International Journal of Semantic Computing
Vol. 3, No. 4 (2009) 445–470

DOI: 10.1142/S1793351X09000872

World Scientific
www.worldscientific.com

NATURAL LANGUAGE SPECIFICATION IMPROVEMENT WITH ONTOLOGIES

SVEN J. KÖRNER* and TORBEN BRUMM†

Institute for Programming and Data Structures (IPD)
University of Karlsruhe, Am Fasanengarten 5
Karlsruhe, 76131, Germany
**koerner@ipd.uka.de*
†*brumm@ipd.uka.de*
www.ipd.uka.de

Requirements engineering can solve and cause many problems in a software development cycle. The difficulties in understanding and eliciting the desired functionality from the customer and then delivering the correct implementation of a software system lead to delays, mistakes, and high costs. Working with requirements means handling incomplete or faulty textual specifications. It is vital for a project to fully understand the purpose and the functionality of a software. The earlier specifications are corrected and improved, the better. We created a tool called RESI[a] to support requirement analysts working with textual specifications. RESI checks for linguistic defects [1, 2] in specifications and offers a dialog-system which makes suggestions to improve the text. It points out to the user which parts of the specification are ambiguous, faulty or inaccurate and therefore need to be changed. For this task, RESI needs additional semantic information which is inherent to natural language specifications. It receives this information by utilizing ontologies.

Keywords: Requirements engineering; ontologies; linguistics.

1. Introduction

Requirements engineering (RE) deals with customer needs which have to be casted into requirements and specifications. It has faced the same challenges ever since.

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. User requirements are usually expressed in natural language which leaves a problem with the expressiveness, the completeness, and the accuracy of the statements. Since many of these tasks are manual, a skilled analyst is necessary. He needs to write good specifications to avoid and remove these problems. Today's requirements engineering tools [3] offer modeling and management functions, but are hardly able to cope with the psychological and linguistic problems. Usually, the analyst has to identify the problems without tool

[a]Requirements engineering specification improver.

support which then have to be alleviated in dialog with the customer. In Sec. 4 we present a tool that can help the analyst identify these problems in specifications.

Besides detecting grammatical errors in natural language, humans use common sense to discover semantic problems that arise with faulty specifications. When we create sophisticated models and specifications, many decisions need to be made which cannot be taken by a machine. We need the human decision making process in conjunction with human common sense knowledge [4]. To support analysts in this process, a software tool would also need to use "common sense". Being able to use inherent semantics from textual specifications by using ontologies makes it possible to automate certain parts of the analyst's tasks as described in [2, 5].

We support the opinion that a deeper understanding of human language processing — and therefore psychology — can help build better machine models to process language [6]. If we want the computer to be able to do that, we need to utilize large knowledge bases. Ontologies are a powerful tool that can render this kind of knowledge usable by machines.

In 2008 we showed [7] that decisions which need to be made during the automatic modeling (of textual specifications into UML) can be performed by a system which gathers information from ontologies. Prior to including ontologies, these decisions had to be made from a human user. After utilizing "common sense" from ontologies, the system alone was capable of making the correct decisions. As we kept improving the combination of automatic model creation and applying ontology based decisions, we realized that many requirements engineering problems can be solved in early stages of the specification validation and creation. RESI supports the experienced analyst with profound pre-processing, it does not replace him. Neither does it replace the relationship of the analyst and the stakeholders, nor can it make all the decisions for the customer. RESI is developed to make suggestions to the user whenever possible, but its performance largely depends on the quality of the ontology used and its knowledge.

Section 2 discusses related work, especially ontology usage and the linguistic and psychological aspects. Section 3 explains the details of the linguistic problems and how they can be tackled with ontology knowledge. Section 4 explains RESI and its application. This includes an architectural overview as well as a description of the tool's functionality. Section 5 evaluates a case study that has been conducted with specifications that have been discussed in other papers concerning the subject of requirements engineering. The paper closes with the limitations in Sec. 6 and the outlook in Sec. 7.

2. Related Work

Requirements engineering is concerned with the elicitation of high level goals. These goals are to be achieved by the envisioned system [8]. Requirements engineering includes the refinement of such goals and their operationalization into specifications of services and constraints.

In 2000, Nuseibeh and Easterbrook [9] drafted a road map which shows future research areas to solve the problems in requirements engineering. They especially emphasize the development of new techniques and bridging the gap between contextual inquiry and formal representations. In 2007, Cheng and Atlee [10] wrote a detailed summary about the state of the art of requirements engineering. They categorize the various topics and try to predict the future of requirements engineering research. Since requirements engineering consumes a lot of time during the software engineering process and has long-term effects to every successive stage of the development process, focusing on its improvement is desirable. Cheng and Atlee conclude that it is important to realize that requirements define the problem, not the software itself. They show that requirements engineering activities — in contrast to other software engineering activities — are more iterative, involve more players who have more varied backgrounds and expertise, require more extensive analyses of options, and call for more complicated verifications of more diverse components (e.g. software, hardware, human). These mostly complex and time consuming manual tasks do not yet have pervasive tool support [10]. Automating parts of this procedure could not only speed up the processing times of requirements but also decrease error rates.

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 textual specifications that are legible and do not need special training. Additionally, specifications signed by the customer and the contractor are expected to be legally final and binding. Depending on the interpretation of the specification, this might lead to serious legal or liability problems.

As a possible solution, formal representations of natural language formalize the semantics of statements. This is impossible without a loss of usability or information. Formal specification languages are often perceived as difficult to use by practitioners [11]. Formal approaches need especially trained and skilled analysts to formalize natural language requirements [12]. Also after the formalization, it is hard for stakeholders to take part in further discussions about the requirements. They are not trained to think, act, and talk in formal representations like predicate logic or similar techniques [13]. So far, there is almost no evaluation of how well requirements engineering research is reflected in industrial applications.

2.1. *Ontologies in requirements engineering*

Today, several approaches use domain specific ontologies to cope with the problems that occur in the requirements engineering process [14–18]. They use ontology based systems to correctly classify textual information that has been delivered

from stakeholders. Other projects for example make sure that the correct (domain specific) wording is used when the specification documents are elicited from different stakeholders. Some projects have a narrow field of application and are specified for certain conditions and use cases [19, 20]. Until today, real world applications as listed in [3] have not yet adopted many of the solutions resulting from current research.

2.2. *Problems in specifications*

Related work in requirements engineering demands for complete, correct and unmistakable specifications [21]. But there are only a few papers that describe which concrete problems can occur and which aspects a human analyst should consider. Specifications in natural language make it difficult for the reader to comprehend what the customer tried to express. The specification should only leave room for one distinct interpretation and fully match the customer's idea.

Berry *et al.* [1] wrote a handbook to help writers of specifications 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. They are omnipresent in every language. This is due to two steps in requirements elicitation:

(1) The customer transforms reality (i.e. the desired software) into a mental picture (the idea of the software). During this process, the customer filters features that seem unimportant to him and focuses on things he deems important.
(2) The mental picture (the idea) is converted into natural language when the requirements are being communicated. Transformation and arbitrary filtering takes place here as well.

According to Bandler [22], Rupp divided the linguistic defects into three categories: deletion, generalization and distortion. We explain these categories and describe their manifestations.

2.2.1. *Deletion*

Deletion is the process of omitting 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 omitting necessary details in a specification, the specification

is incomplete. Some of the mentioned manifestations of deletion are:

- Incompletely specified process words[b] need further specification.
- Modal verbs of possibility need further specification *why* these actions can (or cannot) be taken.
- Modal verbs of necessity need further specification of what to do in the exceptional case when the desired behavior cannot be met.

2.2.2. *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.
- Incompletely specified conditions (*If* ...) also need an else-case.
- 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.

2.2.3. *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.
- Stretched verbs[c] are often less precise than the according normal verb.

2.3. *Solutions*

Solving the problems mentioned in Sec. 2.2 has many facets. One possible approach are inspections which base on the ideas of Fagan [23]. This technique has been used for several decades [24] and offers a proven approach to finding mistakes of specifications in a manual but half-deterministic way [25]. Since these inspections are sometimes seen as one-dimensional [4], Kamsties *et al.* provided an improved inspection type [26].

[b]Process words are mostly verbs or predicates [5].
[c]A stretched verb is a complex predicate composed of a light verb and an eventive noun.

2.3.1. *Constricted natural language*

Another approach to avoid errors in specifications is to control the natural language in which the specification is written. This means that the specification author needs to follow patterns for sentence structures and also has only a subset of the language available for use. The latter of which ensures that the amount of possible ambiguities is kept to a minimum. One of the most popular approaches is *Attempto Controlled English* (ACE) [27].

Denger *et al.* [28] present a pattern-based approach to avoid ambiguities. They focus on embedded systems and explain that it is difficult to merge existing specifications into a pattern-based form.

Propel [29] is another solution which uses a constricted language. The Propel system requires the parallel development of a finite state machine in combination with a natural language specification. The natural language specification is then constricted by the model patterns (*Disciplined Natural Language*). Since both representations have to be kept consistent, one always has the advantage of a formal and non-formal specification representation. The problem is the huge effort which needs to be undertaken to achieve consistency.

Constricting natural languages leads to disadvantages that are similar to using concrete formal representations such as first-order predicate calculus (FOPC). First, the corresponding system needs to be learned. Second, the approach cannot be directly used for existing specifications, but only for new ones.

2.3.2. *Criteria and metrics*

Davis *et al.* [30] present 24 criteria to measure the quality of a specification with the help of metrics. They point out that the perfect specification cannot exist since the individual quality criteria affect and sometimes even contradict each other (e.g. legibility vs. redundancy).

Wilson *et al.* [31] examine the occurrences of certain expressions in specifications. After that, they categorize these expressions and create metrics which can be used as quality indicators for the specification, such as completeness, consistency, etc.

Chantree *et al.* [32] engage with ambiguities in specifications. Existing ambiguities are supposed to be found automatically and then manually removed by the specification editor. They differentiate between potentially dangerous ambiguities and non-dangerous ones. Potentially dangerous ambiguities can have different meanings for different readers. These are the ones that need to be replaced in the specification. The heuristics eliminate ambiguities which people interpret easily, leaving the so-called nocuous ones to be analyzed and rewritten by hand. The nocuous ambiguities (potentially dangerous ones which have not been discovered as ambiguity by the reader) were also discovered during our evaluation. See also Sec. 5.6.

Fabbrini *et al.* [33] created a tool called QuARS (Quality Analyzer of Requirement Specification). QuARS checks natural language specifications for certain words

which are indicators for potential problems. According to the frequency of the indicators, the sentences are then flagged as problematic or non-problematic to point the user to potential flaws in the specification. Berry *et al.* also offer an extension to QuARS in their *New Quality Model* work [34]. Fantechi *et al.* [35] also use QuARS as conceptual basis for their own metrics.

Pisan [36] takes another route and begins with specifications that are evaluated and known to be of high quality. These already existing specifications or parts thereof are then being used to complete new specifications. The idea is to match the analogies of old specifications and new specifications.

3. Solving Linguistic Defects with Ontologies

All the problems mentioned that can occur in specifications can be classified and prioritized. We focused on the high priority problems when developing RESI as listed in [2]. The problems were examined for resolvability with ontologies by mapping linguistic elements to concept of the various ontologies.

These problems are:

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

3.1. *Nominalization*

3.1.1. *Problem*

Nominalizations hide complex processes in a single noun. Information which is vital for the specified process description is lost.

3.1.2. *Example*

```
After signing in, the user is redirected to his personal page.
```

Signing in is a complex process which includes many other tasks and has side-effects. Therefore this process needs to be described in more detail without the nominalization.

3.1.3. *Solution*

The ontology tests if the nouns in the specification are a nominalization. The ontology then recommends which main verb to use via the RESI GUI.

3.2. *Incompletely specified process words*

3.2.1. *Problem*

Process words often require more than one argument to be completely specified. If a process word lacks arguments, important information for the specification process description are missing. A frequently seen special case are passive sentences which lack an acting entity.

3.2.2. *Example*

```
Username and password are entered.
```

Who enters the data? Where is the data entered?

3.2.3. *Solution*

The ontology delivers argument lists for the corresponding process word. If these arguments are matched with the actual content of the sentence, one can deduct which arguments is missing and need to be specified.

3.3. *Nouns without reference index*

3.3.1. *Problem*

Incompletely specified nouns are called nouns without reference index. If a noun refers to a set, it is not always clear which part of the set is being referenced. The wrong usage of articles and quantifiers are examples for this problem.

3.3.2. *Example*

```
The personal data can be seen on the personal page.
```

What personal data? All the data the system has, or only the data relevant to the user, such as email, username, etc.?

3.3.3. *Solution*

The ontology determines the correct meaning of an article or quantifier. The meaning in the sentence is then being compared with the intended meaning to discover if changes to the specification are necessary.

3.4. *Incompletely specified conditions*

3.4.1. *Problem*

Conditional sentences describe the behavior which occurs under certain circumstances (`if... then...`). Many times the specification lacks the description of what happens if the mentioned premise is not met (`else...`).

3.4.2. *Example*

```
After the login, the user can access his personal page.
```

And what if the login was not successful? Will the user be able to try again? Will the account be locked for several minutes?

3.4.3. *Solution*

Ontologies as well as lexicons detect signal words which initiate conditions. This conditions can then be checked for completeness.

3.5. *Modal operators of necessity*

3.5.1. *Problem*

Modal operators of necessity are used to describe behavior which occurs in the common case. If so, the exceptional case also has to be described in the specification. Doing this can significantly decrease the amount of errors [37].

3.5.2. *Example*

```
The user credentials must be checked by consulting the database.
```

What if the database is not available? Is the process being canceled, are exceptions being handled? What is the standard procedure in the worst case?

3.5.3. *Solution*

Ontologies detect modal verbs of necessity and check for exceptions.

3.6. *Polysemy*

3.6.1. *Problem*

Terms with multiple meanings can be interpreted in different ways and lead to confusion. The reader might not know which of the corresponding meanings is the most suitable one. In the worst case, the reader assumes his interpretation to be correct whilst overlooking possible other meanings which are a much better description of the specification process. This problem is described in Sec. 2.3.2 as well as in Sec. 5.6

3.6.2. *Example*

```
Bank
```

The building? The financial institution? The edge of a river?

3.6.3. *Solution*

The ontology determines possible (semantic) interpretations for a word (as also done by [4]). This ensures that the reader can check possible meanings that are proposed by the ontology and crosscheck them with his own understanding.

3.7. *Words with similar meaning, cluttering*

3.7.1. *Problem*

Several terms which refer to the same object often confuse the reader and are hotbed for mistakes.

3.7.2. *Example*

```
Every client has a personal page which shows the personal data of the
    user.
```

Are `client` and `user` equivalent? Or do they have different characteristics?

3.7.3. *Solution*

Ontologies determine synonyms and other similar word pairs in the text. One term is being substituted by the other term to decrease confusing and increase the understanding. It might even be possible to introduce a brand new term that replaces the terms originally used in the specification text.

4. RESI — Requirements Engineering Specification Improver

RESI is based on prior work [38] of the AutoModel research project. AutoModel is part of the SaleMX [39] approach to automate the creation of requirement models (UML) from textual specifications. RESI is a part of AutoModel (see Fig. 1) and covers the linguistic processing of textual information. This work focuses on improving textual specifications before the automatic modeling takes place.

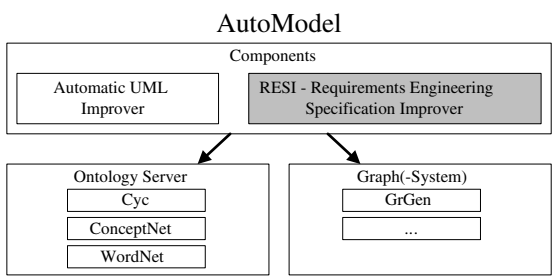

Fig. 1. The AutoModel research project.

A human analyst uses common sense to find problems. RESI uses various ontologies (WordNet [40], ResearchCyc [41], ConceptNet [20] and YAGO [42]) to access "common sense". These ontologies provide RESI with knowledge to identify problems in specifications and to suggest the corresponding solutions. Therefore, RESI can help the analyst to avoid several of the problems mentioned in Sec. 2.2.

4.1. *The RESI workflow*

RESI works in four steps as depicted in Fig. 2.

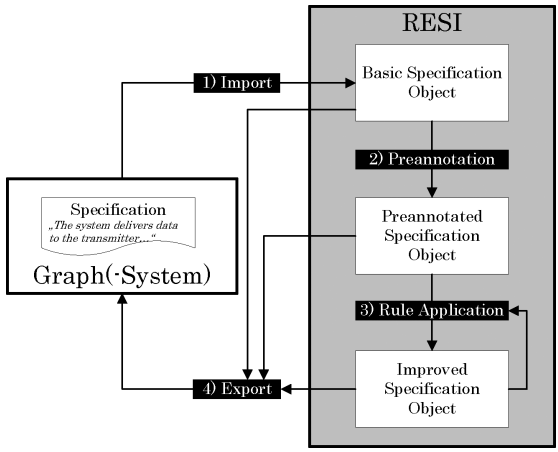

Fig. 2. The RESI workflow.

4.1.1. *Import of specification*

First, RESI imports a specification. The specifications have to be in a format RESI accepts, i.e. a graph-representation of the textual specification [43]. Prior to import, the graphs are easily created using the SALE model [44] from our other research project. RESI is an open platform which offers an interface if other forms of specification representations are desired.

4.1.2. *Preannotation of specification*

RESI then adds additional information to the specification. Preannotation helps the tool to distinguish the words in the sentence while it later increases the quality of the queries to the ontology. Each word is tagged with the according *part of speech* (POS) information and — if it is a noun or verb — with its base form. Taggers execute this job fully automatically. If desired, the RESI user makes manual changes.

4.1.3. *Application of rules*

RESI then applies the so called *rules*. A rule is an implementation of one of the solutions mentioned in Sec. 3. The rules and the corresponding ontologies are selected

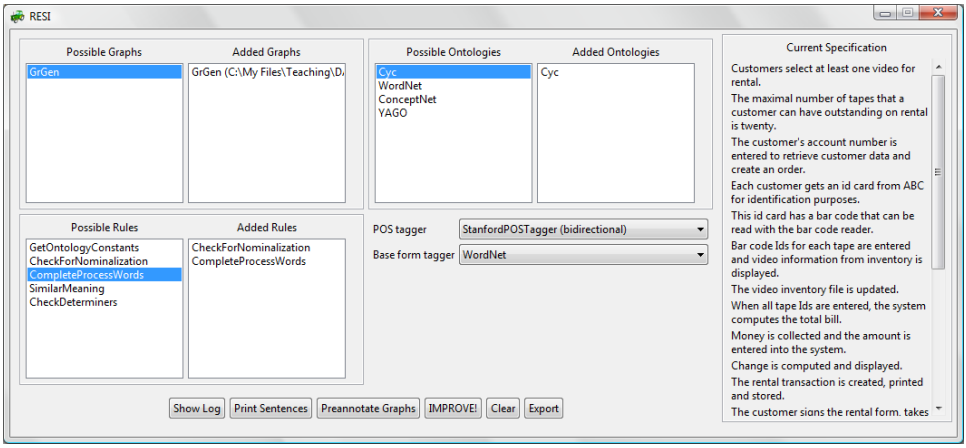

Fig. 3. The RESI GUI.

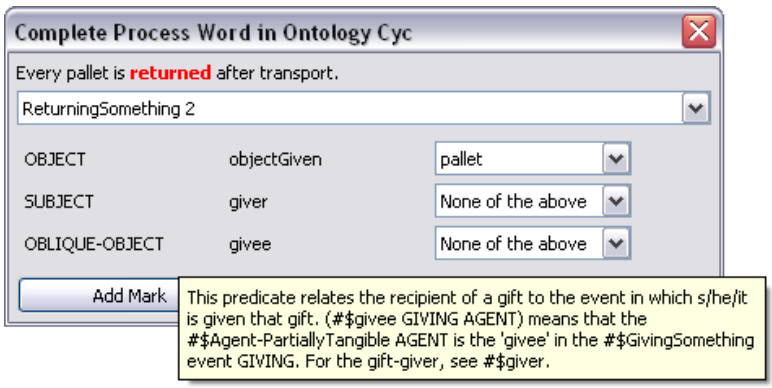

Fig. 4. User dialog for process word completion.

in the user interface. A screen shot can be seen in Fig. 3. 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: the problematic sentence *"Every pallet is returned after transport."* is written in passive voice and therefore lacks arguments of the process word, this leads to the output depicted in Fig. 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. RESI displays the possible argument lists to the user in the GUI (see Sec. 4). The possible list of arguments is shown below.

ReturningSomething

- SUBJECT: *performedBy*
- OBJECT: *objectOfPossessionTransfer*

ReturningSomething 2

- OBJECT: *objectGiven*
- SUBJECT: *giver*
- OBLIQUE-OBJECT: *givee*

The ontology states that *return* needs to have an *objectGiven*, a *giver*, and a *givee*,[d] but only the *objectGiven* can be found in the examined sentence. The *subject* and the *oblique-object* need to be further specified. The RESI user marks the sentence to ask the customer who the *giver* and the *givee* are.

The use of determiners and quantifiers is also automatically discovered by RESI as shown in Fig. 5. RESI recognized the quantifier *every* and suggests that it should be specified as *All* (*without exception*). The user changes the quantifier detailing if RESI is wrong. If the correct quantifier is unclear, the user sets a mark and comments the sentence. The correct meaning can then be specified in coordination with the customer later.

Any rule 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.

4.1.4. *Export of specification*

After the specification has been 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 any time.

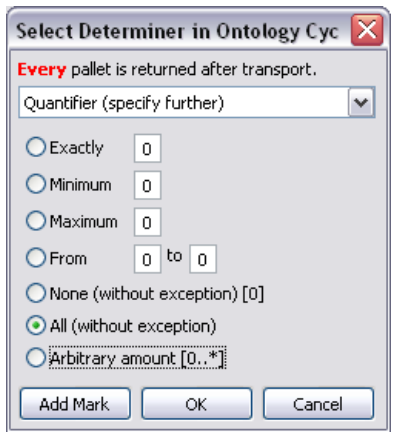

Fig. 5. User dialog for determiners and quantifiers.

[d]The exact ontology definition is shown in the tool tip in Fig. 4.

4.2. *RESI architecture*

RESI comprises 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 Fig. 6.

4.2.1. *Rules*

As mentioned in Sec. 4.1.3, a rule represents a function which 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 Sec. 3.7) gets all nouns from the graph and checks which of the nouns have similar meanings using the ontology. RESI utilizes the *generalization* and *is a* similarity-reasoning of

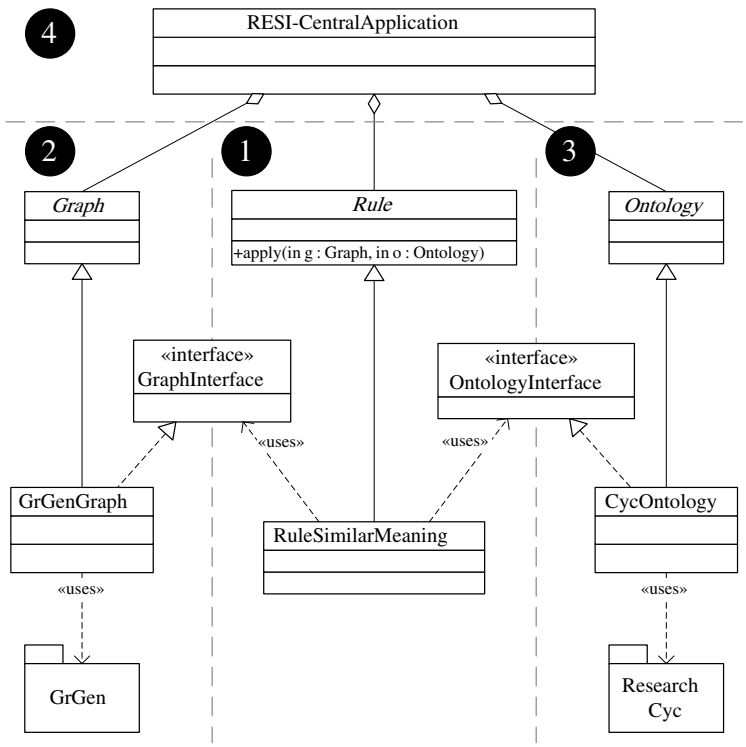

Fig. 6. UML layout of RESI (for detailed description see Sec. 4.2).

the ontology. The ontology is used to compare words from the text and to check them for similarity. If a pair of synonyms is found in the text, the user is asked if he wants to replace one noun with another to decrease cluttering. This makes the specification easier to read since *ThingA* is always called *ThingA* and not *ThingB* or *ThingC*. If the user accepts, the nouns are unified in the graph.

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

4.2.2. *Graphs*

A graph object allows the tool to read from and write to the specification. This is not possible for plain text files yet. The graph class supports a rule, if it implements the graph interface of the rule.

For now we have implemented a graph class that imports GrGen [43] graphs from a file (depicted as *GrGenGraph* in Fig. 6). This graph class supports all rules. RESI can be extended to read any representation of a natural language specification by implementing a new graph class.

4.2.3. *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 a 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.

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.

4.2.4. *Central application*

The central application controls which of the components mentioned above interact (see screen shot in Fig. 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. For example, the interaction could look like Fig. 7 which shows the sequence of the example mentioned in Sec. 4.2.1.

The central application also includes the preannotation (see Sec. 4.1.2). 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* [45] as POS tagger and WordNet as base form tagger.

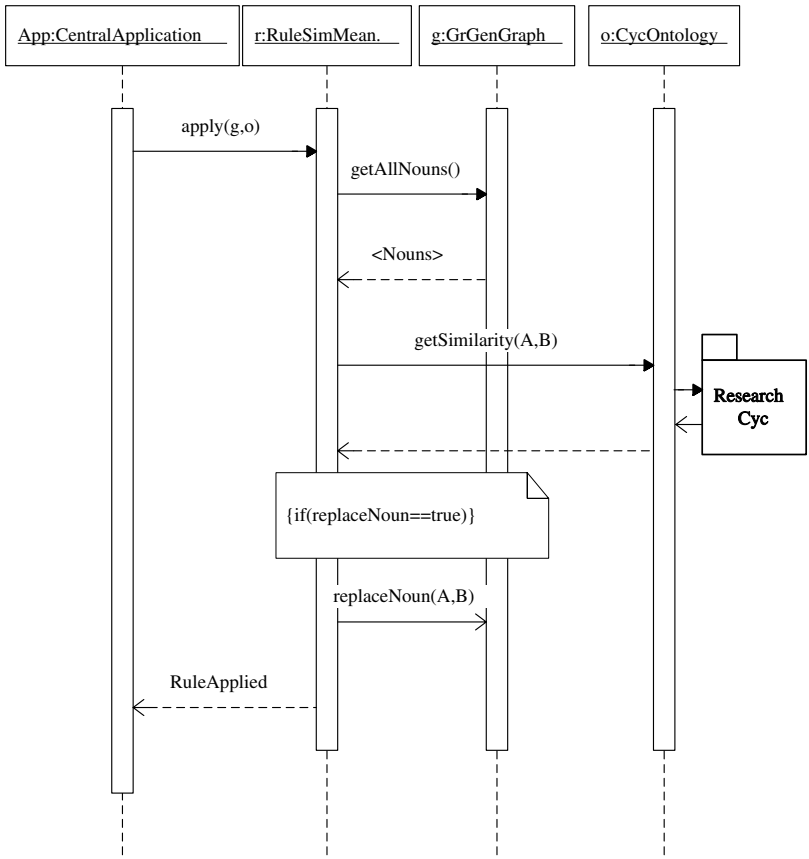

Fig. 7. Sequence diagram of RESI class interaction.

5. Evaluation

We evaluated the tool RESI with two specifications that have been widely used and examined in other papers: one is the example of a video rental store as used in [4], the other one is a small text from Berry's ambiguity handbook [1] (denoted as *Spec1* & *Spec2* respectively in Table 1).

5.1. *The example specifications*

The video rental specification (*Spec1*) text is comprised of 17 sentences with a total amount of 222 words. The text is listed below.

As can be seen in Table 1, this excerpt of a specification 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 possible additional meanings.

The example taken from [1] (*Spec2*) consists of 99 words and 6 sentences. Each sentence describes the individual use of a pressure system monitor which

Table 1. Case study: RESI specification improvements.

	Spec1	*Spec2*	*Spec3*
Statistics			
# Sentences	17	6	1
# Words	222	99	7
# Characters (no spaces)	1036	486	38
# Nouns	76	27	2
# Verbs	36	14	1
KPI			
Ambiguity Statistics:			
# 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

has automatic functions, e.g. to initiate safety injections, that can be overridden manually. It is a simplified version of the ESFAS (Engineered Safety Feature Actuation System) requirements that were originally introduced by Courtois and Parnas [46]. Kamsties *et al.* also used this text in their research concerning inspections for requirements documents [47]. They discuss in detail one of the problems that can be found using their inspection technique. This problem is also automatically detected by RESI.

5.2. *Direct comparison*

We also directly compared the results yielded with RESI to existing publications. Kiyavitskaya *et al.* [4] manually transcribed the listed *Spec1* and detected several errors. The results of the *RESIfication*[e] of *Spec1* are listed in Fig. 8. The detection rates are explained in more detail in Table 2. The first two columns show the absolute detection rates compared to total appearances of flaws. The last two columns list the relative detection rates proportionally. No false negatives were found. False positives cannot be rated since the approach tries to detect possible flaws. These flaws have then to be checked by the analyst and customer — their impact cannot be detected objectively at this moment and lies in the eye of the beholder.

[e]Checking the specification with RESI.

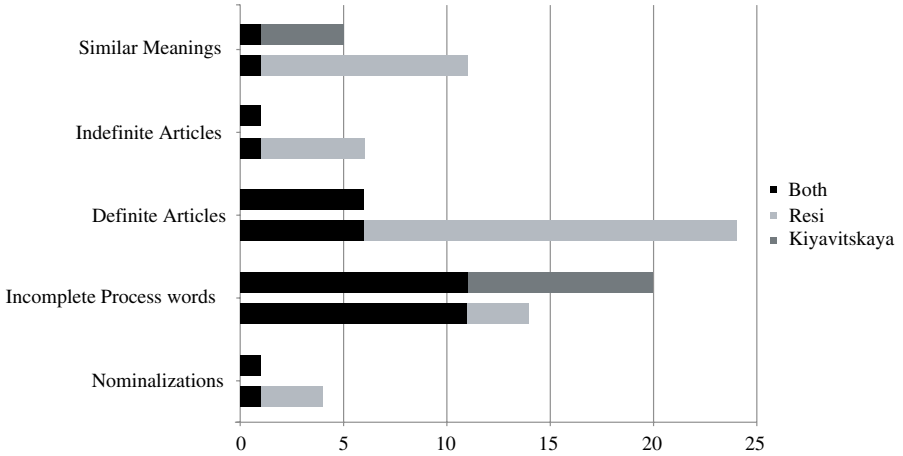

Fig. 8. Number of found problems in specification *Spec1*.

Table 2. Case study: absolute detection values.

Statistics	RESI abs.	Kiya. abs.	RESI (%)	Kiya. (%)
Similar Meanings	11/15	5/15	73	33
Indefinite Articles	6/6	1/6	100	17
Definite Articles	24/24	6/24	100	25
Imcomplete Process Words	14/23	20/23	61	87
Nominalizations	4/5	1/5	80	20

5.2.1. *Check for words with similar meaning and detect cluttering*

RESI suggests replacing 11 terms. The only sensible substitution is `video` with `tape`. The other suggestions are either incorrect (e.g. the substitution of `clerk` with `system`) or unnecessary (e.g. the replacement of `number` with `amount`). Compared to RESI, Kiyavitskaya *et al.* found four different substitutions. Since RESI has to compare every noun with every other noun in the specification, the processing time increases with the power of two ($O(n^2)$).

5.2.2. *Check for quantifiers and (in-)definite articles*

Kiyavitskaya *et al.* point out six definite articles which have no reference index. RESI detects all 24 definite articles, including the six. Additionally, it extracts the 6 indefinite article which all mean "exactly one" in their case. Therefore RESI suggests to replace these indefinite articles with *one*. The tool also finds the text's eight quantifiers, though it does not correctly reflect the meaning of `at least one`, because RESI only examines single words. This leads to the interpretation that `one` is being recognized as *exactly 1*. The meaning of `each` is also not correctly asserted with the corresponding number value in the ResearchCyc ontology. The correct meaning

cannot be suggested to the user automatically, though the possible mistake in the specification is still pointed out to the RESI user.

5.2.3. *Check for incomplete process words*

RESI finds 14 incompletely specified process words. Some of which lack more than one argument, therefore RESI points out 18 missing arguments in total. Kiyavitskaya *et al.* focus on passive constructs in the sentences which occur 20 times in the specification. RESI concurs with 11 of these but cannot find the other nine (five of which are the same verb `enter`). This is due to the fact that the used ontologies do not carry the argument lists for these verbs. Therefore RESI is "blind" in these occurances. But RESI also finds three additional process words which lack arguments which Kiyavitskaya *et al.* probably oversaw. This strengthens our opinion that human analysts tend to miss flaws (for more see Sec. 5.6).

5.2.4. *Check for nominalizations*

RESI detects all four nominalizations in the text. These are `order`, `identification`, and `return` twice. Kiyavitskaya agree that a problem exists with the word `identification`, but they did not find the other nominalizations in the text.

5.2.5. *Check for ambiguous words*

RESI detects 44 possible ambiguities in the 222 words of *Spec1*. 12 of these words are precisely explained in detail through the explanation of the ontology. The detecting rule of RESI works similar to the approach of Kiyavitskaya *et al.* and struggles with the same problems: most of the possible ambiguities can be omitted by a human reader. A human reader perceives the words in their corresponding context and usually chooses the correct meaning while reading the text. This is especially true if the reader is a domain expert. RESI lets the user choose from a list of possible meanings and marks these meaning in the text. Therefore even non-domain experts who are involved in the production process can easily understand the improved specification later.

5.3. *Different ontologies*

The different ontologies provide only limited information due to the differences in their conceptual construction. Depending on the use case, one or the other ontology is a lot more usable or not at all. ResearchCyc seems to be one of the more general ontology approaches which supports all rules that RESI checks.

WordNet supersedes ResearchCyc in terms of ambiguities though it suffers from the same problems, but even more severe. WordNet detects 27 ambiguous words in *Spec2* (compared to 15 from ResearchCyc) which can have an additional 191 (ResearchCyc: 42) meanings next to the desired one. This leaves the user with too many options to choose from.

Checking words for similar meanings is supported by all ontologies. Unfortunately, WordNet offers three possible substitutions which are pointless. YAGO on the other hand takes almost 20 minutes with a search depth of two without finding a suitable suggestion. ConceptNet also does not deliver serious suggestions and takes several hours to process the specifications which is unacceptable for the user.

This leads to the conclusion that only ResearchCyc and partially WordNet can be used to comfortably process a specification. Therefore these two knowledge bases are used as default ontologies for RESI and provide the best average results. Specialized domain ontologies will lead to even better results, but these were not available for this evaluation.

The ontologies that appear to be working best are used in conjunction during the RESI process. All ontologies can be integrated into the decision making process though we use default settings which ontology to use for which kind of query. If multiple ontologies are queried and provide information for the same term, different micro theories (knowledge classifications in ontologies) might predominate in terms of quality and speed towards their query answer. The GUI of RESI does not yet allow to show a list of all ontology answers to the user.

5.4. *Scalability*

RESI's several rules scale differently for large specifications. Rules for completing process words or finding the correct meaning for quantifiers scale in a linear fashion. Rules for finding and replacing words with similar meaning scale a lot worse since their search try increases exponentially with every new sentence that appears. The corresponding rule then has to check all words in all sentences with each other (see Sec. 5.2.1).

RESI works in various domains and the results mostly rely on the quality of the knowledge base. It is not specifically tailored to meet software engineering requirements. RESI can also be used to improve any other kind of specification. Our prototypes inspected specification reaching from the mentioned video rental specifications, system descriptions and also the rules for the game of chess.

The post processing of the information RESI gathers is transformed into UML models. This is indeed a special purpose for software engineering.

5.5. *Error handling*

RESI has rudimentary error correction in the sense of spelling errors in text and ontology query misses. Spelling errors are corrected using the ontology database as well as WordNet features to remedy typos. If an ontology does not provide any kind of semantic information for a query RESI fires, the user can always manually override the changes made on the text.

During runtime, RESI utilizes various ontologies. If any of these ontologies fail during a server crash or similar, RESI notifies the user that it cannot receive certain

information. In this case, RESI may not be as much of service to the analyst as it could be.

5.6. *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" [48] 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 these specification flaws without RESI in such an early stage. We had four specialists testing the tool who came up with three to eight flaws (average: 4.7) additional flaws in *Spec1*. To our knowledge, traditional tools for requirements engineering do not provide this kind of user support, but rather extensive management features.

Chantree *et al.* [32] 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 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 1:

- *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 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.

6. Limitations

RESI is not yet fully evaluated. First experiences show promising results in speed-up and precision in detecting flaws. There are various other aspects we would like to be able to detect when working through textual specifications, such as non domain style language which should be changed to domain specific styles, etc. We plan to further evaluate the tool in additional studies with more substantial cases. For this reason, we received original specifications from various companies in different industries, such as automotive, medical engineering, software, and software consulting. We are setting up a study to examine RESI's impact on software quality compared to human software analysts and architects.

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 in the new revision of the textual specification and is planned to track the changes that were made since the original specification.

7. 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 specifications which applies ontologies to provide "common sense" for machines. We showed that software utilizing semantics is indeed capable of solving some of the issues analysts deal with daily though there certainly is a long way ahead of us.

A tool has many advantages over a human centered process. This way, we ensure that the quality of the requirements does not rely solemly 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.

Appendix A. Ontology Queries

This is the example of a query to the ResearchCyc ontology to recover the argument lists of a process word. The example sentence used is listed in Sec. 5.6.

The queries to ResearchCyc are written in CycL [49], a ResearchCyc specific language. Predicates and fixed terms are prefixed with `#$` and variables are denoted using a `?`. ResearchCyc is structured into so called Microtheories. The relevant queries for RESI are being made to the Microtheory *GeneralEnglishMt*.

Argument lists can be identified with a single query. Before that, the process word (in this case `return`) has to be morphed into a word constant. This is done be capitalizing the first letter of the word and adding `-TheWord` to the corresponding word string.

```
(#$verbSemTrans #$Return-TheWord ?SENSECOUNTER ?FRAMETYPE
   ?FRAME)
```

The variable `?FRAME` then holds the argument list in the following format:

```
(#$and
   (#$objectGiven :ACTION :OBJECT)
   (#$isa :ACTION #$ReturningSomething)
   (#$giver :ACTION :SUBJECT)
   (#$givee :ACTION :OBLIQUE-OBJECT))
```

The line with the predicate `#$isa` denotes the meaning of the word in its context. Only if the meaning of the process word matches the argument list, the argument list is returned. The other lines show the arguments in their semantic (e.g. `#$objectGiven`) and their syntactic (e.g. `:OBJECT`) roles.

To convert the semantic roles into something the RESI user comprehends, the descriptions of the semantic roles are also determined:

```
(#$comment #$giver ?COMMENT)
```

To fill these argument lists with recommendations from the corresponding sentence, RESI checks every argument with which value/word it could be filled:

```
(#$arg2Isa #$objectGiven ?WHATFITS)
```

This leads to the result that an `#$objectGiven` can be every `#$SomethingExisting`.

Finally, every word of the sentence is checked whether it does suffice as one of the needed arguments. RESI checks if the word which is supposed to be used as argument (`#$SomethingExisting`) is a generalization of the meaning of the word (`#$Pallet-TransportationConstruct` for `pallet`) from the sentence:

```
(#$genls #$Pallet-TransportationConstruct
   #$SomethingExisting)
```

If this query delivers a true return value, then the according word is inserted into the argument list. After checking all arguments and all words as described, the list is being returned to RESI.

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/ambiguity Handbook.pdf.

[2] C. Rupp and die SOPHISTen, *Requirements-Engineering und Management*, 4th ed. (Carl Hanser Verlag, 2006).

[3] Volere, List of requirement engineering tools, 2009. [Online]. Available: http://www.volere.co.uk/tools.htm.

[4] 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.* **13**(3) (2008) 207–239.

[5] C. Rupp, Requirements and Psychology, *IEEE Software*, May/June 2002.

[6] D. Jurafsky and J. H. Martin, *Speech and Language Processing* (Prentice Hall, 2000).

[7] S. J. Körner and T. Gelhausen, Improving automatic model creation using ontologies, in *Proceedings of the Twentieth International Conference on Software Engineering & Knowledge Engineering*, July 2008, pp. 691–696.

[8] A. van Lamsweerde, R. Darimont and E. Letier, Managing conflicts in goal-driven requirements engineering **24**(11) (1998) 908–926.

[9] B. Nuseibeh and S. Easterbrook, Requirements engineering: a roadmap, in *ICSE '00: Proceedings of the Conference on the Future of Software Engineering*, New York, NY, USA (ACM Press, 2000), pp. 35–46.

[10] B. H. C. Cheng and J. M. Atlee, Research directions in requirements engineering, in *Proc. Future of Software Engineering* (*FOSE '07*), 23–25 May 2007, pp. 285–303.

[11] S. Konrad and B. H. Cheng, Facilitating the construction of specification pattern-based properties, *IEEE International Conference on Requirements Engineering*, 2005 pp. 329–338.

[12] W. M. Adam Pease, An english to logic translator for ontology-based knowledge representation languages, *IEEE 0-7803-7902-0/03*, 2003, pp. 777–783.

[13] C. L. Heitmeyer, R. D. Jeffords and B. G. Labaw, Automated consistency checking of requirements specifications, *ACM Trans. Softw. Eng. Methodol.* **5**(3) (1996) 231–261.

[14] H. Kaiya and M. Saeki, Ontology based requirements analysis: lightweight semantic processing approach, in *Proc. Fifth International Conference on Quality Software (QSIC 2005)*, 19–20 Sept. 2005, pp. 223–230.

[15] H. Kaiya and M. Saeki, Using domain ontology as domain knowledge for requirements elicitation, in *Proc. IEEE International Conference Requirements Engineering*, 11–15 Sept. 2006, pp. 189–198.

[16] M. Saeki, Ontology-based software development techniques, *ERCIM News* **58** (2004) 14–15.

[17] Zhang, R. Witte, J. Rilling and V. Haarslev, An ontology-based approach for traceability recovery, 2006.

[18] W. Meng, J. Rilling, Y. Zhang, R. Witte and P. Charland, An ontological software comprehension process model, *3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering* (*ATEM 2006*), October 1st, Genoa, Italy, 2006.

[19] H. Liu and P. Singh, ConceptNet — a practical commonsense reasoning tool-kit, *BT Technology Journal* **22** (2004). [Online]. Available: http://larifari.org/writing/BTTJ2004-ConceptNet.pdf.

[20] 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.

[21] IEEE recommended practice for software requirements specifications, *IEEE Std 830-1998*, Oct. 1998.

[22] R. Bandler, *Metasprache und Psychotherapie: Die Struktur der Magie I* (Junfermann, 1994).

[23] M. E. Fagan, Design and code inspections to reduce errors in program development, *IBM Systems Journal* **15**(3) (1976) 182–211.

[24] A. Ackerman, L. Buchwald and F. Lewski, Software inspections: an effective verification process, *IEEE Software* **6** (1989) 31–36.

[25] N. Power, Variety and quality in requirements documentation, in *REFSQ'01: Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality*, pp. 165–170, 2001.

[26] E. Kamsties, A. V. Knethen, J. Philipps and B. Schätz, An empirical investigation of the defect detection capabilities of requirements specification languages, in *EMMSAD'01: Proceedings of the Sixth CAiSE/IFIP8.1 International Workshop on Evaluation of Modelling Methods in Systems Analysis and Design*, 2001.

[27] N. E. Fuchs, U. Schwertel and R. Schwitter, Attempto Controlled English — not just another logic specification language, *LNCS*, 1559, 1999, pp. 1–20.

[28] C. Denger, D. M. Berry and E. Kamsties, Higher quality requirements specifications through natural language patterns (IEEE Computer Society, Los Alamitos, CA, 2003), p. 80.

[29] R. Smith, G. Avrunin, L. Clarke and L. Osterweil, Propel: an approach supporting property elucidation, in *ICSE 2002: Proceedings of the 24rd International Conference on Software Engineering*, 2002, pp. 11–21.

[30] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, A. Ta and M. Theofanos, Identifying and measuring quality in a software requirements specification, in *Proceedings of the First International Software Metrics Symposium*, May 1993, pp. 141–152.

[31] W. Wilson, L. Rosenberg and L. Hyatt, Automated analysis of requirement specifications, in *ICSE '97: Proceedings of the 19th International Conference on Software Engineering*, May 1997, pp. 161–171.

[32] F. Chantree, B. Nuseibeh, A. de Roeck and A. Willis, Identifying Nocuous 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.

[33] F. Fabbrini, M. Fusani, S. Gnesi and G. Lami, The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool, in *SEW '01: Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, Washington, DC, USA (IEEE Computer Society, 2001), p. 97.

[34] D. M. Berry, A. Bucchiarone, S. Gnesi and G. Trentanni, A New Quality Model for Natural Language Requirements Specifications, 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.5268.

[35] A. Fantechi, S. Gnesi, G. Lami and A. Maccari, Application of linguistic techniques for use case analysis, *IEEE International Conference on, Requirements Engineering* (2002), p. 157.

[36] Y. Pisan, Extending requirement specifications using analogy, in *ICSE '00: Proceedings of the 22nd International Conference on Software Engineering*, New York, NY, USA (ACM, 2000), pp. 70–76.

[37] J. L. Elshoff, An analysis of some commercial pl/1 programs, *IEEE Transactions on Software Engineering*, (1976). [Online]. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=35893&arnumber=1702349&count=16&index=6.

[38] S. J. Körner and T. Brumm, Improving Natural Language Specifications with Ontologies, in *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering*, Knowledge Systems Institute, Ed., July 2009.

[39] 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.

[40] G. A. Miller, C. Fellbaum, R. Tengi, P. Wakefield, H. Langone and B. R. Haskell, WordNet. [Online]. Available: http://wordnet.princeton.edu/

[41] Cycorp Inc., ResearchCyc. [Online]. Available: http://research.cyc.com/

[42] F. M. Suchanek, G. Kasneci and G. Weikum, Yago: A Core of Semantic Knowledge, 2007. [Online]. Available: http://suchanek.name/work/publications/www2007.pdf.

[43] 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*, Lecture Notes in Computer Science (Springer, 2006), pp. 383–397, natal, Brasil. [Online]. Available: http://www.info.uni-karlsruhe.de/papers/grgen_icgt2006.pdf.

[44] 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.

[45] The Stanford Natural Language Processing Group, Stanford Log-linear Part-Of-Speech Tagger. [Online]. http://nlp.stanford.edu/software/tagger.shtml.

[46] 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.

[47] 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.

[48] C. H. D. Heath, *Made to Stick: Why Some Ideas Survive and Others Die*, 1st ed. (Random House, 2007).

[49] Cycorp Inc., The syntax of CycL. [Online]. Available: http://www.cyc.com/doc/handbook/oe/02-the-syntax-of-cycl.html.