

RESI - A Natural Language Specification Improver

Sven J. Körner

Institute for Programming and Data Structures
University of Karlsruhe
76128 Karlsruhe, Germany
Email: koerner@ipd.uka.de

Torben Brumm

Institute for Programming and Data Structures
University of Karlsruhe
76128 Karlsruhe, Germany
Email: brumm@ipd.uka.de

Abstract—Working with requirements means dealing with problems related to incomplete or faulty textual specifications. Specifications are created in various types and combinations such as models, drawings, and textual documents. The information gap between the specification and the implementation of any (software) system leads to delays, mistakes, and high costs due to problems in the production stages. Therefore, the earlier specifications are corrected and improved, the better. We provide a tool called RESI¹ to support analysts while working with textual specifications. RESI considers many of the directives to be followed when processing customer specifications [1], [2]. It offers a dialog-system which makes suggestions and inquires the user when parts of the specification are ambiguous, faulty or inaccurate. RESI utilizes various ontologies to discover such problems and to deliver common sense solutions.

I. INTRODUCTION

Requirements engineering (RE) has been struggling with the same type of problems since its discovery as a discipline in software engineering. RE deals with customer needs which have to be casted into requirements and specifications.

During the software engineering process, the customer's vague formulations need to be analyzed and documented in a way that the software developers can implement the desired functionality of the product. Due to the fact that user requirements are usually expressed in natural language, there is a problem with the expressiveness, the completeness, and the accuracy of user statements. Until today, human analysts have to evade these problems with a sleek psychological approach towards the customer and a skilled mind to ask the right questions.

Humans use common sense to discover these questions. To support analysts in this process, a tool would need to also use "common sense". Ontologies offer a possibility to provide this kind of knowledge to machines.

Our research project AUTOMODEL is part of the SaleMX [3] stream to automate the creation of requirement models from requirement specifications. The first component of AUTOMODEL was the *Automatic UML Improver* [4] as can be seen in Figure 1. AUTOMODEL utilizes ontologies to improve the requirements engineering process and aims to complete the existing tools for requirements management and processing. Today's requirements engineering tools offer modeling and management functions [5], but are hardly able to cope with the psychological and linguistic problems. RESI

¹Requirements Engineering Specification Improver

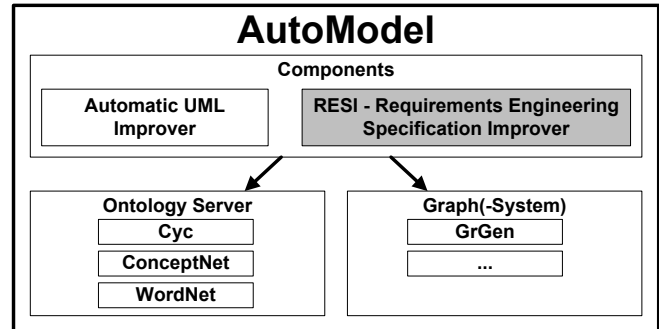


Fig. 1. The AutoModel Research Project

is part of the AUTOMODEL approach and deals with the linguistic processing of textual information.

This paper is divided into 6 sections. The following Section II discusses related work, especially ontology usage and psychological aspects. Section III explains the challenges for our tool RESI and the solutions it provides. This includes an architectural overview as well as a description of the tool's functionality. Section IV evaluates a case study that has been conducted with two different specifications. The paper closes with an outlook in Section V and the conclusion in Section VI.

II. RELATED WORK

Previous work has shown that formal representations of natural language are a way to address the problems analysts face when eliciting requirements with the customer [6], [7], [8]. Formal representations make natural language machine processable and allow the modeling of meaning next to a phrase's syntax and semantics. They provide a solid base for approaches like First Order Predicate Calculus (FOPC) and the therein involved quantifiers, variables, modus ponens, and so forth. Problems with formal representations are user-acceptance and usability as they grow large and confusing when used in conjunction with complex phrases, larger paragraphs, or extensive texts.

On the one hand, it is vital that the requirements specifications are machine processable to provide machine support for requirement engineers. On the other hand, it is important that the customer fully understands the specification so that he can take part in the requirements verification. This is only possible if we provide easy-to-understand models or improved

TABLE I
THEMATIC ROLES AND THEIR MEANING

Thematic Role	Explanation
AG	The acting person or thing executing the action.
ACT	The action, executed by person or thing.
PAT	Person or thing affected by the action or on which the action is being performed.
HAB	Possession or belonging; person or thing being received or passed on by person or thing.
POSS	The (current) owner of an element. The “possessor”.

textual specifications that are legible and do not need special training to understand. Additionally, specifications signed by the customer and the contractor are expected to be final and binding in a legal point of view. Depending on the interpretation of certain parts of the specification, this might involve serious legal or liability problems.

Gelhausen and Tichy [9] show that thematic roles [10], [11] can be used to add semantics to natural language textual specifications. A short list of thematic roles can be found in Table I. Textual annotations are used to convert a textual specification into a graph. Graphs are a machine processable format of language which can be automatically transformed into UML models [12]. But when creating more sophisticated models, many decisions need to be made which cannot be taken by a machine. The results stem from decision making processes which humans use in conjunction with their common sense knowledge [13]. Our research combines the formal approach and the human approach most analysts (have to) use today.

Thus, common sense is necessary when working with natural language. We support the opinion that a deeper understanding of human language processing – and therefore psychology – can help building better machine models to process language [11]. If we want the computer to be able to do that, we need to utilize large knowledge bases: ontologies.

A. Using Ontologies to Decipher Meaning

As we were able to show in 2008 [4], many of the decisions that need to be taken during automatic modeling can be made by a system which gathers information from ontologies. For example, the thematic role ACT² can either be represented as a method, a role, or a state-transition in a UML diagram. Prior to including ontology knowledge, these decisions had to be made from humans during the semi-automatic UML modeling process. After utilizing common sense from ontologies, the system alone was capable to decide which type of UML model representation to choose.

Improving the UML model representation of a system after it has been annotated and converted into a graph [9] can be seen as post-processing of a specification. As we kept improving the combination of automatic model creation and applying ontology based decisions, we realized that many

²Mostly a verb

other specification problems could be ruled out in even earlier stages: a profound pre-processing is able to remedy problems the requirements analyst will be faced with later. Therefore, this work focuses on improving textual specifications even before the automatic modeling takes place.

B. Psychological and Human Problems

When formulating specifications in natural language, it is difficult to ensure that the reader of the specification knows exactly what the customer wanted to express. The specification should match the customer’s idea completely with only one distinct interpretation.

Berry et al. wrote a handbook [1] to help writers of specifications (contracts) to avoid ambiguities. They cover a huge area of possible ambiguities (e.g. usage of *all*, *each* and *every*; quantifiers as *a*, *all*, *any* etc.), explain the according problems by example and show how to avoid them. They also mention other problems like ambiguous words or phrases that may not seem ambiguous to the writer but have a different meaning for the reader.

Rupp and the SOPHIST group dedicate a complete chapter [2] to finding and avoiding “linguistic defects” in natural language specifications. These defects are produced unintentionally when formulating sentences in natural language and lead to inferior specifications. According to Bandler [14], Rupp divided them into three categories: deletion, generalization and distortion. We explain these categories, describe their manifestations, and give example sentences.

1) *Deletion*: Deletion is the process of leaving out details from experiences to make them easier to process, e.g. filtering voices in a room full of people to understand the dialog partner. When unintentionally leaving out necessary details in a specification, the specification is incomplete. Some of the mentioned manifestations of deletion are:

- Incompletely specified process words³ need further specification. Passive sentences for example often lack of a needed actor (AG) and some process words (ACT) do not have all arguments⁴ specified. An example sentence is:

User name and password are entered.

The corresponding questions would be: Who enters the data? Where does he or she enter it?

- Modal verbs of possibility need further specification *why* these actions can (or cannot) be taken.

A user cannot access the personal page of another user.

Why not? What stops him from doing that?

- Modal verbs of necessity need further specification what to do in the exceptional case when the desired behavior cannot be met.

The user credentials must be checked by consulting the database.

³Process words are mostly verbs or predicates [15].

⁴Arguments are phrases that appear in a syntactic relationship with the process word (subject, object, etc.)

What if the database is not available? Does the process wait or cancel?

2) *Generalization*: Generalization is the process of using an experience as an embodiment for all similar experiences, e.g. never touching a hot plate again after touching one specific hot plate. If this generalization is too strong, it might lead to incomplete specifications. Mentioned examples are:

- Universal quantifiers that are not really meant for every single member of the set should have exceptional cases specified.

Each user can access only his personal data.

What about administrator users?

- Incompletely specified conditions (*If...*) also need an else-case, when the mentioned conditions do not meet.

In case of a successful login the user is redirected to his personal page.

What if the login was not successful?

- Nouns without reference index (nouns for which it is not clear what part of a set they are referencing to) need to be used with the correct determiner or quantifier.

The personal data can be seen on the personal page.

Which personal data? Everything? Only the important data? And which personal page? Can Paul's personal data be seen on Peter's personal page?

3) *Distortion*: Distortion is the process of rearranging specific details, e.g. talking about a wrong decision (a single event) although it was a continuing process of deciding. In specifications this leads to written requirements that do not match the real requirements. Rupp mentions two examples:

- Nominalizations may hide a complex process behind a simple event.

After the login the user can access his personal page.

And how exactly is this login done?

- Stretched verbs⁵ are often less precise than the according normal verb.

Mary has a drink.

The normal verb *drink* would automatically raise the question what she drinks.

A skilled analyst who can write good specifications to avoid or remove the problems listed above is necessary. Usually, he has to identify the problems without tool support. After identifying the problems, they have to be alleviated in dialog with the customer to improve the specification. In the following Section III we present a tool that can help the analyst to identify these problems in specifications.

⁵A stretched verb is a complex predicate composed of a light verb and an eventive noun.

III. REQUIREMENTS ENGINEERING SPECIFICATION IMPROVER (RESI)

RESI is based on prior conceptual work [16]. A human analyst uses common sense to find problems. RESI uses various ontologies (WordNet [17], ResearchCyc [18], ConceptNet [19] and YAGO [20]) to access "common sense". These ontologies provide RESI with knowledge to identify problems in specifications and to suggest the corresponding solutions.

A. Identifying Problems

RESI can help an analyst to avoid several of the problems mentioned in II-B:

1) *Avoid Ambiguous Words*: RESI offers a list of possible meanings for each word in the specification. The analyst can see if a word has a meaning which he did not anticipate and which differs from the intended meaning. If an ambiguous word offers other possible meanings for the sentence, the analyst might want to change the word or formulation.

2) *Avoid Nominalization*: RESI finds possible nominalizations in the specification and suggests verbs to use instead.

3) *Avoid Incompletely Specified Process Words*: RESI returns a list of arguments for each process word with both syntactic and semantic role. It suggests which element of the sentence to use for each argument or states that there is no matching element in the sentence. This way it is possible to see if all process words are completely specified or if sentences require more information.

4) *Avoid Similar Meanings, Cluttering*: RESI searches for words with similar meanings. It is possible to replace one of two similar words with the other one to avoid confusion when using different words for the same object.

5) *Avoid Nouns without Reference Index and Wrongly Used Universal Quantifiers*: RESI checks each occurrence of a quantifier or determiner like *all*, *each*, *a*, *the* etc. The analyst can verify the quantifier's meaning or explicitly define the intended meaning.

RESI addresses the top three of the problems that should be eliminated with high priority according to Rupp [2] plus a few more. High priority problems are:

- 1) Nominalization
- 2) Incompletely specified process words
- 3) Nouns without reference index
- 4) Incompletely specified conditions
- 5) Modal verbs of necessity

B. Functionality

RESI works in four steps as shown in Figure 2.

1) *Specification Import*: First we import a specification (represented as a graph) that we want to improve. By now RESI only supports GrGen [21] graphs that are created using the SALE model [9]. It is possible to implement interfaces to other representation forms of the specification (see Section III-C2).

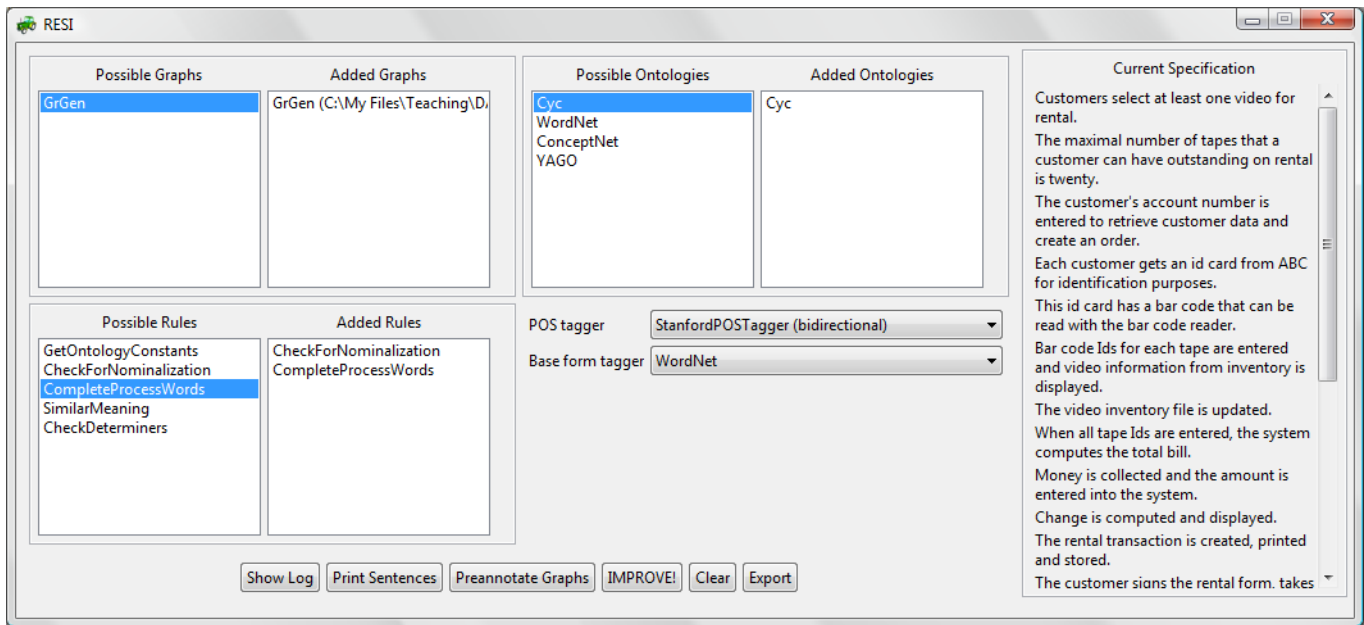


Fig. 3. Configuration of RESI

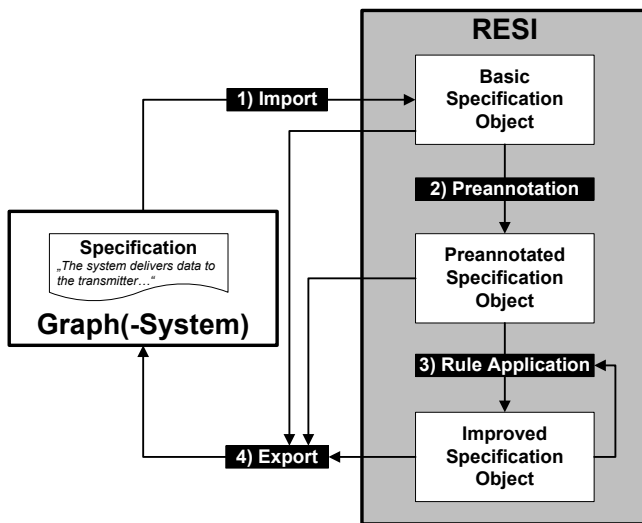


Fig. 2. The RESI Workflow

2) *Preannotation*: For proper function RESI needs further information about the words of the specification. Each word is tagged with the according *part of speech* (POS) and – if it is a noun or verb – with its base form. Taggers execute this job fully automatically. Afterwards, the RESI user can adjust the tags manually if desired.

3) *Rule Application*: The specification is improved by applying so called *rules*. Each rule represents one of the solutions mentioned in Section III-A. The rules to apply and the ontologies to use for applying the rules can be selected. A screen shot can be seen in Figure 3.

It is not mandatory to apply all of the rules or to use all ontologies RESI supports. During the application of a rule

the RESI user can set marks to comment the specification. This simplifies the rediscovery of problematic sentences or words in the specification which need to be adjusted in consent with the customer. For example: if a problematic sentence is written in passive voice and therefore lacks the arguments of the process word, this leads to the output depicted in Figure 4. RESI discovers *return* as a process word in the sentence. In this case, *ReturningSomething2* is the best match from the ontology for this specific sentence and is therefore chosen by the RESI user. As can be seen, *return* needs to have an *objectGiven*, a *giver*, and a *givee* (the exact definition from the ontology is shown in the tool tip), but only the *objectGiven* can be found in the examined sentence. The subject and the oblique-object need to be further specified. In this case, the RESI user would mark the corresponding sentence to clarify with the customer who the *giver* and the *givee* are. Without this information, the specification is incomplete.

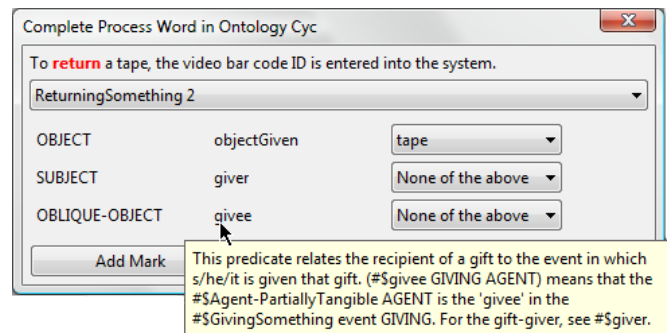


Fig. 4. User Dialog for Process Word Completion

Another example would be the use of determiners and quantifiers which are automatically discovered by RESI as

shown in Figure 5. RESI recognized the quantifier *twenty* and suggests that it should be specified with the numeral *20* and the property *exactly*. The user could make corrections to the quantifier detailing if RESI makes a wrong suggestion. If the correct quantifier is unclear, the user sets a mark and comments the corresponding word of the sentence. This way, its meaning can be further specified in coordination with the customer later.

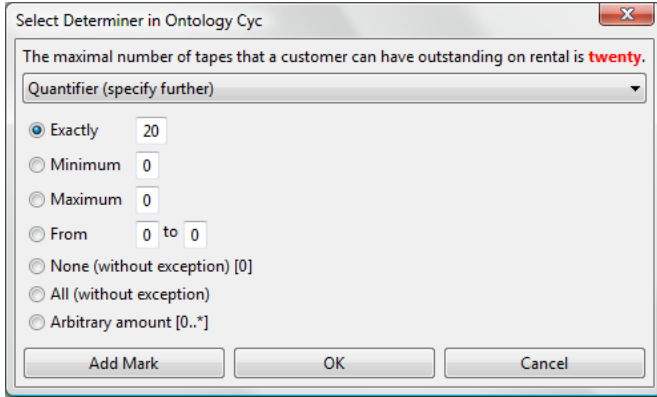


Fig. 5. User Dialog for Determiners and Quantifiers

Rules can be applied repeatedly and as often as needed. The RESI user is allowed to skip certain parts of the specification and reapply the rules later. This is true for using different rules as well as the same ones to edit prior work.

4) *Specification Export*: When the specification is improved it can be exported into the format of the original specification. All additional information discovered during the use of RESI is also stored in the export. It is possible to re-import the exports for further processing. The export can be performed after import (although not reasonable as there is no new information), after preannotation, after manual adjustments, or after applying any number of rules.

C. Architecture of RESI and Interfaces to Adjoining Systems

The internal architecture of RESI consists of four parts: rule objects that provide functions, graph objects that represent the specification, ontology objects that provide the “common sense” and the central application that controls all components. The UML layout of RESI is shown in Figure 6.

1) *Rules*: As mentioned in Section III-B3, a rule represents a function RESI provides. Each rule reads information from a graph and processes it. The rule queries an ontology in case it needs further information (the step a human analyst would use common sense for). If the rule identifies a possible problem, it prompts the RESI user for interaction and provides suggestions. Any information gathered from these steps (including comments entered by the user) are written back to the graph.

Example for a rule: The rule to avoid cluttering (described in Section III-A4) gets all nouns from the graph and checks which of the nouns have similar meanings using the ontology. If a pair of synonyms is found, the user is asked if he wants to replace noun *A* with noun *B*. If he accepts, noun *A* is replaced with noun *B* in the graph.

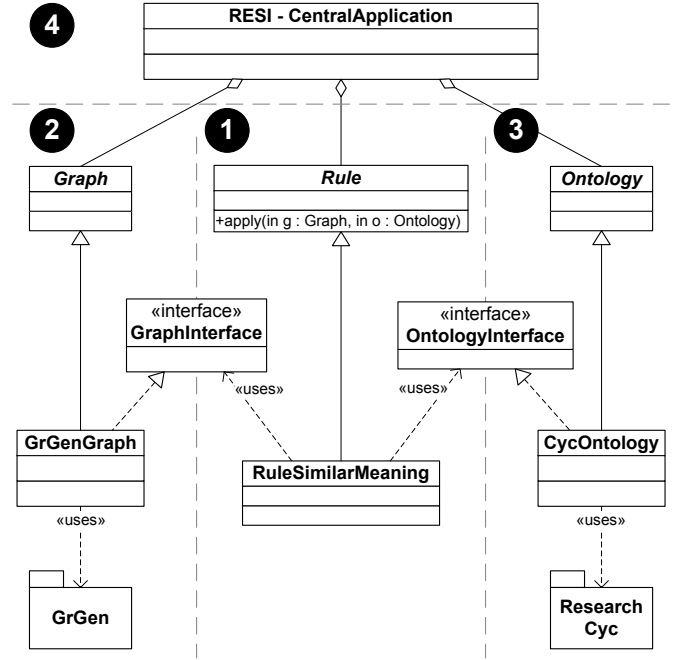


Fig. 6. UML Layout of RESI (Detailed description see III-C)

Each rule defines interfaces for graph and ontology classes which these must implement in order to support the rule. The UML diagram in Figure 6 shows the rule *RuleSimilarMeaning* with its interfaces. The functionality of RESI can be extended by implementing more rules for other problems.

2) *Graphs*: A graph object provides possibilities to read from and write to the specification. The graph class supports a rule, if it implements the graph interface of the rule.

For now we have only implemented a graph class that imports GrGen graphs from a file (depicted as *GrGenGraph* in Figure 6). This graph class supports all rules. RESI can be used with any representation of a natural language specification by implementing a new graph class (with all interfaces from needed rules).

3) *Ontologies*: The ontology class communicates with an external ontology by converting queries from rules to queries for the external ontology. If the ontology class implements the ontology interface of a rule, it supports this rule.

The external ontology can be run on the same machine or on a server if both the ontology class and the external ontology support communication via TCP. ResearchCyc runs as server application; we implemented our own server applications for the other ontologies. This way we do not have to run the external ontologies (which use a lot of resources) locally.

We support ResearchCyc, WordNet, ConceptNet and YAGO, but the use of ConceptNet and YAGO is very limited due to their ontology design. Latter of which only support the rule to find synonyms. WordNet finds synonyms and possibly ambiguous words. ResearchCyc supports all of our rules (shown in Figure 6 as *CycOntology*). Other external ontologies can be added to RESI by implementing a corresponding

ontology class. This class must implement the interfaces for the rules the ontology should support.

4) *Central Application*: The central application controls which of the components mentioned above interact (see screen shot in Figure 3). It manages *which* rules shall be applied to *which* graphs with the support of *which* ontologies.

These decisions are made by the RESI user. E.g., the interaction could look like Figure 7 which shows the sequence of the example mentioned in Section III-C1.

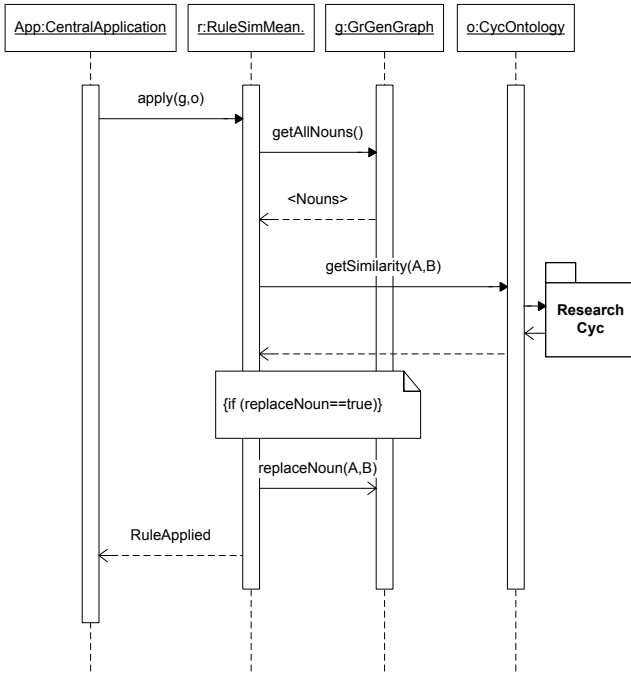


Fig. 7. Sequence Diagram of RESI Class Interaction

The central application also offers the possibility of pre-annotation (see Section III-B2). The taggers for POS and base forms are defined via an ontology interface, similar to the interfaces a rule must provide. Currently we use the *Stanford Log-linear Part-Of-Speech Tagger* [22] as POS tagger and WordNet as base form tagger.

IV. CASE STUDY

We tested our approach in a case study with two specifications: one is the example of a video rental store as used in [13], the other one is a small text from Berry’s ambiguity handbook [1] (denoted as *Spec1* & *Spec2* respectively in Table II). We used ResearchCyc as ontology for the rules.

A. Key Points of Interest

The key points of interest (KPI) as listed in Table II are divided according to Section III-A. The amount of ambiguous words used in the text as well as their possible additional meanings are given in the table section *Ambiguity Statistics*. For some unambiguous words, RESI is able to specify an exacter meaning. If a word appears multiple times (e.g. *customer*)

in the specification, it is detected every time but accounted for only once in the statistics.

The number of nominalizations is listed underneath followed by the amount of incomplete process words. We also state the total amount of arguments that were missing for the process words. If words with similar meaning are used, RESI resolves cluttering as many times as shown in the row below. All quantifiers and all determiners are correctly detected and the according meaning is displayed to be verified by the RESI user.

B. The Example Specifications

The video rental specification (*Spec1*) text is comprised of 17 sentences with a total amount of 222 words. Parts of the text can be found in the screen shot in Figure 3. As can be seen in the table, this short text already includes 44 possible ambiguities. If we take a closer look at the possible meanings of each word in the specification, we even end up with 198 meanings in addition to the intended ones.

The example taken from [1] (*Spec2*) exists of 99 words and 6 sentences. Each sentence describes the individual use of a pressure system monitor which has automatic functions, e.g. to initiate safety injections, that can be overridden by manual interference. It is a simplified version of the ESFAS (Engineered Safety Feature Actuation System) requirements that were originally introduced by Courtois and Parnas [23]. Kamsties et al. also used this text in their research concerning inspections for requirements documents [24]. They discuss in detail one of the problems that can be found using their inspection technique. This problem is also automatically detected by RESI.

C. Statistical Subtleties

Working with RESI during the evaluation led to astonishing discoveries. When manually working through the sentences, we realized that the “curse of knowledge” [25] occurs more often than requirement engineers anticipated. This phenomenon describes the psychological aspect of implicit assumptions based on context and domain knowledge that inevitably leads to incomplete specifications. This was detected when requirement engineers went through the specifications using RESI. When RESI pointed out various ambiguities in the sentences, many requirement specialists only then realized that the sentence actually had multiple meanings. Therefore they discovered some problems with RESI which they were not able to detect manually. Without RESI, they used their domain knowledge and context information to model the corresponding sentences. The specialists were convinced that they would not have realized the specification flaws without RESI in such an early stage.

Chantree et al. [26] describe this occurrence of unacknowledged nocuous ambiguities in their paper. They show that unacknowledged ambiguity has consequences for the implementation. The stakeholders might not realize that the corresponding sentence embodies ambiguity depending on each stakeholder’s point of view. The problems in the specification

TABLE II
CASE STUDY: RESI SPECIFICATION IMPROVEMENTS

Statistics	Spec1	Spec2	Spec3
# Sentences	17	6	1
# Words	222	99	7
# Characters (no spaces)	1036	486	38
# Nouns	76	27	2
# Verbs	36	14	1
KPI	Spec1	Spec2	Spec3
<i>Ambiguity Statistics:</i>			
# Ambiguous Words	44	15	2
# Additional Meanings	198	42	5
# Exacter Meaning	12	6	1
# Nominalizations	4	4	1
# Incomplete Process Words (Verbs)	14	3	1
# Missing Arguments	18	5	2
# Detected Cluttering	11	2	0
# Quantifiers	8	0	1
# Definite Articles	24	12	0
# Indefinite Articles	6	6	0

will never be resolved with the result of an incorrect implementation.

It is apparent that even straightforward and easy to read specifications contain a large number of possible errors which the analyst has to scrutinize and fix. If you consider the following sentence (*Spec3*) for example:

Every pallet must be returned after transport.

This simple sentence includes several potential error sources which must be deleted before the specification is ratified. RESI found the following problems which are also listed in Table II:

- *return* and *transport* have multiple meanings.
- Though *pallet* is not ambiguous for RESI, RESI adds additional semantic information to avoid an ambiguity that could not be detected.
- *transport* is a nominalization and should be specified in more detail.
- The process word *return* lacks of two arguments: the person who returns the pallets and the place where the pallets are returned to.
- The quantifier *every* implies that there is no exception. The RESI user should check this statement (e.g. specify further that this only concerns the pallets that were used for this transport and that were not broken).

Manually detecting all flaws in a timely manner is not possible. With RESI, the user is able to jump through the specification, make changes and mark special parts for clarification with the customer. This is much faster, more complete, and more profound as any human could improve and deliver an (initial) specification.

V. OUTLOOK

RESI is not yet fully complete. There are various other aspects we would like to be able to detect when working

through textual specifications. We plan to further evaluate the tool in additional studies with more substantial cases. This includes the examination of RESI's impact on software quality.

So far, our automatic model creation [4], [9], [12] via transformation from a graph-system to UML is rudimentary though it does include all four UML layers provided by the OMG [27]. We plan to expand the features of the UML converter to include all the information RESI adds to the graph.

Additionally, we are working on a textual exporter of the RESI graphs. This way we ensure that changes made by RESI and the RESI user can be easily read by the customer. The exporter needs to include these changes and comments and is planned to track the changes that were made since the original specification.

VI. CONCLUSION

Requirement engineers spend a lot of time improving requirement specifications. Trying to detect defects before they emerge in later production stages is vital. We provide an automated approach to improve textual specification which applies ontologies to provide "common sense" for machines. This way, we ensure that the quality of the requirements does not rely only on the behavioral codex and the skill of the analyst, but that every analyst can use the software to provide valuable information about possible flaws and errors in the specification. Still, many decisions cannot be made directly and need to be discussed with the customer. RESI offers a way to mark certain words or sentences which can then be verified in dialog with the customer.

Best to our knowledge, using ontologies as "common sense engine" to interact with the user during RE is a new approach, which offers many possibilities for every RE process.

ACKNOWLEDGMENT

Research on the AUTOMODEL project is funded by ec4u expert consulting ag, Germany in cooperation with the University of Karlsruhe, Germany.

REFERENCES

- [1] D. M. Berry, E. Kamsties, and M. M. Krieger, "From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity - A Handbook," November 2003. [Online]. Available: <http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf>
- [2] C. Rupp and die SOPHISTen, *Requirements-Engineering und Management*, 4th ed. Carl Hanser Verlag, 2006.
- [3] T. Gelhausen, B. Derre, M. Landhäußer, T. Brumm, and S. J. Körner, "SaleMX project website," 2009. [Online]. Available: <http://svn.ipd.uni-karlsruhe.de/trac/mx/wiki>
- [4] S. J. Körner and T. Gelhausen, "Improving Automatic Model Creation using Ontologies," in *Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering*, Knowledge Systems Institute, Ed., Jul. 2008, pp. 691–696.
- [5] Volere, "List of requirement engineering tools," 2009. [Online]. Available: <http://www.volere.co.uk/tools.htm>
- [6] W. M. Adam Pease, "An English to Logic Translator for Ontology-based Knowledge Representation Languages," *IEEE 0-7803-7902-0/03*, pp. 777–783, 2003.
- [7] S. Konrad and B. H. Cheng, "Facilitating the Construction of Specification Pattern-based Properties," *Requirements Engineering, IEEE International Conference on*, pp. 329–338, 2005.

- [8] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw, "Automated consistency checking of requirements specifications," *ACM Trans. Softw. Eng. Methodol.*, vol. 5, no. 3, pp. 231–261, 1996.
- [9] T. Gelhausen and W. F. Tichy, "Thematic Role Based Generation of UML Models from Real World Requirements," in *Proc. International Conference on Semantic Computing ICSC 2007*, 2007, pp. 282–289.
- [10] T. Gelhausen, B. Derre, M. Landhäußer, T. Brumm, and S. J. Körner, "SaleMX project website - The thematic roles of SALE," 2009. [Online]. Available: <http://svn.ipd.uni-karlsruhe.de/trac/mx/wiki/MX/SALE/ThematicRoles>
- [11] D. Jurafsky and J. H. Martin, *Speech and language processing*. Prentice Hall, Prentice-Hall International, 2000.
- [12] T. Gelhausen, B. Derre, and R. Geiss, "Customizing GrGen.NET for Model Transformation," in *GRaMoT '08: Proceedings of the 3rd International Workshop on Graph and Model Transformation*. Leipzig, Germany: ACM, May 2008, pp. 17–24. [Online]. Available: <http://www.ipd.uka.de/Tichy/uploads/publikationen/180/gramot2-gelhausen.pdf>
- [13] N. Kiyavitskaya, N. Zeni, L. Mich, and D. M. Berry, "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications," *Requir. Eng.*, vol. 13, no. 3, pp. 207–239, 2008.
- [14] R. Bandler, *Metasprache und Psychotherapie: Die Struktur der Magie I*. Junfermann, 1994.
- [15] C. Rupp, "Requirements and Psychology," *IEEE, May/June 2002*, vol. IEEE SOFTWARE, 2002.
- [16] S. J. Körner and T. Brumm, "Improving Natural Language Specifications with Ontologies," in *Proceedings of the Twenty First International Conference on Software Engineering & Knowledge Engineering*, Knowledge Systems Institute, Ed., Jul 2009.
- [17] G. A. Miller, C. Fellbaum, R. Teng, P. Wakefield, H. Langone, and B. R. Haskell, "WordNet." [Online]. Available: <http://wordnet.princeton.edu/>
- [18] Cycorp Inc., "ResearchCyc." [Online]. Available: <http://research.cyc.com/>
- [19] C. Havasi, R. Speer, and J. Alonso, "ConceptNet 3: a Flexible, Multilingual Semantic Network for Common Sense Knowledge," in *Recent Advances in Natural Language Processing*, Borovets, Bulgaria, September 2007. [Online]. Available: <http://web.media.mit.edu/~jalonso/cnet3.pdf>
- [20] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A Core of Semantic Knowledge," 2007. [Online]. Available: <http://suchanek.name/work/publications/www2007.pdf>
- [21] R. Geiß, G. V. Batz, D. Grund, S. Hack, and A. M. Szalkowski, "GrGen: A Fast SPO-Based Graph Rewriting Tool," in *Graph Transformations - ICGT 2006*, ser. Lecture Notes in Computer Science, A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, Eds. Springer, 2006, pp. 383 – 397, natal, Brasil. [Online]. Available: http://www.info.uni-karlsruhe.de/papers/grgen_icgt2006.pdf
- [22] The Stanford Natural Language Processing Group, "Stanford Log-linear Part-Of-Speech Tagger." [Online]. Available: <http://nlp.stanford.edu/software/tagger.shtml>
- [23] P.-J. Courtois and D. L. Parnas, "Documentation for safety critical software," in *ICSE '93: Proceedings of the 15th international conference on Software Engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1993, pp. 315–323.
- [24] E. Kamsties, D. M. Berry, and B. Paech, "Detecting Ambiguities in Requirements Documents Using Inspections," in *Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01)*, 2001, pp. 68–80.
- [25] C. H. . D. Heath, *Made to Stick: Why Some Ideas Survive and Others Die*, 1st ed. Random House, 2007.
- [26] F. Chantree, B. Nuseibeh, A. de Roeck, and A. Willis, "Identifying Nocuuous Ambiguities in Natural Language Requirements," in *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 56–65.
- [27] Object Management Group, "Unified Modeling Language: Superstructure Version 2.2," PDF, February 2009. [Online]. Available: <http://doc.omg.org/formal/2009-02-02.pdf>