Natural Language Processing: Mature Enough for Requirements Documents Analysis?

Leonid Kof

Technische Universität München, Fakultät für Informatik,
Boltzmannstr. 3, D-85748 Garching bei München, Germany
kof@in.tum.de

Abstract. Requirements engineering is the Achilles' heel of the whole software development process, because requirements documents are often inconsistent and incomplete. Misunderstandings and errors of the requirements engineering phase propagate to later development phases and can potentially lead to a project failure.

A promising way to overcome misunderstandings is to extract and validate terms used in requirements documents and relations between these terms. This position paper gives an overview of the existing terminology extraction methods and shows how they can be integrated to reach a comprehensive text analysis approach. It shows how the integrated method would both detect inconsistencies in the requirements document and extract an ontology after elimination of inconsistencies. This integrated method would be more reliable than every of its single constituents.

1 Ontology as a Requirements Engineering Product

Requirements engineering is the very first stage of any software project. This stage is extremely important, because requirements engineering should ensure that the specification of the product to be built meets customer's wishes. (Are we building the *right* product?) The goal of the later development stages, to the contrary, is to ensure that a product is being built correctly *with respect to the specification*, produced in the RE phase. So, requirements engineering errors either potentially lead to project failure or must be corrected in later phases, which is much more expensive than correction in the RE phase.

It is often believed that RE errors are due to forgotten or misinterpreted requirements. Praxis shows, however, that misunderstandings come into play much earlier: the same seemingly unambiguous word used in the requirements document can be interpreted in different ways. Obviously, misinterpretation of concept meanings is fatal for requirements engineering.

Zave and Jackson [1] give an example of such a concept misinterpretation. This example handles a hypothetical university information system and the definitions of a "student" and the binary relation "enrolled" for this system:

A. Montoyo et al. (Eds.): NLDB 2005, LNCS 3513, pp. 91–102, 2005.

Able: Two important basic types are *student* and *course*. There is also a binary relation *enrolled*. If types and relations are formalized as predicates, then

$$\forall s\ \forall c\ (enrolled(s, c) \Rightarrow student(s) \wedge course(c)).$$

Baker: Do only students enroll in courses? I don't think that's true.
Able: But that's what I mean by *student*!

This example shows that domain concepts must be precisely defined and that a simple glossary (term list) is insufficient. A more appropriate definition of domain concepts and relations between them would be an *ontology*.

According to Tom Gruber an ontology is a *specification of a conceptualization*[1]. In the context of this paper the ontology is defined as a taxonomy (term hierarchy), enriched by some general associations. The taxonomy itself consists of a set of terms and the "is-a"–relation.

The goal of this paper is to propose a comprehensive ontology extraction approach, based on existing methods. This proposed approach would both detect terminology inconsistencies in documents and extract an ontology from requirements documents. The paper shows how existing methods can be used in different stages of ontology building and how they would augment each other when integrated.

The paper is organized in the following way: Section 2 introduces ontology construction steps, without concrete recipes for particular steps. Section 3 presents an ontology construction approach implementing the steps introduced in Section 2. This approach was proven feasible in case studies. However, there is always room for improvement. Section 4 gives an overview of other existing text analysis approaches in RE and sketches their possible place in the ontology construction process. Section 5 integrates the text analysis methods into the comprehensive ontology building process, providing both detection of terminology inconsistencies and ontology extraction. Section 6 summarizes the integrated proposal introduced in Section 5.

2 Ontology Construction Steps

Ontology was introduced in artificial intelligence as a communication means for intelligent agents. Now it was recognized as a universal means to communicate concept dependencies. As software development involves experts from different disciplines, an ontology is also a good means to establish a common language in a software project.

Breitman and Sampaio do Prado Leite [2] see an application ontology as one of the products of the requirements engineering activity. They list in their paper several ontology construction methodologies. The listed methodologies all share the same basic steps, shown in Figure 1[2]. These steps include, apart from

[1] Cited after `http://www-ksl.stanford.edu/kst/what-is-an-ontology.html`
[2] "LEL", used in the figure, means "Language Extended Lexicon", a notation used in [2]

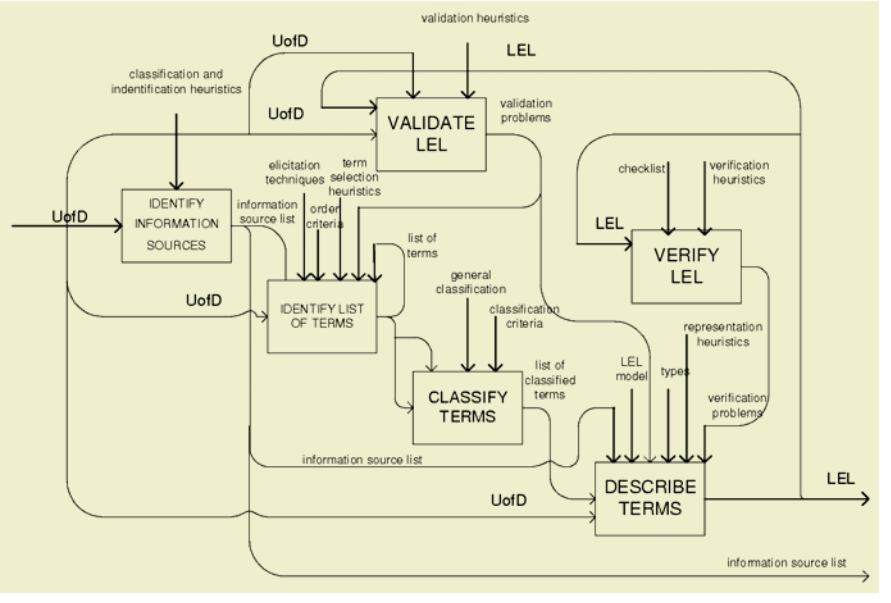

Fig. 1. Steps of Ontology Construction [2].

validation and verification, identification of information sources, identification of the list of terms, classification of the terms and their description.

All the approaches listed by Breitman & Sampaio do Prado Leite [2] are rather abstract in the sense that they do not specify *how* to identify information sources, *how* to classify terms, and so on. When ontology construction is considered as a phase of the requirements engineering process, identification of the information sources is simple: the primary information source is the requirements document. Other steps of ontology construction can be done by analyzing this document. The next section introduces a text analysis approach that performs these ontology building steps.

3 Ontology Building by Means of Text Analysis

This section is a very short summary of [3]. It shows how text analysis can be applied to the previously introduced ontology construction steps. Figure 2 shows single steps of the ontology extraction approach. The steps correspond to those shown in Figure 1: "parsing and subcategorization frames extraction" corresponds to the identification of the term list, "term clustering & taxonomy building" and "association mining" correspond to term classification. Term description, validation and verification have to be done manually and are not shown in Figure 2.

The approach shown in Figure 2 is interactive, i.e. some decisions must be made by the analyst. Interactivity is important because fully automatic extraction procedure could not detect inconsistencies, that are extremely common in

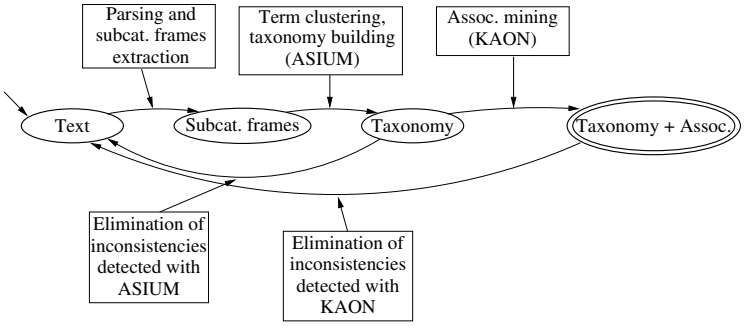

Fig. 2. Ontology Building Procedure [3].

requirements documents. The overall process of ontology construction consists of four steps: term extraction, term clustering, taxonomy building (as a cluster hierarchy) and relations mining.

Extraction of terms from the requirements text: To extract terms, each sentence is parsed and the resulting parse tree is decomposed. Noun phrases that are related to the main verb of the sentence are extracted as domain concepts. For example, from the sentence "`The control unit sends an alarm message in a critical situation`" "`send`" is extracted as the main verb, "`control unit`" as the subject and "`alarm message`" as the direct object.

Term clustering: The second phase clusters related concepts. Two concepts are considered as related and put in the same cluster if they occur in the same grammatical context. For example, if the requirements document contains two sentences like

1. "The control unit sends an alarm message in a critical situation"
2. "The measurement unit sends measurements results every 5 seconds",

the concepts "`control unit`" and "`measurement unit`" are considered as related, as well as "`alarm message`" and "`measurements results`".

Taxonomy building: Concept clusters constructed in the previous step are used for taxonomy construction by joining intersecting clusters to larger clusters. The resulting larger clusters represent more general concepts.
For example, the basic clusters {`alarm message`, `measurements results`} and {`control message`, `measurements results`} can be joined into the larger cluster:

{`alarm message`, `control message`, `measurements results`}

representing possible messages. The tool ASIUM [4] is used both to cluster terms and to build a taxonomy.
During this step the terminology is validated with respect to synonyms[3]. Synonyms are often contained in the same cluster. For example, if a cluster contains both "`signal`" and "`message`", the analyst can identify them as synonyms.

[3] different names for the same concept

Associations/relations mining: There is a potential association between two concepts if they occur in the same sentence. Each potential association again has to be validated by the requirements engineer. Note that the validation of the association proposed by the association mining tool automatically implies a validation of the requirements document. If the tool suggests an association that *can not* be valid (i.e., a pair containing completely unrelated concepts), then we have detected an evidence that the requirements document contains some inconsistent junk that must be eliminated (see [5] for an in–depth treatment of association mining).

The remainder of this section presents term extraction in more detail, because these details will be important for the integrated approach as well (see Sections 4 and 5).

3.1 Term Extraction

Term extraction bases on parsing of each sentence and the representation of every sentence as a parse tree. To build a parse tree, the parser by Michael Collins [6] was used. This parser provides for each parse tree node information about the head (most important) child. An example parse tree is shown in Figure 3. Meanings of the tags used in Figure 3 are introduced in the "Bracketing Guidelines for Treebank II Style Penn Treebank Project" by Bies et al. (`http://www.cis.upenn.edu/\~treebank/home.html`). In a nutshell, S marks complete sentences, VP marks verb phrases, VB marks verbs, MD modal verbs, NP marks noun phrases, PP marks prepositional phrases and NN marks nouns.

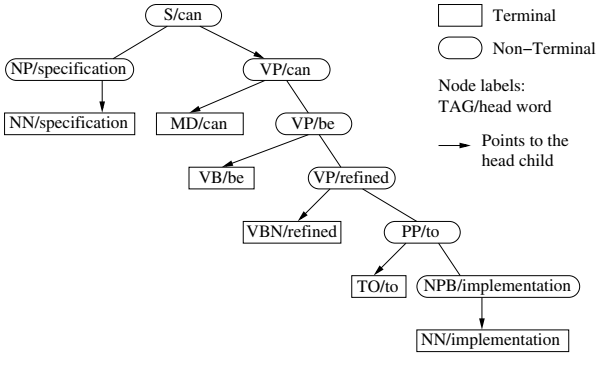

Fig. 3. Parse tree for "`Specification can be refined to implementation.`".

The term extraction algorithm extracts not only the terms, but also the predicates. The predicates will be used later for term clustering. 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 . . . ” (Figure 3). “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 3, and looks for sibling verb or verb phrase nodes. In the case of Figure 3 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 3 and traverses the parse tree to the left until it finds a noun phrase. In Figure 3 it finds “NP/specification”. Then it descends to the head child of the noun phrase, which is “NN/specification”. The objects are extracted in a similar way.

The algorithm described above extracts just concepts consisting of a single word. There is also an algorithm extension, extracting compound concepts like “failure of some unit”: It starts with the leaf concept node (“failure”) and goes up in the parse tree, looking for noun phrases and “of”–constructions. The extension is not presented here in more detail for the sake of brevity. See [3] for the detailed description.

The provided information about subjects and objects is used for term clustering, as explained above. It will also be used later for resolution of pronominal anaphora (see Subsection 4.1).

4 Alternative Text Analysis Approaches

The approach presented in the previous section was tested on several case studies [3] where it produced good results. However, there is still room for improvement. For example, the method presented above cannot cater for cross–sentence references and cannot extract terms that occur solely in grammatically incorrect sentences. The goal of this section is to show other existing approaches that would augment the original approach and produce even better results, when integrated.

The whole plethora of the developed text analysis methods can be classified in three categories. Ben Achour [7] classifies the linguistic methods as either lexical or syntactic or semantic. Lexical methods, as for example AbstFinder [8] 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. Subsection 4.1 describes it in more detail, in order that it can be used in the integrated approach in Section 5.

Syntactical approaches analyze sentence structure. They are more demanding to the grammatical correctness of the text than lexical approaches, but in

return they extract more information from the text. The approach presented in Section 3 is a syntactical one. Other syntactical approaches suggest other methods of term classification and clustering. These methods will be considered in Subsection 4.2.

Semantical approaches promise more than the other two classes: They interpret each sentence as a logical formula. However, semantical approaches, as for example [9], are extremely fragile, as they require firm sentence structure. For this reason they are barely applicable to real world requirements documents and will not be considered further in this paper.

4.1 Lexical Approaches: Term Identification

The great advantage of the lexical methods as compared to syntactical and semantical ones is their robustness. This robustness stems form the fact that they consider each sentence just as a character sequence. AbstFinder by Goldin and Berry [8] is based on this idea: It finds common character subsequences in every sentence pair. For example, consider the following two sentences (taken from one of the case studies discussed in [3]):

> The steam-boiler is characterized by the following elements:

and

> Above m2 the steam-boiler would be in danger after five seconds, if the pumps continued to supply the steam-boiler with water without possibility to evacuate the steam.

They contain a common character sequence "steam-boiler", so "steam-boiler" is identified as a potential domain concept. AbstFinder is interactive, so the requirements analyst may decide which of the extracted character sequences really represent domain–specific terms.

Anaphora Resolution: The term extraction method introduced above assumes that the terms are explicitly present in the text. However, the usage of pronouns is very frequent, which undermines this assumption. For example, in the following two sentences, taken from the one of the case studies from [3], the second sentence does not name the received message explicitly:

> The program enters a state in which it waits for the message steam-boiler-waiting to come from the physical units. As soon as this message has been received the program checks whether the quantity of steam coming out of the steam-boiler is really zero.

So, AbstFinder would not identify "`message steam-boiler-waiting`" as a common substring of the two sentences. Usage of pronouns poses problems to the term extraction approach based on parse trees (introduced in Subsection 3.1) as well: It would extract just "this message" as a subject of "received" from the second sentence.

This problem can be solved by the means of anaphora resolution [10]. Resolution of pronominal anaphora would identify "`this message`" in the second sentence with "`message steam-boiler-waiting`" in the first one. An additional advantage of applying anaphora resolution would be detection of referential ambiguities. A referential ambiguity, according to the definition by Kamsties et al. [11] "is caused by an anaphora in a requirement that refers to more than one element introduced earlier in the sentence or in a sentence before". For example, in the sentences

```
The controller sends a message to the pump.
It acknowledges correct initialization.
```

"it" can refer both to the pump and to the controller and to the message. Explicit anaphora resolution would disambiguate this reference. In the case of wrong resolution it would make the referential ambiguity visible.

Anaphora resolution, as presented by Judita Preiss [10], depends on the extraction of grammatical roles. The term extraction algorithm, presented in Subsection 3.1, extracts subjects and objects from each sentence, so it can be used as preprocessor for anaphora resolution.

4.2 Syntactical Approaches: Clustering and Taxonomy Building

Syntactical approaches make use of sentence structure to classify the extracted terms. The definition of a cluster, used in the tool ASIUM [4] described before, is very simple: A cluster is built by all the subjects or all the objects of some verb. It is also possible to use other sentence information for the classification purpose. Nenadić et al. [12] introduce following definitions of related terms:

Contextual Similarity of two terms measures the number of common and different contexts for the two terms whose similarity should be determined. For this measure the context is defined as a sequence of particular words with their Part–of–Speech (POS) tags (noun, verb, etc.) occurring in the sentence before and after the term. It is up to the analyst to use all the context words and tags or to define some words or word classes (adjectives, conjunctions, ...) as irrelevant and filter them out. It depends on the text domain which contexts (POS sequences, lexica, etc.) provide better term clustering. For this similarity measure to work, the requirements analyst has to decide which contexts to use. This decision can rely on the quality measure for contexts, also introduced by Nenadić et al. [12].

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". Lexical heads are provided by the parser, used in Subsection 3.1. So, lexical similarity can be measured on the basis of parse subtrees for each term, extracted in Subsection 3.1.

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, a linear combination of the three above measures is calculated. Terms with high net similarity can be grouped to clusters. Subsequent taxonomy building on the basis of term clusters and cluster intersections can be done in exactly the same way as in Section 3.

5 Integrated Ontology Building Approach

Previous section showed that there are many methods potentially able to improve the original ontology extraction approach from Section 3. This section shows how all the approaches can be integrated. The goal of the integration is to join the strengths and to hide the weaknesses of the isolated methods.

Figure 4 shows an overview of the proposed integrated approach. Just as the approach shown in Figure 2, it starts with the specification text, written in natural language, and extracts an ontology. However, it consists of much more steps and extracts more information from the text. The remainder of this section presents each step in detail.

Parsing and anaphora resolution: The goal of this very first step is to get rid of pronominal cross–sentence references. Anaphora resolution is necessary for the later steps, because it replaces pronouns by fully-fledged terms, so that these terms instead of pronouns can be extracted.
The results of anaphora resolution should be examined by the domain expert. It is possible that some anaphora be resolved incorrectly, either due

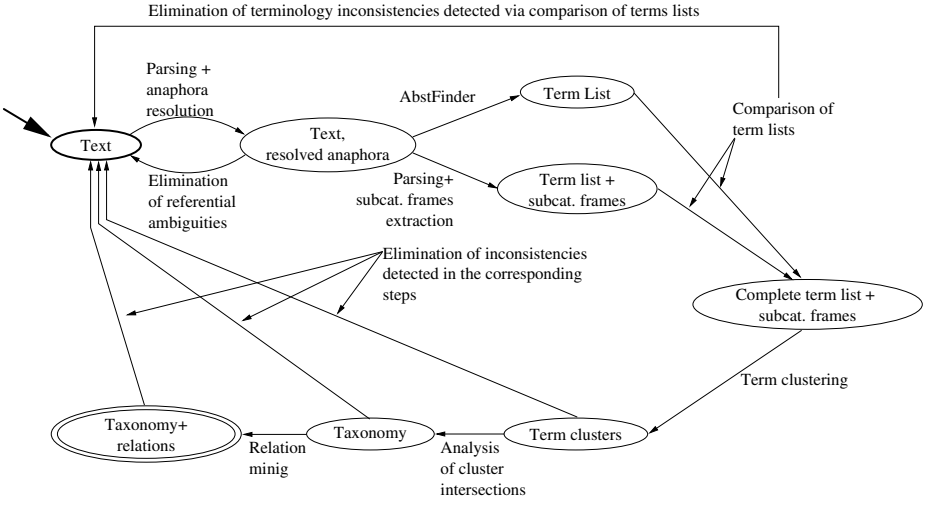

Fig. 4. Integrated Ontology Extraction Approach.

to referential ambiguity (several possibilities to resolve an anaphora) or due to deficiencies of the used parser or resolution algorithm. The manual revision of the resolution results would make sure that the terms are substituted correctly.

AbstFinder: AbstFinder considers the sentences just as character sequences and extracts character sequences that are common for at least two sentences. The requirements analyst may decide which sequences really represent a term. The extraction of *common* character sequences explains the necessity for anaphora resolution: The resolved pronouns become sensible character sequences.

Parsing and subcategorization frames extraction: The text with resolved anaphora should be parsed anew. (Actually, it is necessary to parse only sentences that were changed because of anaphora resolution.) Then, the terms can be extracted using the technique described in Section 3.1. In this way the subcategorization frames (verbs and their arguments) are extracted. Verbs can be used later to cluster terms.

Comparison of term lists: Neither AbstFinder nor the extraction of subcategorization frames can *guarantee* the extraction of *all* the terms: AbstFinder extracts terms that occur at least twice in the text, and subcategorization frames extraction can extract terms that occur in grammatically correct sentences only.

Comparison of the two extracted term list can give important information about the text: It shows which terms are often (at least twice) used in the text, but solely in grammatically incorrect sentences. It shows also which terms are used just once. If this term list comparison discovers some omissions in the document, it is up to the requirements analyst to change the text to correct the flaws.

Term clustering: To build a taxonomy, it is necessary to find related terms first. The criteria that can be used to cluster related terms were introduced in Subsection 4.2: contextual, lexical and syntactic term similarity. The weights of each of the similarity measures can vary depending on the analyzed text. The produced term clusters should be examined by the requirements analyst. Unrelated terms put in the same cluster usually signalize either a terminology inconsistency or inaccurate phrasing somewhere in the text. The sentences using inconsistent terminology can be found by simple text search: for the inconsistent cluster it is known which terms, used in which context, caused the cluster inconsistency. So, it is sufficient to look for the sentences containing the term in the corresponding context. The detected inconsistencies should be corrected before the analysis continues.

It could be argued that the clustering heuristic itself is a potential source of cluster inconsistencies. However, the clustering heuristic presented by Nenadić et al. [12], that shall be used in the integrated approach, can be trained on the particular domain, which minimizes this inconsistency source.

Taxonomy building: To build a taxonomy, it is necessary to determine, which clusters are related. This can be done for example by analysis of cluster intersections and joining them to larger clusters, representing more general concepts. This step is the same as in the simpler ontology extraction approach, described in Section 3.

Relation mining: In the last step the taxonomy is augmented by more general relations. This step is exactly the same as in the simpler ontology extraction approach, described in Section 3: There is a potential association between two concepts if they occur in the same sentence. Each potential association again has to be validated by the requirements engineer. The validated associations are absorbed into the ontology.

The proposed approach requires manual intervention. However, manual intervention is necessary to detect inconsistencies. As Goldin and Berry state [8], complete automation is not desirable if it could lead to information loss or wrong results. Thus, interactivity is not a weakness but an important feature of the proposed approach. Due to this interactivity the extracted ontology is validated *by construction*. The result of the whole procedure is a validated application domain ontology *and* a corrected textual specification, free from terminology inconsistencies. The corrected textual specification is itself as important as ontology extraction.

6 Conclusion

Requirements engineering is a non–trivial task and the proposed approach is not able to solve all the requirements engineering problems. However, it tackles an extremely important step, namely establishing a common language for the stakeholders. An application domain ontology serves as such a common language. After the construction of this common language it is also important to validate the results, which is also achieved by the proposed approach.

This paper showed how the existing text analysis approaches aiming at ontology extraction can be combined to produce better results than each approach on its own. Drawbacks of every single technique are compensated in the proposed integrated approach by complementary analysis methods: Extraction of subcategorization frames works for grammatically correct sentences only, but it can be augmented by AbstFinder, analyzing character sequences. For AbstFinder to provide better results, it is necessary to replace pronouns by the terms they refer to, and so on. It makes no sense to list here all the interdependencies: this would lead to complete repetition of the previous section.

The final claim of the paper is that natural language processing is mature enough to be applied to ontology extraction in the context of requirements engineering, in spite of the necessary manual intervention. Automation is possible for every ontology construction step and a comprehensive approach is "just" a matter of integration of the existing techniques. Surely the integrated approach needs validation on case studies. The simpler approach has already been validated [3] and, obviously, the integrated approach can be at least as good as the simpler one.

Acknowledgements

Here I want to thank David Faure, Claire Nédellec, Helmut Schmid and Goran Nenadić for their insightful cooperation during the work. I also want to thank the anonymous reviewers who helped to improve the paper.

References

1. Zave, P., Jackson, M.: Four dark corners of requirements engineering. ACM Trans. Softw. Eng. Methodol. **6** (1997) 1–30
2. Breitman, K.K., Sampaio do Prado Leite, J.C.: Ontology as a requirements engineering product. In: Proceedings of the 11th IEEE International Requirements Engineering Conference, IEEE Computer Society Press (2003) 309–319
3. Kof, L.: An Application of Natural Language Processing to Domain Modelling – Two Case Studies. International Journal on Computer Systems Science Engineering **20** (2005) 37–52
4. Faure, D., Nédellec, C.: ASIUM: Learning subcategorization frames and restrictions of selection. In Kodratoff, Y., ed.: 10th European Conference on Machine Learning (ECML 98) – Workshop on Text Mining, Chemnitz Germany (1998)
5. Maedche, A., Staab, S.: Discovering conceptual relations from text. In W.Horn, ed.: ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, IOS Press, Amsterdam (2000) 321–325
6. Collins, M.: Head-Driven Statistical Models for Natural Language Parsing. PhD thesis, University of Pennsylvania (1999)
7. Ben Achour, C.: Linguistic instruments for the integration of scenarios in requirement engineering. In Cohen, P.R., Wahlster, W., eds.: Proceedings of the Third International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'97), Barcelona, Catalonia (1997)
8. Goldin, L., Berry, D.M.: AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. Automated Software Eng. **4** (1997) 375–412
9. Fuchs, N.E., Schwertel, U., Schwitter, R.: Attempto Controlled English (ACE) language manual, version 3.0. Technical Report 99.03, Department of Computer Science, University of Zurich (1999)
10. Preiss, J.: Choosing a parser for anaphora resolution. In Cohen, P.R., Wahlster, W., eds.: DAARC 2002, 4th Discourse Anaphora and Anaphor Resolution Colloquium, Lisbon, Edições Colibri (2002) 175–180
11. Kamsties, E., Berry, D.M., Paech, B.: Detecting ambiguities in requirements documents using inspections. In: Workshop on Inspections in Software Engineering, Paris, France (2001) 68 –80
12. Nenadić, G., Spasić, I., Ananiadou, S.: Automatic discovery of term similarities using pattern mining. In: Proceedings of CompuTerm 2002, Taipei, Taiwan (2002) 43–49