

# Domain Dependent Semantic Requirement Engineering

## Research-In-Progress

Wolf Fischer and Bernhard Bauer

Programming Distributed Systems Lab, University of Augsburg, Germany  
[wolf.fischer|bauer]@informatik.uni-augsburg.de

**Abstract.** Requirements Engineering is one of the most important phases in any product development life cycle since it is the basis for the complete software or product design and realization. Therefore in a worst case scenario any error within a requirement can result in a project loss. Computer support for requirement engineers is still premature as most RE applications can not cope with more sophisticated techniques like semantics, natural language processing etc. In this paper we will present the ongoing research work in semantic requirement engineering which will try to close the gap between computer understandable semantics, namely ontologies, and the natural language input of a human.

## 1 Introduction

Requirements engineering is known to be one of the most important phases in the development of a product. Correctly acquired requirements can lead to a solid and fluid creation of the to be developed artifacts whereas incorrect, contradictory or incomplete requirements can result in a complete project loss in a worst case scenario (e.g. 63% of all errors in the medical sector result from the requirement engineering phase [4]). In the past different solutions have been proposed to treat these problems. They can basically be classified in two categories: Informal and strongly formal ones. Informal systems are such which rely on natural language and somehow try to cope with the existing problems (the majority of requirements is written in natural language [11]). Their advantages are the overall simple usability, at least in the beginning. However, especially in larger projects the increasing complexity makes it difficult to cope manually with requirements documents written in natural language. Strong formal systems rely on the specification of requirements using logical formulas having the advantage that they can be analyzed and verified by computers. Their main drawback is that on the business level most of the people are not familiar using logical formulae. Moreover it is a complex and time-consuming undertaking. Some approaches have tried to combine the advantages of both worlds and created ways of capturing requirements in models (e.g.  $i^*$  [17] and UML [12]). Although these

techniques are easier to understand than logical formalisms there is still the problem of many humans adapting to new technologies. Therefore the system has to adapt to humans, i.e. the system has to be able to cope with text in natural language, extract relevant information from it and use it to support the user. Such a system could be used not only by professional requirement engineers but also by customers to identify their intentions and guide them to find previously gathered solutions automatically, as we will show in this paper.

We describe a highly integrated approach to requirements engineering, combining semantic technologies and NLP for requirements engineering. The rest of the paper is structured as follows: In section 2 the related work relevant to our approach is described. Next we present the overall approach to this topic in section 3. Section 4 exemplary shows how the presented concept works by showing it on a small example. Next an outlook on future work as well as a discussion on the possibilities and boundaries of this approach is outlined in section 5. Finally, section 6 concludes the paper.

## 2 Related Work

For treating requirements in a formal manner there have been different approaches presented in the past. In [17], Yu used  $i^*$  to model requirements and reason on them. This approach however does not cope with the textual description of requirements and how one can come from this informal representation to a more formal one. Many different approaches use the advantage of ontological technologies but did not close the gap between a textual description of the requirements and the semantic representation: Lee and Ghandi [9] developed an ontology acting as a common language for all stakeholders of a domain. It allows the modelling of different viewpoints, goals and the requirements itself. Dobson et al. ([5]) presented an ontology for describing non-functional requirements.

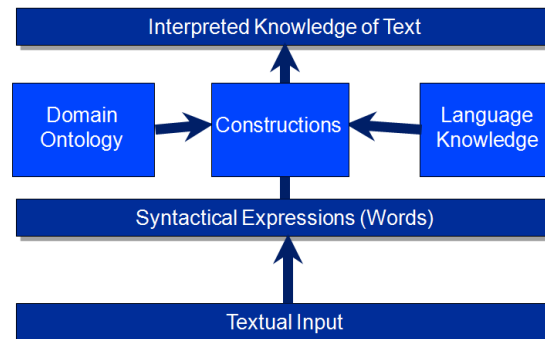
However some approaches tried to combine semantic technologies with NLP in the requirement engineering field. Kof ([7]) relied on ontology extraction from text without additional domain information in the beginning. His main goal is to give the user some hints on how to write more precisely. However, there is no consequent combination of semantics and syntax. In [13], the NLP analysis was done without semantic information at hand based on a clustering approach. Thus problems like resolving synonyms remain.

Lohmann et al. [10] developed a semantic wiki for requirement engineering, especially for a large number of participants. In this case the semantics are based on the connections between different requirements (each requirement has its own URI). A deeper semantic representation of requirements is possible, as every term within a text can be linked to another wiki page. However there is no automatic mechanism to annotate a requirement semantically.

The market offers many different tools for eliciting requirements. The most prominent one is probably IBM Rational Doors<sup>1</sup>. It allows an efficient, hierarchical management of textual requirements as well as creating dependencies

<sup>1</sup> <http://www-01.ibm.com/software/awdtools/doors/productline/>





**Fig. 1.** Big picture of the analysis process

life cycle of requirements, i.e. from early requirements (e.g. a customer request) over the resulting business requirements to the delivered product. The main concepts of the requirements ontology can be seen in figure 2. An **AbstractRequirement** is anything which can contain a requirement, a wish or a problem description. There are further specializations like a **Request** (intended for initial user requests) or a 'typical' **Requirement** (which is further endetailed by a **NonFunctionalRequirement** and a **FunctionalRequirement**). Each requirement is created by a specific **Stakeholder**. Exemplatory a Stakeholder can also be a **User** or an **ExternalStakeholder**. There are multiple other types of stakeholders possible, but are not described in detail here because of lack of space. Information about the competence of the user can be gathered by the **UserCompetence**. Finally, specific Stakeholders are responsible for a requirement which is indicated by the 'isResponsibleFor' association. To describe a requirement it contains a **RequirementDescription**. This one references different **RequirementResources**. A RequirementResource is a very generic concept which can represent anything domain specific. In the figure there are four specializations, i.e. the **Product** (which a requirement is about), an **Event** (e.g. to specify a certain point when a problem occurs), a **Document** (which is e.g. the source of the requirement) or a **System** (which could also act as a requirement source).

### 3.2 Linguistic Knowledge

This requirements ontology serves as a reference for the application and the ontology engineer: The application 'knows' the type of certain domain specific information (e.g. that the CEO of the company is a Stakeholder). The ontology engineer has a reference for which information the system might expect from a user and can model the domain ontology accordingly (an example is given in section 4).

In order to use the purely semantic information it has to be enriched with syntactic information. The basic concept behind it is described in [6]. The basic



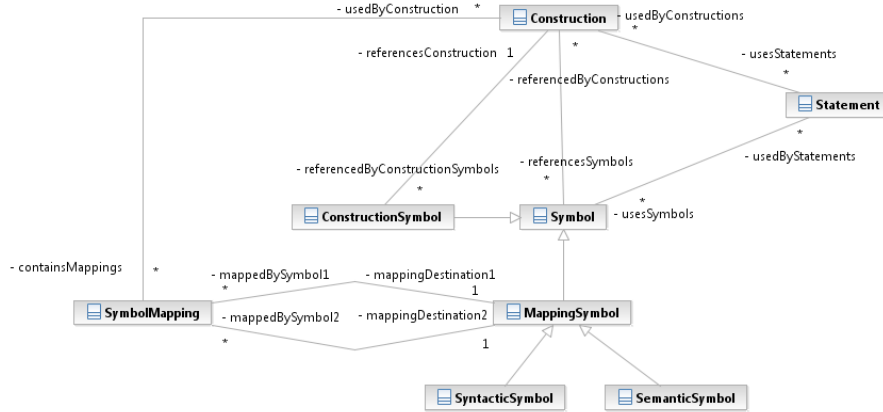


Fig. 3. Excerpt of the construction structure

a specific problem (i.e. the car rattles if it reaches a specific speed range). Further information is available in the ontology, i.e. that the cylinders are connected to the rattling. Concepts which are entangled with a blue rectangle, belong to the previously described requirements ontology.

To use these information in an NLP application the information is enriched with linguistic information as shown in figure 5. On the upper left side of the figure the semantic element 'Car' is displayed. This element is mapped on a possible string representation 'Vehicle'. In order to use this mapping it has to be referenced by a construction (in this case the **Car Construction**). To use a semantic element in a construction a symbol is introduced which is referenced by the construction. The symbol itself then references the corresponding semantic element. The cause for this structure is that the same syntactic or semantic information has to be used twice or more within a construction. A symbol can therefore be seen as a variable name and the element it references is the variable type. Mapping a semantic element to a syntactic representation is a simple task in contrast to describing more complex phrasal structures. The right side of the figure contains a construction which specifies a simple subject, predicate, object construction. It references the syntactic category noun twice ('Noun Symbol 1' and '2') as well as the verb category. To specify more concrete structures e.g. the order in which specific terms must have been written in the text can be done by using so called 'Statements'. A statement is a function which gets different symbols as its arguments. In this example the 'InOrder' statement checks if the given symbols appear in the correct order. Statements can also be used to check for certain semantic conditions (e.g. if a certain word has a specific semantic meaning which is useful in case of homonyms) or to create specific semantic constructions (which will result in the interpretation of a text): In figure 5 an additional statement in the Subj→Pred→Obj construction could define that the



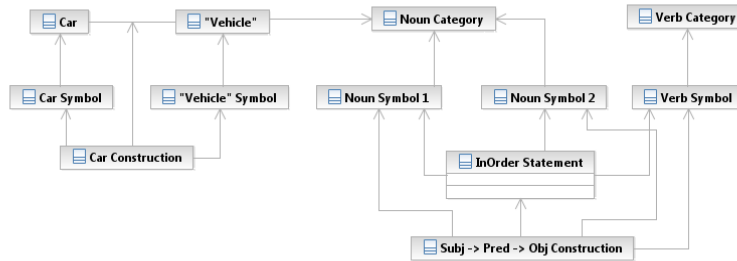


Fig. 5. Enrichment of the domain with linguistic information

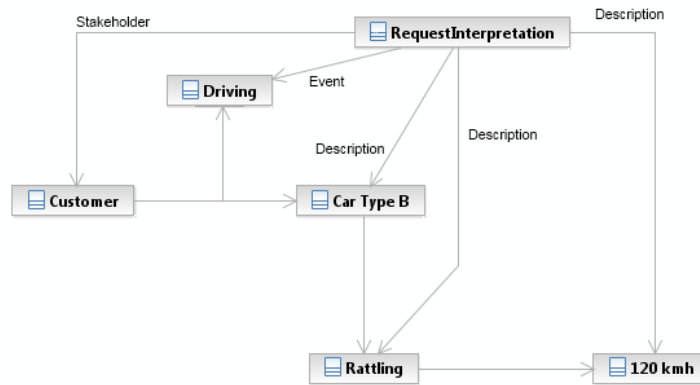


Fig. 6. Interpretation of the text resulting in a possible requirement





ically detecting overlaps and duplicates. Further erroneous as well as incomplete requirements could be identified automatically and therefore help the requirements engineer doing his / her work. However there are several limitations to this approach. As for every knowledge intensive system there must be an intensive amount of knowledge, especially linguistic information. At the moment there are many different treebanks available which represent a good foundation for this linguistic source of knowledge. We are currently developing an import mechanism for that task. It will be interesting to see how good the information from these sources can be adapted to our system and how much time this process will consume (i.e. how many changes a human has to make to this imported information for them to be usable). Further, this knowledge needs to be updated regularly (i.e. either existing information has to be changed or new one has to be added as domains are dynamic systems). This is a tedious and time consuming task and one of the biggest problems with these system types. A way to circumvent this problem are learning components, i.e. concepts and algorithms which help the system to adapt to new and unknown situations. This will be a part of our future work especially as the core of our concept proves a very promising approach to this task (i.e. its combination of syntax and semantics). Another limitation is metaphorical reasoning, i.e. the possibility to resolve metaphors and what they mean in the corresponding context. In a linguistically limited domain this might not be a big problem (as a clear linguistic description is required and therefore metaphors should not be used) but e.g. business / early requirements might contain an imprecise linguistic description (i.e. metaphors).

Next detecting references within text is very difficult for any NLP system. Our approach also faces this problem however due to the usage of semantics from the very beginning we hope to be better suited to this problem, as we can resolve dependencies not only on a syntactic but on a fact driven level as well.

A fifth boundary (one we cannot tackle in the near future) is pragmatics. Pragmatics can be described as semantics in context. An example would be a scenario, consisting of a room with an opened window, a person A standing at the window and another person B standing at the door. B utters 'It's cold in here', which from a purely semantic point of view is the statement that the air in the room is cold. From a pragmatic point of view it also is a request to person A to close the window. To identify and manage pragmatics a system would have to 'imagine' the situation which is even more difficult than just interpreting semantics (and computer science has problems to do even this). The last sections contained many theoretical aspects and examples that we are working on. Some of our ideas have not yet been implemented and therefore it will take some more time until we can deliver a prototype which is capable of delivering the results that we have described in this paper. What our prototype is currently capable of is analyzing simple sentences and creating an interpretations for it. In figure 7 a full and automatically created interpretation of the sentence 'The car is driven by the CEO' can be seen. The lower part mostly contains complex (preceded by a 'Con' prefix) or mapping constructions ('Mapping' prefix), whereas the upper part contains the different syntactic (prefix 'SynSym') as well as semantic infor-



6. W. Fischer and B. Bauer. Cognitive-linguistics-based request answer system. Springer, 2009.
7. L. Kof. Natural Language Processing for Requirements Engineering: Applicability to Large Requirements Documents. In *Proc. of the Workshops, 19th International Conference on Automated Software Engineering*. Citeseer, 2004.
8. R. Langacker. An introduction to cognitive grammar. *Cognitive Science: A Multidisciplinary Journal*, 10(1):1–40, 1986.
9. S. Lee and R. Gandhi. Ontology-based active requirements engineering framework. In *Proc. 12th Asia-Pacific Soft. Engg. Conf.(APSEC 05)*, IEEE CS Press, pages 481–490, 2005.
10. S. Lohmann, P. Heim, S. Auer, S. Dietzold, and T. Riechert. Semantifying Requirements Engineering—The SoftWiki Approach. In *Proceedings of the 4th International Conference on Semantic Technologies (I-SEMANTICS)*, pages 182–185, 2008.
11. M. Luisa, F. Mariangela, and N. Pierluigi. Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(1):40–56, 2004.
12. L. Nielsen. *Requirements Engineering: Anforderungsdefinition mit Hilfe von UML bei der Entwicklung einer ERP Software*. GRIN Verlag, 2008.
13. J. och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell. A linguistic engineering approach to large-scale requirements management. *Managing Natural Language Requirements in Large-Scale Software Development*, 22(1):145, 2005.
14. L. Steels. Fluid Construction Grammar Tutorial. Tutorial. 2004.
15. L. Steels and J. De Beule. A (very) brief introduction to fluid construction grammar. In *Proceedings of the Third Workshop on Scalable Natural Language Understanding*, pages 73–80. Association for Computational Linguistics.
16. L. Steels and J. De Beule. Unify and merge in fluid construction grammar. *Lecture Notes in Computer Science*, 4211:197, 2006.
17. E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, page 226. Citeseer, 1997.