What Did You Mean?
Facing the Challenges of User-generated Software Requirements

Michaela Geierhos, Sabine Schulze and Frederik Simon Bäumer
Heinz Nixdorf Institute, University of Paderborn, Fürstenallee 11, D-33102 Paderborn, Germany

Keywords: User-generated Natural Language Requirements, Interactive Requirement Disambiguation and Completion.

Abstract: Existing approaches towards service composition demand requirements of the customers in terms of service templates, service query profiles, or partial process models. However, addressed non-expert customers may be unable to fill-in the slots of service templates as requested or to describe, for example, pre- and post-conditions, or even have difficulties in formalizing their requirements. Thus, our idea is to provide non-experts with suggestions how to complete or clarify their requirement descriptions written in natural language. Two main issues have to be tackled: (1) partial or full inability (incapacity) of non-experts to specify their requirements correctly in formal and precise ways, and (2) problems in text analysis due to fuzziness in natural language. We present ideas how to face these challenges by means of requirement disambiguation and completion. Therefore, we conduct ontology-based requirement extraction and similarity retrieval based on requirement descriptions that are gathered from App marketplaces. The innovative aspect of our work is that we support users without expert knowledge in writing their requirements by simultaneously resolving ambiguity, vagueness, and underspecification in natural language.

1 INTRODUCTION

In the Collaborative Research Centre "On-The-Fly Computing", we develop techniques and processes for the automatic ad-hoc configuration of individual service compositions that fulfill customer-specific requirements[1]. Upon request, suitable basic software and hardware services available on world-wide markets have to be automatically discovered and composed. For that purpose, customers have to provide service descriptions.

Existing approaches towards service composition demand requirements of the customers in terms of service templates, service query profiles, or partial process models. "The requirements include: list of sub-services, inputs, outputs, preconditions and effects [...] of the sub-services, and the execution order of these sub-services" (Bhat et al., 2014). Moreover, others force the customers to specify their service requirements by using a formal specification language. However, addressed non-expert customers, may be unable to fill-in the slots of service templates as requested, or to describe, for example, pre- and post-conditions, or even have difficulties in formalizing their requirements. Furthermore, we have to deal with partial requirements because "the person who writes the requirements might forget to consider relevant concepts of the problem, either because [he or] she postpones their analysis, or because they are unclear and hard to specify, or because the input documents include too many concepts to consider them all" (Ferrari et al., 2014).

When non-experts get the opportunity to describe service requirements by using natural language (NL), they produce highly individual user-generated text. For this purpose, we develop strategies for the resolution of ambiguity, vagueness and incompleteness of specifications created by non-expert users. Our approach therefore analyzes (unrestricted) service requirement descriptions written in NL on the one hand and provides suggestions how to complete or clarify their initial service descriptions on the other hand.

This paper is organized as follows: In Section 2, we give a brief overview of related work and describe the challenges in natural language processing (NLP) for user-generated specifications in Section 3. Section 4 outlines our approach to automatically identify service requirements and how to counterbalance their fuzziness. We conclude in Section 5 and present possible directions of future work in Section 6.

[1]Refer to http://sfb901.uni-paderborn.de for more information.

2 RELATED WORK

There have already been studies mining unstructured texts represented in NL for software requirement specifications (Meth et al., 2013; Ferrari et al., 2014). Besides, the (in)completeness of requirements has already been an issue in NL requirement engineering (Ferrari et al., 2014; Espana et al., 2009; Menzel et al., 2010; Kaiya and Saeki, 2005; Génova et al., 2013). However, there is only one approach so far that generates automatic suggestions for NL requirement completion (Ferrari et al., 2014). Ferrari et al. (2014) propose the so-called Completeness Assistant Approach (CAR) that automatically suggests relevant term candidates and possible relations among terms in natural language to be used in the requirements. Unlike Ferrari et al. (2014), we use a domain ontology-based approach – similar to Kaiya and Saeki (2005) – for the NL requirement analysis. Domain ontologies are widely used in requirement engineering due to their significant advantages in the field of knowledge sharing and reuse (Bhat et al., 2014). In particular, they allow requirement engineers to analyze specifications with respect to the domain-specific application semantics and to detect incompleteness in requirement documents even though they do not support rigorous NLP techniques (Kaiya and Saeki, 2005). Unlike Ferrari et al. (2014) we focus not only on keyword extraction for requirement completion, but also on requirement dependencies within the discourse structure.

Moreover, we also consider ambiguities occurring in user-generated requirement specifications. Since the majority of requirements are often written in NL, they entail ambiguity (Yang et al., 2010a). However, some approaches focus on the prevention of ambiguities in requirement descriptions rather than resolving them (Fuchs and Schwitter, 1995; Boyd et al., 2005). Nevertheless, there exist also strategies to reduce and to eliminate ambiguity. Since we focus on interactive disambiguation techniques, we have to mention the works by Yang et al. (2010b) and Kiyavitskaya et al. (2008) providing user support to resolve ambiguities. For instance, Yang et al. (2010b) presented a tool that assists requirement engineers by highlighting ambiguous expressions in their requirement specifications. Kiyavitskaya et al. (2008) also pursue a similar approach. They developed a method for the identification and measurement of ambiguity in requirement documents written in NL. In particular, all sequences which were recognized as probably ambiguous in a certain sentence are shown to the user. This warning gives the user the opportunity to remove them and thus improve the initial requirement specifications. However, this approach does not cover semantic ambiguity since this requires deep language understanding which is beyond the scope of this tool.

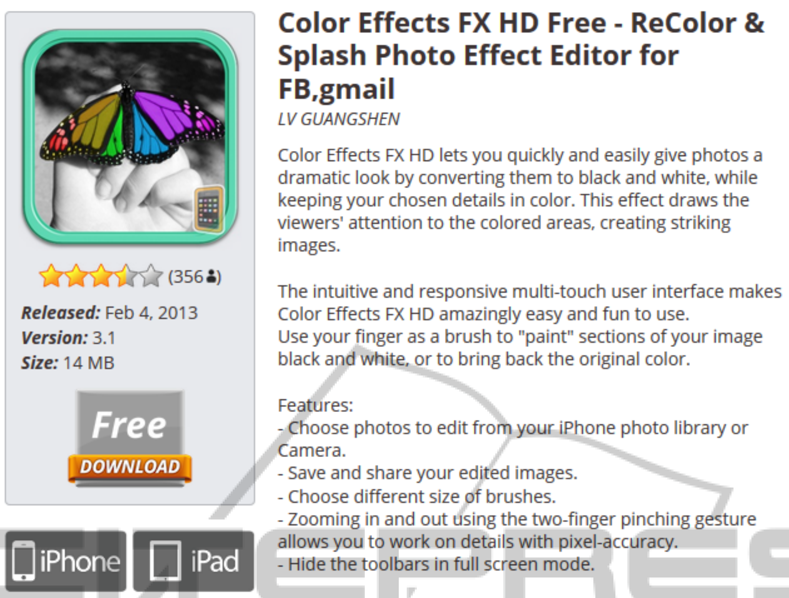
Discovering the sources for ambiguity is another research topic Gleich et al. (2010) concentrate on. Their tool provides "for every detected ambiguity [...] an explanation why the detection result represents a potential problem" (Gleich et al., 2010). In comparison to the other approaches, this technique is able to identify lexical, syntactic, semantic and pragmatic ambiguity which usually occur in NL (Gleich et al., 2010). By means of lexical and syntactic shallow parsing, they process the unstructured NL requirement documents (that also do not have any grammatical restrictions). Furthermore, they use the Ambiguity Handbook (Berry et al., 2003) and Siemens-internal guidelines for requirements writing as gold standard in order to detect ambiguities.

However, the target group of users that we address might not be able to resolve possible ambiguities on their own, even if some information is provided where to spot the ambiguity. Those customers have no prior knowledge of requirement engineering and need support in terms of disambiguated requirement descriptions. Besides, functional and non-functional requirements, system goals or design information may not be clearly distinguished by non-experts. They describe specifications in different ways than experts do. That's why we have to create a gold standard of non-expert service descriptions in order to detect ambiguity, vagueness, variability and underspecification in requirements written in NL.

3 CHALLENGES IN NLP

We have to face various problems when non-experts specify requirements by using NL. According to Sommerville (2010), there are (1) ambiguity, (2) variability and (3) vagueness in requirement descriptions which are illustrated by selecting examples from Figure 1.

1. If requirements are not expressed in a precise and unambiguous way, they may be misunderstood. (E.g., *"hide"* in Figure 1 means to make something invisible on a screen, but it can also mean *"hide and seek"* in a gaming context.)
2. Requirements specifications represented in NL are overflexible. One can say the same thing in completely different ways. (E.g., *"choose photos"* could also be expressed as *"select photos"*.)
3. Several different requirements may be expressed together as a single requirement but have to be separated for NLP. (E.g., *"save and share your*



Color Effects FX HD Free - ReColor & Splash Photo Effect Editor for FB,gmail

LV GUANGSHEN

Released: Feb 4, 2013
Version: 3.1
Size: 14 MB

Color Effects FX HD lets you quickly and easily give photos a dramatic look by converting them to black and white, while keeping your chosen details in color. This effect draws the viewers' attention to the colored areas, creating striking images.

The intuitive and responsive multi-touch user interface makes Color Effects FX HD amazingly easy and fun to use.
Use your finger as a brush to "paint" sections of your image black and white, or to bring back the original color.

Features:
- Choose photos to edit from your iPhone photo library or Camera.
- Save and share your edited images.
- Choose different size of brushes.
- Zooming in and out using the two-finger pinching gesture allows you to work on details with pixel-accuracy.
- Hide the toolbars in full screen mode.

Figure 1: Sample service requirement description written in natural language[2].

edited images" contains two requirement descriptions *"save your edited images"* and *"share your edited images"* in one sentence.)

Hence, ambiguity, underspecification and vagueness can lead to misunderstandings between customers and developers. This is also one of the main reasons why personalized software is not meeting user requirements (Murtaza et al., 2013). Nevertheless, the accurate capture of requirements from NL specifications is an essential step in the software development process because incorrect requirement definitions inevitably lead to problems in system implementation and design (Ilieva and Ormandjieva, 2005). The need to face the challenges in NLP of requirement descriptions becomes even more clear when considering that software requirements are commonly written in NL due to its expressiveness (Meth et al., 2013). In the following three sections, we discuss these issues for processing NL requirements in more detail.

3.1 Ambiguity

"Ambiguity is a pervasive phenomenon in natural language, and thus in natural language requirements documents" (Yang et al., 2011). It is the possibility of interpreting an expression in two or more distinct ways. If an expression is ambiguous, then there still remains uncertainty of its meaning or intention in a certain context. Because of that, ambiguity is problematic for computational semantics and for NLP, especially when their various interpretations have to be identified by algorithms. Even worst, ambiguous expressions can be misinterpreted by developers or custom service providers, and thus, can lead to providing customers with wrong software and/or hardware packages. In the requirement description above (cf. Figure 1), an example for an ambiguous word is *"brushes"*. Depending on the context, *"brush"* has different meanings. According to an English dictionary, it is most commonly used in the sense of sweeping with a broom. However, it can also be defined as a tool for painting (i.e. image editing).

[2]http://iosnoops.com/appinfo/x/597847507

3.2 Underspecification

Requirements written in NL are underspecified if features are omitted in underlying representations (Fabbrini et al., 2001). The concept of underspecification is particularly used to refer to cases in which requirements represented in NL do not bear an entire set of feature-values, and are thus compatible with a wide range of potential semantic interpretations. In addition, ambiguity can cause underspecification which leads to difficulties in formal representation and processing of service requirements. Therefore, "it is important to use computational representations that preserve the information available in ambiguous and underspecified expressions, without distorting or interpreting them incorrectly" (Loukanova, 2013). An example for underspecification shown in Figure 1 is *"share"* in the expression *"share your edited images"* because it is not clear via which channel the images can be shared. Possible options would be e-mail or social media platforms like Facebook.

3.3 Vagueness

Vagueness occurs when a phrase has a single meaning from grammatical point of view, but still leaves room for interpretation but, when considered as a requirement, leaves room for varying interpretations. "The system should react *as fast as possible* provides an example of such a vague phrase" (Gleich et al., 2010). It is important that vaguely expressions of requirements are detected. Further examples taken from Figure 1 are *"different size of brushes"'* and *"with pixel accuracy"* because it is not specified which sizes or what degree of accuracy are meant exactly. By timely detection, they can be either prevented, e.g., by interactions with the customers, or represented in a formal way without imposing misinterpretations or overinterpretations.

4 APPROACH

In the customers' descriptions of requirements for a certain domain-specific service, we first have to extract the relevant specifications from their input texts (cf. Section 4.1). Then, the user-generated requirements written in NL are analyzed for ambiguity, vagueness (cf. Section 4.2) and underspecification (Section 4.3) before providing suggestions how to improve or complete their initial requirements (cf. Section 4.4).

4.1 Ontology-based Requirement Extraction from Descriptions

The core topic of this task is the extraction of requirements from service descriptions represented in NL produced by customers as plain text in terms of full sentences or phrases (cf. Figure 2). The goal of this step is to discover and structure relevant domain-specific information about service specifications in the user-generated texts.

The decision criteria, determining what relevant information is, comes from domain-specific ontologies that underlie the service specifications[3] which determine the available, relevant information, coming from requirement descriptions represented in NL. Moreover, the requirement extraction is supported by domain-specific terms and rules ($\exists x$[image(x)$\wedge$save(x)$\wedge$share(x)]) to include further information types that have to be extracted. Thereby, we apply semantic-syntactic patterns in order to recognize concepts and relations within the requirement descriptions written in NL. These semantic-syntactic patterns are represented by local grammars. Local grammars have the capability to describe semantic-syntactic structures that cannot be formalized in electronic dictionaries. They are represented by directed acyclic graphs and implemented as finite state transducers (Geierhos, 2010). These transducers produce output in terms of semantic annotations (i.e. labels) for recognized requirements and evaluative expressions in the review texts. The grammar rules can be instantiated with high-frequent n-grams and then are generalized. For a comprehensive analysis of service descriptions with simultaneous pattern generation, optimal feature combinations can be automatically determined by means of machine learning techniques[4]. These patterns (i.e. local grammars) are then applied to enhance the requirement extraction process for a better data coverage.

In order to provide large training corpora from scratch, service descriptions from online App marketplaces (e.g. Google Play) are gathered. A data set of similar size cannot be collected in a short time only by user input. These texts contain various service descriptions from different domains and help us to cover different variations in the requirement texts (as described in Section 3). Besides they simultaneously serve as reference corpora (so-called test corpora) for the disambiguation and compensation of the user-generated underspecification within their service descriptions (cf. Section 4.3). After annotating a small amount of training data, we automatically induce rules (represented by local grammars) from this therefor built sub-corpus of service descriptions. Before applying those rules to the non-annotated part of our corpora for the instantiation of the corresponding service description templates, we have to abstract from the given (annotated) service requirement samples (Geierhos, 2010). When using semi-supervised machine learning techniques due to the small amount of annotated training data, we have to collect a large amount of unstructured service descriptions in order to get reliable results. But for the given input – individual specifications by non-experts – we have no access to a reference corpus. We therefore plan to simultaneously apply a bootstrapping approach based on predefined requirement patterns on the non-annotated user input. That way, we want to determine the most suitable extraction method for the given use case.

[3]For more information refer to http://sfb901.upb.de/sfb-901/projects/project-area-b/tools/service-specification-environment.html.

[4]There are other approaches, but they are out of scope of this paper.

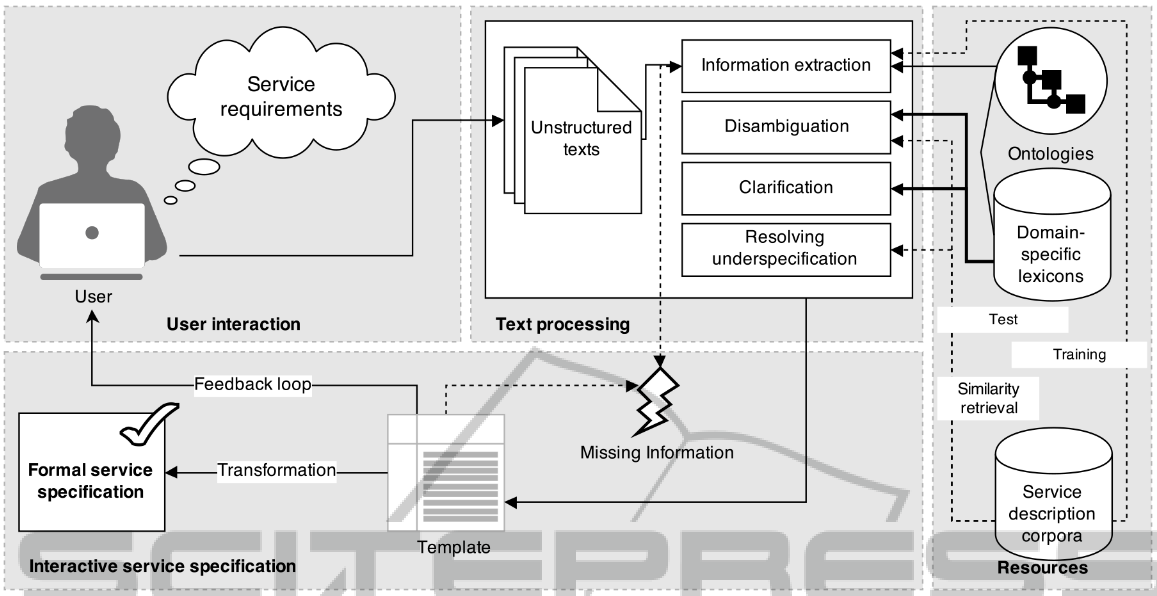


Figure 2: Pipeline for service requirement analysis using NLP

4.2 Disambiguation and Clarification

After providing requirement representations in terms of templates (cf. Figure 2), strategies for disambiguation and resolution of vagueness (i.e. clarification of what, e.g., *"different sizes"* means in Figure 1) in descriptions represented in NL have to be developed.

First of all, it is necessary to create test corpora out of service descriptions. We therefore have to manually annotate all ambiguous and/or vague expressions in the test data built from a few randomly-chosen service descriptions, before classifying the other occurrences by means of co-training. The degree of vagueness depends on the valence of the predicates and their unspecified arguments as well as on the imprecision of the term extensions.

When applying the local grammars originally developed for requirement extraction in Section 4.1 on the test corpora, we can detect free arguments ($\exists x$[image(x)$\wedge$save(x,_)$\wedge$share(x,_)]) in service descriptions using predicate-argument-tuples such as save(x,_) or share(x,_) if details are provided where to save or how to share (Choi et al., 2012). Apart from lexical ambiguity, unfilled argument positions are good indicators for syntactic ambiguity and vagueness because of imprecise information. This way, it can be automatically identified which arguments (especially their position and semantic type) are still unspecified for certain predicates within requirement descriptions represented in NL (i.e. unfilled slots in the template created after requirement extraction). On the one hand, this step lays the foundations to determine the correct context-sensitive reading and to disambiguate the initial user-generated requirement descriptions.

On the other hand, the number of unfilled slots can be quantified (amount of free arguments) and we get insights in the probability of the domain-specific sense of predicate-argument-tuples representing requirements.

4.3 Resolving Underspecification

Every user-generated NL requirement description of a service is unique. By extracting its main components, it loses its individuality. With every further non-expert user, new requirements or at least variants of already known requirements[5] will extend the repository of formal expressions, components, and templates. Since our NL requirement extraction process (cf. Section 4.1) works iteratively, we generate and instantiate templates (so-called requirement scenarios) per description text. In this step of resolving underspecification, all resulting requirement scenarios are aggregated. Here, our goal is to automatically detect semantic similarities in requirement patterns for their assignment to domain-specific service classes such as "issue tracking system" or "event management". Furthermore, we perform similarity retrieval on this repository in order to provide suggestions for completion of requirements written in NL.

[5]Service requirement descriptions collected by user input or gathered from online App marketplaces.

Since the repository grows incrementally, it is not possible at the beginning of the extraction process to apply a common classifier for feature learning. Due to a lack of comparative scenarios statistical similarity measures are not suitable. Therefore, a seed list of classification rules (similarity criteria) is generated on the basis of instances and semantic-syntactic patterns identified during the requirement extraction process (cf. Section 4.1). Moreover, it has to be investigated in how far already existing (fuzzy) matching techniques can be applied for this.

4.4 Interactive Service Specification

Based on our findings, subsequently methods for completion and disambiguation of requirement specifications are developed through user interaction. In this step, incomplete requirement specifications are compared to similar scenarios, in order to automatically complete them (if necessary). Especially in the course of the pattern matching during the requirement extraction process, templates cannot be fully instantiated if there is missing information. This can lead to gaps in the structured requirement representations which have to be compensated before they are transformed into the formal service specification.

4.4.1 Similarity Retrieval for Auto-Completion

On the one hand, incomplete requirement specifications are matched with similar requirement scenarios (cf. Section 4.3) in order to complete them if necessary. Especially in case of the pattern matching (cf. Figure 2) the templates cannot be instanced completely in case of missing information of the end user. This may sometimes lead to unfilled slots in the structured representations, which have to be avoided in order to describe a configurable service.

4.4.2 Suggestion Generation for Disambiguation

The most probable readings based on the grammar analysis are suggested to the non-expert users. Then, they take on the role of the trainer for the purpose of validating identified ambiguities in the service description corpora. This way, it is ensured that no incorrect readings are learned. In case of requirements being so vague that too many grammatical patterns would be possible, paraphrases are generated based on domain-specific similarities in the requirement scenarios.

4.4.3 Non-expert User Feedback

It is assumable that non-experts in requirement engineering will not define the pre- and post conditions or signatures when formulating their requirements. In case of missing requirement scenarios, we ask the users how to complete and specify their initial service descriptions. However, when the users do not provide this information, we cannot transform the requirement represented in NL into the formal service specification. As soon as the users provide additional information, we have to restart the extraction, disambiguation and clarification process.

5 CONCLUSION

In the context of non-expert customers describing their service requirements, two main issues have to be tackled: (1) The incapability of non-experts to specify their requirements and (2) the challenges that arise when analyzing these requirements represented in NL such as ambiguity, vagueness and underspecification. This paper presented ideas how to face these challenges by means of ontology-based information extraction and similarity retrieval based on a requirement description written NL corpora gathered from App marketplaces. In particular, these strategies are also partially supported by user interaction. The overall goal of this approach is to support non-experts with suggestions how to complete or clarify their service descriptions while resolving ambiguity, vagueness and underspecification that often occur in requirements written in NL.

6 FUTURE WORK

A major direction of future work is the development of a gold standard of non-expert service descriptions in order to detect ambiguity, vagueness, and underspecification in requirements written in NL. Therefore, we first have to create a database of requirements represented in NL which we plan to aggregate from service descriptions on App marketplaces. Furthermore, it will be interesting to investigate in how far existing matching approaches and approaches for requirement reuse can be applied for the disambiguation and completion tasks. Moreover, with regard to disambiguation of requirement descriptions represented in NL we plan to explore how to refine our disambiguation strategies with regard to existing disambiguation approaches.

ACKNOWLEDGEMENTS

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre 901 “On-The-Fly Computing”.

REFERENCES

Berry, D. M., Kamsties, E., and Krieger, M. M. (2003). From contract drafting to software specification: Linguistic sources of ambiguity. https://cs.uwaterloo.ca/%7Edberry/handbook/ambiguityHandbook.pdf.

Bhat, M., Ye, C., and Jacobsen, H.-A. (2014). Orchestrating SOA Using Requirement Specifications and Domain Ontologies. In *Service-Oriented Computing*, pages 403–410. Springer.

Boyd, S., Zowghi, D., and Farroukh, A. (2005). Measuring the expressiveness of a constrained natural language: an empirical study. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 339–349. IEEE.

Choi, S.-P., Song, S.-K., Jung, H., Geierhos, M., and Myaeng, S. (2012). Scientic Literature Retrieval based on Terminological Paraphrases using Predicate Argument Tuple. *Advanced Science and Technology Letters*, 4:371–378. Information Science and Industrial Applications.

Espana, S., Condori-Fernandez, N., Gonzalez, A., and Pastor, Ó. (2009). Evaluating the completeness and granularity of functional requirements specifications: a controlled experiment. In *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International*, pages 161–170. IEEE.

Fabbrini, F., Fusani, M., Gnesi, S., and Lami, G. (2001). An automatic quality evaluation for natural language requirements. In *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 1, pages 4–5.

Ferrari, A., dell'Orletta, F., Spagnolo, G. O., and Gnesi, S. (2014). Measuring and Improving the Completeness of Natural Language Requirements. In *Requirements Engineering: Foundation for Software Quality*, pages 23–38. Springer.

Fuchs, N. E. and Schwitter, R. (1995). Specifying logic programs in controlled natural language. In *Proceedings of the Workshop on Computational Logic for Natural Language Processing*, pages 3–5. Arxiv.

Geierhos, M. (2010). *BiographIE – Klassifikation und Extraktion karrierespezifischer Informationen*, volume 5 of *Linguistic Resources for Natural Language Processing*. Lincom, Munich, Germany.

Génova, G., Fuentes, J. M., Llorens, J., Hurtado, O., and Moreno, V. (2013). A framework to measure and improve the quality of textual requirements. *Requirements Engineering*, 18(1):25–41.

Gleich, B., Creighton, O., and Kof, L. (2010). Ambiguity detection: Towards a tool explaining ambiguity sources. In *Requirements Engineering: Foundation for Software Quality*, pages 218–232. Springer.

Ilieva, M. and Ormandjieva, O. (2005). Automatic transition of natural language software requirements specification into formal presentation. In *Natural Language Processing and Information Systems*, pages 392–397. Springer.

Kaiya, H. and Saeki, M. (2005). Ontology based requirements analysis: lightweight semantic processing approach. In *Proceedings of the Fifth International Conference on Quality Software (QSIC 2005)*, pages 223–230. IEEE.

Kiyavitskaya, N., Zeni, N., Mich, L., and Berry, D. M. (2008). Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. *Requirements Engineering*, 13(3):207–239.

Loukanova, R. (2013). Algorithmic granularity with constraints. In *Brain and Health Informatics*, pages 399–408. Springer.

Menzel, I., Mueller, M., Gross, A., and Doerr, J. (2010). An experimental comparison regarding the completeness of functional requirements specifications. In *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE)*, pages 15–24. IEEE.

Meth, H., Brhel, M., and Maedche, A. (2013). The state of the art in automated requirements elicitation. *Information and Software Technology*, 55(10):1695–1709.

Murtaza, M., Shah, J. H., Azeem, A., Nisar, W., and Masood, M. (2013). Structured Language Requirement Elicitation Using Case Base Reasoning. *Research Journal of Applied Sciences, Engineering and Technology*, 6(23):4393–4398.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley, Harlow, England, 9th edition.

Yang, H., De Roeck, A., Gervasi, V., Willis, A., and Nuseibeh, B. (2010a). Extending nocuous ambiguity analysis for anaphora in natural language requirements. In *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE)*, pages 25–34. IEEE.

Yang, H., De Roeck, A., Gervasi, V., Willis, A., and Nuseibeh, B. (2011). Analysing anaphoric ambiguity in natural language requirements. *Requirements Engineering*, 16(3):163–189.

Yang, H., Willis, A., De Roeck, A., and Nuseibeh, B. (2010b). Automatic detection of nocuous coordination ambiguities in natural language requirements. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 53–62. ACM.