is received every 100 ms. Status-Door-dd is transferred by the CAN bus.

Some transformations according to the writing rules were necessary as well. (See also Subsection 3.2.2.) These writing rules include:

- always use the same name for the same concept. (The original text obeyed this rule, so no correction was necessary.)
- In the case of compound names, either use names that, put in the sentence, remain grammatically correct (e.g., "normal mode" instead of "mode normal") or mark the compound names as such (i.e., "mode-normal"). In the instrument cluster specification, "position radio" was replaced with "radio position", "switched off position" with "switched-off-position", etc.

Such transformations made syntax–based analysis possible. All these transformations took 1.5 working days, which is justifiable for a 80–page document. The overall time cost for document preparation up to this point amounted to 2.5 working days.

4.4 Taxonomy Extraction

Taxonomy extraction is based on the analysis of cluster intersections. The first ASIUM run showed that there were more than 600 cluster intersections produced by the text. To build a taxonomy it is necessary to analyze cluster intersections, so this step could become time consuming.

During taxonomy building single clusters were analyzed as well to detect wrong usage of terms: Every time a cluster containing unrelated concepts was encountered, it was possible to detect the textual source of this inconsistency and eliminate it.

In the instrument cluster case study relatively small number of inconsistencies was detected:

- The verb "denote" produced a huge concept cluster containing unrelated concepts. This was due to the fact that the verb "denote" occurred both in constructions like "⟨`some-signal`⟩ `denotes` ..." and in "⟨`some-parameter`⟩ `denotes` ..." This problem could be corrected for example by replacing "denote" by "influence" when talking about system parameters. In the case study this correction was not done because both signals and system parameters could be clustered using other verbs. The "denotes"–cluster was just ignored.

- During the clustering it was discovered that some concept names that had to be replaced. The replacement was necessary because several different names were used for the same concept. Following concept names were corrected:

  - engine-warning ⟶ engine-warning-signal
  - indicator-left ⟶ indicator-left-signal
  - indicator-right ⟶ indicator-right-signal
  - turn-signal-left signal ⟶ turn-signal-left
  - turn-signal-right signal ⟶ turn-signal-right
  - the pointer of the engine speed indicator ⟶ rev-meter-display-pointer

  With the corrections described above the following taxonomy was built:

- users (subjects of "adjust", subjects of "enter", subjects of "press", subjects of "release"):

$$\{driver,\ service\ man,\ user\}$$

- hardware (subjects of "determine", prepositional objects of "seen as", prepositional objects of "sent to", prepositional objects of "send to", prepositional objects of "transmitted by", subjects of "turned on")

$$\{system,\ engine\ control\ unit,\ message\ receivers,\ message\ transmitters\}$$

  $Message\ receivers$ and $message\ transmitters$ are clusters on their own, so there are following sub-clusters:

  - message receivers (prepositional objects of "sent to", prepositional objects of "send to")

$$\{engine\ control,\ indicator,\ digital\ display,\ radio,\ display\}$$

  - message transmitters (prepositional objects of "transmitted by")

$$\{can\ bus,\ instrument\ cluster\}$$

- displays (direct objects of "watch")

$$\{rev\ meter,\ speedometer,\ outside\ temperature\ display\}$$

- signal (subjects of “equal”, subjects of “sent” (passive form), direct objects of “sending”, subjects of “transferred”, subjects of “describes”, subjects of “sent by” (passive form), direct objects of “sending”, subjects of “processed”, subjects of “received”, subjects of “transmitted”, direct objects of “equal”, subjects of “describe”). There are too many signals to present all of them, so just a subset is presented here.

$$\{actual\text{–}number\text{–}of\text{–}revolutions,$$
$$actual\text{–}number\text{–}of\text{–}wheel\text{–}revolutions\text{–}sensor1, \ldots,$$
$$actual\text{–}number\text{–}of\text{–}wheel\text{–}revolutions\text{–}sensor4,$$
$$but\text{–}down,\ but\text{–}left,\ but\text{–}minus,\ but\text{–}plus,\ but\text{–}right,$$
$$command,\ computed\text{–}second,\ \ldots\}$$

- errors (subjects of “determined”)

$$\{error,\ problem\}$$

- values
  - adjusted values (subjects of “adjusted”, direct objects of “decrease”, direct objects of “increase”)

$$\{time,\ minutes/hours\}$$

  - computed values (subjects of “computed”, subjects of “calculated”)

$$\{time,\ speed,\ car\ speed\}$$

- pointer (prepositional objects of “goes to”, subjects of “steered” (passive form))

$$\{rev\text{–}meter\text{–}display-pointer,\ the\ pointer\ of\ the\ engine\ speed\ indicator\}$$

- temperature (direct objects of “falling”, subjects of “sinking”)

$$\{temperature\ values,\ internal-temp\ values\}$$

- scale position (prepositional objects of “is below”, direct objects of “remain at”)

$$\{horizontals,\ minsv,\ right\ scale\ end\}$$

- warning (subjects of “appear”)

$$\{warnings\ of\ level\ 2,\ other\ warnings,\ warning\}$$

- actuator (direct objects of “activate”, prepositional objects of “turn off”, subjects of “turned on” (passive form), subjects of “deactivated”, subjects of “activated” (passive form)).

$$\{stepping\ motor,\ automatic\ door\ lock,\ both\ arrows\ of\ the\ indicator\ lights,$$
$$indicatior\ lights,\ lights,\ turn\ signal,\ hazard\ warning,\ display,\ attribute,$$
$$the\ left\ arrow\ of\ the\ indicator\ lights,\ the\ right\ arrow\ of\ the\ indicator$$
$$lights,\ the\ display\ of\ the\ engine\ warning\ light,\ the\ indication\ of\ the$$
$$outside\ temperature,\ radio,\ instrument\ cluster,\ ic,\ ignition,\ engine\}$$

- indication (direct objects of “stop”)

$$\{visible\ and\ audible\ indication,\ hazard-warning\ signal\ flasher,\ blinking\}$$

- suppressed information (subjects of “suppressed” (passive form), subjects of “ignored”)

$$\{numbers\ of\ revolutions\ below\ 320\ min^{-1},\ warnings,\ the\ warnings\ of\ level\ 3,\ messages\ of\ level\ 2,\ rising\ internal-temp\ values,\ r-ic-stat\ messages,\ r-stat\ messages\}$$

- settings (subjects of “stored” (passive form), subjects of “damping”)

$$\{blink-frequency-adj,\ blink-frequency-colon,\ ice-threshold,\ parameter-value,\ release-bit,\ damping,\ variant-car,\ adjustment-speed-minutes,\ 12-24-time-format,\ variant-specific-bit-temp,\ adjustment-speed-hours\}$$

  - $Damping$ is itself a cluster, consisting of subjects of “damping”:

$$damping = \{damping-pt1,\ damping-pt2\}$$

Analyzing the whole plethora of cluster intersections and building a taxonomy (concept and cluster hierarchy) took approximately 1.5 working days. The overall time cost up to this point amounted to 4 working days.

4.5 Association Mining

To explain scalability problems potentially posed by association mining, it is sensible to start by repeating some definitions from Subsection 2.3: For an item set A, let $trans(A)$ be the set of transactions containing A and let N be the total number of transactions. The *support* of the association $A \Rightarrow B$ is defined as $\frac{|trans(A \cup B)|}{N}$. The *confidence* of the association $A \Rightarrow B$ is defined as $\frac{|trans(A \cup B)|}{|trans(A)|}$.

In the case studies the analysis was performed on the per–sentence basis and a transaction was defined as a pair of concepts occurring in the same sentence. For the instrument cluster case study this definition led to more than 1000 potential associations. In order that this plethora of potential associations become manageable, the associations were sorted lexicographically by $(absolute\ frequency,\ confidence)$. Absolute frequency of the association $A \Rightarrow B$ is defined as $|trans(A \cup B)|$. *Formally*, two associations with the same support have also the same absolute frequency, so it is possible to use the standard measure $support$. Due to rounded support values presented by KAON to the user, $absolute\ frequency$ gives more information. Lexicographical sorting means that the associations were sorted by $absolute\ frequency$ and in the case of equal $absolute\ frequency$ they were sorted by $confidence$.

For the ontology building the associations with $absolute\ frequency \geq 5$ were used, which corresponded approximately to the most frequent 25% of associations. It took about one working day to manually validate these associations and to include the relevant ones into the ontology. The overall time cost up to this point amounted to 5 working days.

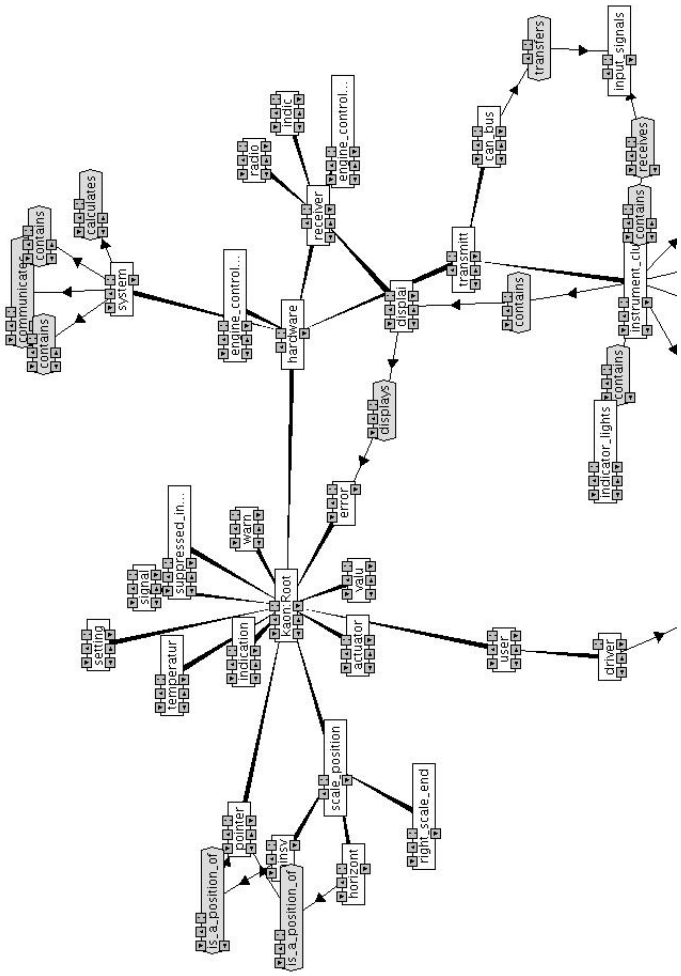

Figure 8: Instrument Cluster: part of the extracted ontology

4.6 Instrument Cluster Case Study: Results and Lessons Learned

The goal of the instrument cluster case study was to see whether the ontology extraction approach presented in section 2 still works for large documents and whether the amount of manual work necessary for the extraction is still justifiable.

Figure 8 illustrates that ontology extraction worked for this case study as well: Figure 8 shows an excerpt of the extracted ontology. It shows the top ontology class (`kaon:Root`), its subclasses (actual ontology classes) and relations between them. Colors and arrows have the same meaning as in Figure 7: Variable–width lines denote "is-a"–relations, the thin end pointing to the more general concept and the thick end pointing to the more special one. White boxes denote concepts and classes of concepts, dark boxes are properties (associations) connecting concepts. Arrows are directed in such a way that a property and two concepts that it connects build a sensible sentence, if read in the arrow direction. For example, in Figure 8 one can read:

- Can_bus transfers input_signals
- Instrument_cluster receives input_signals
- Instrument_cluster contains display
- ...

The other goal of the instrument cluster case study was testing the scalability of the approach. During this case study were extracted:

- 123 concepts and concept classes, organized in 13 top–level classes and further subclasses
- 61 associations between different concepts

Additionally to the extraction of concepts and associations inconsistencies in term usage were discovered and corrected. The time cost of 5 working days seems justifiable for an 80–page document, given that inconsistencies were detected and corrected and a domain ontology was constructed.

5 Summary

This paper presented an approach for extraction of domain ontology from text documents. This section gives an overview of the complete approach and of the produced results.

5.1 Summary of the Approach

The ontology extraction approach consists of several steps, most important of which were introduced in Section 2. The following list gives an overview of the whole approach and shows which steps are performed completely automatically and which ones require human interaction.

1.	Format the text (one sentence per line)	partially automatic
2.	Tag each word (Part–of–Speech)	automatic
3.	Parse the tagged text	automatic
4.	Extract predicates and their arguments	automatic
5.	Build concept clusters	automatic
6.	Look for cluster intersections and build a taxonomy	**interactive**
7.	Transfer the taxonomy to the associations mining tool	partially automatic
8.	Look for potential associations	automatic
9.	Decide which associations are sensible	**interactive**

These steps correspond to the principal approach, they do not show detection and correction of inconsistencies. Inconsistencies are detected in interactive steps: concept clustering (Step 6) and decision about sensible associations (Step 9). After the correction of inconsistencies it is necessary to restart with the tagging (Step 2).

As one can see, some steps are marked as partially automatic, while others are interactive. The difference is fundamental: partially automatic steps are not completely automatic yet because of some technical problems. In the case of text formatting (Step 1), there are problems with incomplete or grammatically incorrect sentences that are often

present as bullet points in specification texts. In the case of taxonomy transfer (Step 7) it is a mere problem of tool integration.

For the steps that are marked as interactive complete automation is not desirable. As Goldin and Berry state [14], complete automation is not desirable if it could lead to information loss or wrong results. In the case of taxonomy building (Step 6) and association ascription (Step 9) inconsistencies can be found, which often manifest themselves in senseless term clusters or senseless associations. It is impossible for an automatic tool to decide which clusters/associations are sensible. Even after elimination of inconsistencies not every cluster intersection leads to a *sensible* larger cluster defining a more general concept and not every potential association is a sensible one. So, even for a perfectly consistent text a completely automatic tool would not be feasible. Goldin and Berry would also say that a completely automatic tool is not even desirable. This tool interactivity achieves one of the most important goals of document analysis and validation: detection of terminology inconsistencies.

Although the approach *does require* manual intervention, this cannot be seen as its weakness: Manual intervention leads to better document validation, which is itself as important as terminology extraction.

5.2 Evaluation of the Extraction Results

This subsection evaluates the results of the ontology extraction. As the criterion for the evaluation it is sensible to take *completeness for concepts* ("were all the concepts extracted?"). It does not make sense consider completeness for associations because associations are not explicitly defined in text. It does not also make sense to consider correctness (in the sense "are all the extracted concepts/associations relevant?"), neither as applied to concepts nor as applied to associations. Correctness evaluation makes no sense for concepts, as long as the analyst does not invent terms, but only extract concepts from text. It makes also no sense to evaluate correctness of the extracted associations because every single proposed association is checked manually before it is included into the ontology. So, the associations that are present in the final model are per definition correct from the human analyst's point of view.

Completeness evaluation for the steam boiler case study is easy: The steam boiler specification explicitly defines following concept classes: hardware components, messages, operation modes and failures. As for hardware concepts, all but two were extracted. The approach failed to extract "`operator desk`" and "`message transmission system`". These concepts are mentioned only once in the document. Their role is not further specified. A human reader would extract these two concepts, but would have to guess how they interact with other components. This point can be seen both as a weakness of the extraction technique and as an omission in the document. As for other concept classes (messages, operation modes and failures), the approach succeeded in extracting all the concepts belonging to these classes.

Evaluation of completeness of term extraction for the instrument cluster case study is more difficult because the document does not list concepts and concept classes explicitly. Ad hoc, by skim reading the document, one can identify following concepts: instrument cluster, rev meter, speedometer, indicator lights, engine control light, display, ignition key, radio, ..., that are all present in the extracted model. One can also easily identify some messages and technical parameters, like default pointer positions for dials. Nevertheless, for proper evaluation of completeness it is unwise to rely on such a comparison. Either a domain expert that could evaluate completeness of the extracted model directly or an extraction tool that *guarantees* that all the con-

cepts are extracted is necessary for proper evaluation. In future research it could make sense to try to compare the extraction results of the presented approach with those of AbstFinder [14] and with those produced by a domain expert via manual document analysis.

5.3 Lessons Learned

In the first case study, presented in Section 3, the goal was to see the actual applicability of the approach. For this purpose a relatively small document was chosen, consisting of only 6 pages. On the basis of this first case study the principal applicability of the approach was shown, as long as the text is consistent in term usage. Inconsistencies were detected on the basis of strange–looking term clusters and associations. Detection of inconsistencies is a part of document validation. Furthermore, it focuses the attention of the stakeholders and requirements engineers on terminology problems and helps to use a consistent terminology throughout the whole document.

Terminology inconsistencies can be eliminated only manually, which was not a problem for a short text. However, this gave rise to the question whether the approach is applicable to larger documents.

The goal of the second case study, presented in Section 4, was testing scalability of the approach. For this case study just one iteration of inconsistency correction was necessary, as opposed to several iterations in the first case study because the writing rules produced in the first case study could be used. The overall time cost for both inconsistency elimination and ontology building amounted to 5 working days. Given the fact that mere skim reading of the document took almost one working day, the total of 5 days seems justifiable.

Summing up, I can say that the presented approach is promising for requirements engineering. It can be used in RE–process both for quality assurance on the document level and for bridging the gap between documents and domain–specific ontology.

Acknowledgements

Here I want to thank David Faure, Claire Nédellec, Helmut Schmid, Alexander Pretschner, Jan Philipps, Franz Huber, Maria Spichkova, Alexander Ziegler, Markus Pister, Jewgenij Botaschanjan, Nadine Heumesser, Frank Houdek, Isabel John, Oscar Slotosch, Stefan Berghofer, Martin Deubler, Norbert Diernhofer, Ulrike Hammerschall, Jan Jürjens, Maurice Schoenmakers, Martin Strecker, Martin Wildmoser, Guido Wimmel, Michael Meisinger, Stefan Wagner, Iouri Riabov and Prof. Manfred Broy for their feedback and invaluable advice during the work.

I also want two thank two anonymous referees who provided a lot of comments and suggestions.

References

[1] ABBOTT, R. J. Program design by informal English descriptions. *Communications of the ACM 26*, 11 (1983), 882–894.

[2] ABRIAL, J.-R., BÖRGER, E., AND LANGMAACK, H. The steam boiler case study: Competition of formal program specification and development methods. In *Formal Methods for Industrial Applications* (1996), J.-R. Abrial, E. Borger,

and H. Langmaack, Eds., vol. 1165 of *LNCS*, Springer–Verlag. http://www.informatik.uni-kiel.de/~procos/dag9523/dag9523.html.

[3] BEN ACHOUR, C. Linguistic instruments for the integration of scenarios in requirement engineering. In *Proceedings of the Third International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'97)* (Barcelona, Catalonia, June 16-17 1997), P. R. Cohen and W. Wahlster, Eds.

[4] BERRY, D. Natural language and requirements engineering - nu?, 2001. http://www.ifi.unizh.ch/groups/req/IWRE/papers&presentations/Berry.pdf, accessed 09.01.2003.

[5] BIES, A., FERGUSON, M., KATZ, K., AND MACINTYRE, R. Bracketing guidelines for treebank ii style penn treebank project, 1995. http://www.cis.upenn.edu/~treebank/home.html.

[6] BREITMAN, K. K., AND SAMPAIO DO PRADO LEITE, J. C. Ontology as a requirements engineering product. In *Proceedings of the 11th IEEE International Requirements Engineering Conference* (2003), IEEE Computer Society Press, pp. 309–319.

[7] BUHR, K., HEUMESSER, N., HOUDEK, F., OMASREITER, H., ROTHERMEHL, F., TAVAKOLI, R., AND ZINK, T. DaimlerChrysler demonstrator: System specification instrument cluster. http://www.empress-itea.org/deliverables/D5.1_Appendix_B_v1.0_Public_Version.pdf.

[8] CHEN, P. English sentence structure and entity-relationship diagram. *Information Sciences 1*, 1 (May 1983), 127–149.

[9] COLLINS, M. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[10] FABBRINI, F., FUSANI, M., GNESI, S., AND LAMI, G. The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In *26th Annual NASA Goddard Software Engineering Workshop* (Greenbelt, Maryland, 2001), IEEE Computer Society, pp. 97–105.

[11] FAURE, D., AND NÉDELLEC, C. ASIUM: Learning subcategorization frames and restrictions of selection. In *10th European Conference on Machine Learning (ECML 98) – Workshop on Text Mining* (Chemnitz Germany, April 1998 1998), Y. Kodratoff, Ed.

[12] FUCHS, N. E., SCHWERTEL, U., AND SCHWITTER, R. Attempto Controlled English (ACE) language manual, version 3.0. Tech. Rep. 99.03, Department of Computer Science, University of Zurich, August 1999. http://www.ifi.unizh.ch/attempto/publications/papers/ace3_manual.pdf, accessed 21.05.2004.

[13] GERVASI, V., AND NUSEIBEH, B. Lightweight validation of natural language requirements: a case study. In *4th International Conference on Requirements engineering* (2000), IEEE Computer Society Press, pp. 140–148.

[14] GOLDIN, L., AND BERRY, D. M. AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. *Automated Software Eng. 4*, 4 (1997), 375–412.

[15] LAPPIN, S., AND LEASS, H. J. An algorithm for pronominal anaphora resolution. *Comput. Linguist. 20*, 4 (1994), 535–561.

[16] MAEDCHE, A., AND STAAB, S. Discovering conceptual relations from text. In *ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence* (Berlin, 2000), W.Horn, Ed., IOS Press, Amsterdam, pp. 321–325.

[17] MICH, L., FRANCH, M., AND NOVI INVERARDI, P. Market research on requirements analysis using linguistic tools. *Requirements Engineering 9*, 1 (2004), 40–56.

[18] MIMA, H., ANANIADOU, S., AND NENADIĆ, G. The ATRACT workbench: Automatic term recognition and clustering for terms. In *Text, Speech and Dialogue, 4th International Conference* (Želená Ruda, Czech Republic, September 2001), vol. 2166 of *LNAI*, Springer, pp. 126–133.

[19] MITKOV, R. *Anaphora Resolution*. Longman, 2002.

[20] NATT OCH DAG, J., REGNELL, B., CARLSHAMRE, P., ANDERSSON, M., AND KARLSSON, J. A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering 7*, 1 (2002), 20–33.

[21] NENADIĆ, G., SPASIĆ, I., AND ANANIADOU, S. Automatic discovery of term similarities using pattern mining. In *Proceedings of CompuTerm 2002* (Taipei, Taiwan, 2002), pp. 43–49.

[22] PORTER, M. An algorithm for suffix stripping. *Program 14*, 3 (1980), 130–137. `http://www.tartarus.org/~martin/PorterStemmer/`, accessed 14.07.2003.

[23] RATNAPARKHI, A. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, Institute for Research in Cognitive Science, University of Pennsylvania, 1998.

[24] ROLLAND, C., AND BEN ACHOUR, C. Guiding the construction of textual use case specifications. *Data & Knowledge Engineering Journal 25*, 1–2 (March 1998), 125–160.

[25] RUPP, C. *Requirements-Engineering und -Management Professionelle, iterative Anforderungsanalyse für die Praxis*, second ed. Hanser–Verlag, 05 2002. ISBN 3-446-21960-9.

[26] RYAN, K. The role of natural language in requirements engineering. In *Proceedings of IEEE International Symposium on Requirements Engineering* (1992), IEEE Computer Society Press, pp. 240–242.

[27] SAEKI, M., HORAI, H., AND ENOMOTO, H. Software development process from natural language specification. In *Proceedings of the 11th international conference on Software engineering* (1989), ACM Press, pp. 64–73.

[28] SRIKANT, R., AND AGRAWAL, R. Mining generalized association rules. *Future Generation Computer Systems 13*, 2–3 (1997), 161–180.

[29] VADERA, S., AND MEZIANE, F. From English to formal specifications. *The Computer Journal 37*, 9 (1994), 753–763.