ONTOLOGY-BASED SIMILARITY OF SOFTWARE CASES

Applying Ontology Reasoning to Software Retrieval

Lothar Hotz, Katharina Wolter
HITeC e.V., Department of Computer Science, University of Hamburg
hotz@informatik.uni-hamburg.de, wolter@informatik.uni-hamburg.de

Stephanie Knab, Arved Solth
HITeC e.V., Department of Computer Science, University of Hamburg
knab@informatik.uni-hamburg.de, solth@informatik.uni-hamburg.de

Keywords: Ontology engineering, Metamodelling, Description Logic, Taxonomical Similarity

Abstract: In this paper, we use Description Logic based classification and taxonomical similarity computations for facilitating software reuse. For this purpose we map a metamodelling-based software representation to an ontology. The ontology is classified by a Description Logic reasoner, which makes implicit taxonomical relations explicit. This classification is the basis for the computation of taxonomical similarity. The approach is tested with several industrial software applications.

1 INTRODUCTION

Reusing software is still an open problem in current software practice. Former approaches to solve this problem concentrated on the code level. However, the impact of reuse can be increased when integrated earlier in the development process, e.g. on the level of requirements. Thus, a goal is to design an effective reuse process based on requirements.

In this paper, we describe one step towards this goal. The general reuse process of our approach looks as follows: Starting with an initial requirements specification a repository is searched for similar specifications. This repository contains former software development projects stored in the form of software cases. A *software case* comprises a problem (requirements) and a solution (architecture, design, and implementation), similar to case-based approaches, for example described in (Bergmann et al., 1999). Each requirements specification is mapped to appropriate elements of the solution.[1] The retrieved case is intended to be reused by modifying those parts that need rework and keeping those parts that can be reused without modification.

Retrieval of similar requirements specifications from a repository is a key prerequisite for this reuse process. In the following, we explain how this can be achieved by an ontology-based similarity measure. In order to enable reuse on the basis of meaning, the specifications need to provide more than meaningless strings. The words in one requirements specification need to be linked to an ontology, where the semantics of the words are defined. Our approach depends on requirements specifications in a machine processable form as provided by the requirements specification language RSL (Śmiałek et al., 2007), which we use (Śmiałek et al., 2005; Śmiałek et al., 2007) or ATTEMPO (Fuchs et al., 2005). Prerequisites for our approach are requirements specifications in a machine processable form.

Such a process should enable software developers to express customer needs with more or less formal notations of requirements to gain a machine processable form. For this reason, requirements specification languages provide a notation and tool support for expressing requirements by means of use cases, constrained language sentences (like subject-verb-object, SVO), natural language hypertext etc. (Fuchs et al., 2005; Varelas et al., 2005; Śmiałek et al., 2005; Śmiałek et al., 2007). By doing so, still no *meaning* of

[1] For a complete description of a software case's internal structure we refer to (Śmiałek, 2006).

the terms used for specifying requirements is represented, because such languages basically provide structuring of meaningless strings. Furthermore, for supporting reuse between different software applications terms of one application should somehow be related to terms in other ones. To enable reuse on the basis of meanings, terms used in requirement descriptions of different software applications have to be linked to ontologies, where the semantics of terms are defined.

In this paper, this link is realized by a new combination of formal requirements specifications provided by the requirements specification language RSL with ontology reasoning provided by Description Logics (DL) (Baader et al., 2003). The basic idea is to cover all former software cases in an ontology (*DL-model*) and use the reasoning facilities of a Description Logic reasoner (*DL-reasoner*) to classify the software cases, i.e. their deeply structured elements. When classified, the software cases have certain taxonomical distances between each other. As shown for example in (Salotti and Ventos, 1998) the taxonomical distance can be used for computing the similarity between classes of a taxonomy, i.e. in our case between elements of software cases. Computing the similarity in this way takes into consideration the semantics of used terms.

The remainder of the paper is structured as follows: we start with the description of the requirements specification language RSL (see Section 2). This language is defined in a metamodel, which formalizes various requirement representation facilities like constrained language sentences. This language links the words used in the requirements specification of one software case to WordNet (Fellbaum, 1998) as a basic ontology. This provides the basis for our semantic-based similarity measure. However, for applying an DL-reasoner we had to map the requirements specifications to a DL-model (see Section 3). How a DL-reasoner can be used for classifying a set of requirements specifications and thus, provides new taxonomic relations is described in Section 4. Our similarity measure compares the software cases taking into account the new taxonomical relationships between the cases, the meaning of used terms, and the structure of sentences (Section 5). We implemented the developed approach and evaluated it with industrial software cases taken from software development organizations. These results are reported in Section 6. The approach is discussed in Section 7 and finally the paper is summarized in Section 8.

2 FORMALIZING REQUIREMENTS

Most requirements specifications are still written in natural language. However, natural language is potentially ambiguous, which complicates automatic processing such as similarity estimation. Therefore, the new requirements specification language RSL was defined, which enables precise requirements specifications and is comprehensible by humans at the same time. RSL is based on the same metamodelling approach as used for specifying the Unified Modeling Language (UML, see (OMG, 2007)), i.e. the Meta Object Facility (MOF, see (OMG, 2006)). The RSL metamodel specifies the structure of valid requirements specifications and is integrated in a prototypical tool. The tool supports the user writing specifications and ensures that each specification is an instance of the metamodel. In the following, we describe the RSL elements relevant for the scope of this paper. For a complete introduction of RSL we refer to (Kaindl et al., 2007; Kaindl et al., 2008). Each Requirements Specification[2] is part of a Software Case, which combines all artifacts developed in one software development project (e.g. Architectural Model, Detailed Design Model, and Source Code). However, our similarity measure compares the requirements specifications only. This is based on the assumption that if two software cases have similar requirements their other artifacts are similar too and thus, the reuse potential is high.

RSL allows to write requirements specifications in form of less formal NaturalLanguageHypertextSentences or more formal ConstrainedLanguageSentences. For comparing the natural language parts of RSL requirements specifications an Information Retrieval approach is most appropriate (see (Wolter et al., 2008)). The constrained form of sentences has the advantage of being syntactically unambiguous and semantically rich. For this reason, our approach focuses on this part of the specifications.

One type of ConstrainedLanguageSentences provided by RSL are Subject-Verb-Object (SVO) sentences. They can be used to write SentenceLists and ConstrainedLanguageScenarios in order to define Requirements in detail. Such ConstrainedLanguageScenarios can for example contain sentences like: *The customer changes the order. The system confirms the changes.* etc.

[2] This font denotes elements defined in the RSL metamodel.

In general, the metamodel defines classes, their sub- and superclasses as well as their associations. Figure 1 shows the metamodel of SVOSentences in RSL. Each SVOSentence has a Subject and a Predicate. The Subject has a NounPhrase, which links to a Determiner, a Modifier and a Noun via DeterminerLink, ModifierLink and NounLink, respectively. The Predicate links to a VerbPhrase being either a SimpleVerbPhrase for sentences with one object or a ComplexVerbPhrase for sentences containing two objects. In addition, the SimpleVerbPhrase links a Verb via a PhraseVerbLink. While the phrases enable different sentence structures, the different links contain the information about the words' inflection within a particular sentence. Finally, the links point to Terms, e.g. Nouns, Verbs etc. All Terms of one specification are contained in a Terminology which provides a software case-specific word list.

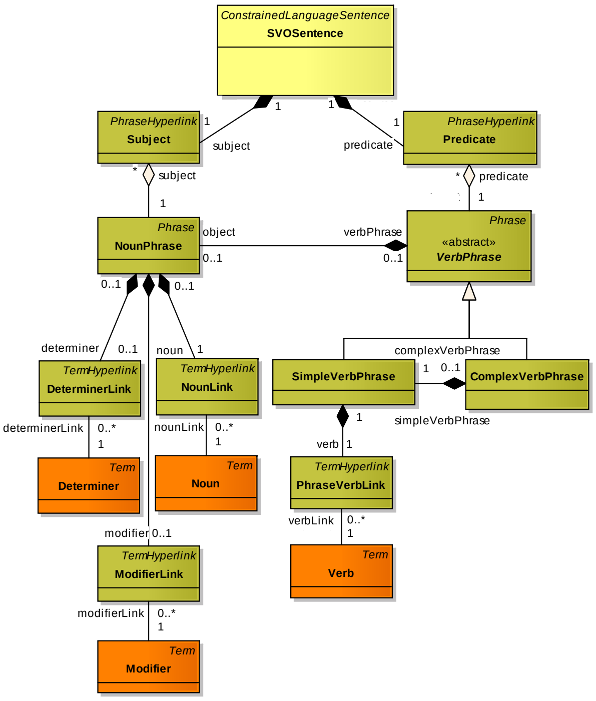


Figure 1: The structure of SVO-Sentences in RSL.

Furthermore, ConstrainedLanguageSentences like SVOSentences contain hyperlinks to a DomainSpecification, which is a software case-specific collection of Notions that acts as a glossary. While this separation of RequirementsSpecification and DomainSpecification prevents contradictory definitions scattered throughout documents, it does not provide a definition of Terms that can be processed automatically.

In RSL, the meaning of Terms is defined by linking them to WordNet[3]. *Content words* (Nouns, Verbs, Adjectives and Adverbs) have a clear lexical meaning in contrast to *Function words* (Determiners, Prepositions, Conjunctions, Pronouns, and ModalVerbs) which have little semantic content on their own and mainly define grammatical relationships. WordNet is a semantic lexicon that was developed at the Cognitive Science Laboratory at Princeton University (Fellbaum, 1998). It groups synonymic words (*synonyms*) of the English language in synonym sets (called *synsets*). It is based on the concept of synonym sets (called *synsets*) that group synonymic words (synonyms) of the English language. Amongst others, the following semantic relations connect synsets: *hypernyms* / *hyponyms* (is-a, is-a invers) and *holonym* / *meronym* (part-of, part-of invers). For each synset, WordNet provides definitions or example sentences. Words with different meanings participate in distinct synsets.

Figure 2 illustrates the general structure of requirements documents defined with RSL and how ambiguity problems of natural language are solved. The RequirementsSpecifications each have their individual Terminology. Terminology#1, for example, contains the term *Customer* while Terminology#2 and #3 both contain the term *Client*. However, *Customer* of Terminology#1 and *Client* of Terminology#2 both link to the same synset, i.e. they are synonyms while the *Client* of Terminology#3 links to another synset, i.e. the two terms *Client* have a different meaning, they are homonyms.

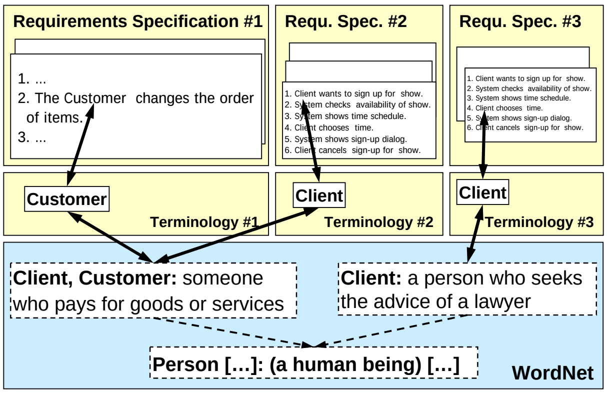

Requirements Specification #1
1. ...
2. The Customer changes the order of items.
3. ...
Requ. Spec. #2
1. Client wants to sign up for show.
2. System checks availability of show.
3. System shows time schedule.
4. Client chooses time.
5. System shows sign-up dialog.
6. Client cancels sign-up for show.
Requ. Spec. #3
1. Client wants to sign up for show.
2. System checks availability of show.
3. System shows time schedule.
4. Client chooses time.
5. System shows sign-up dialog.
6. Client cancels sign-up for show.
Customer
Terminology #1
Client
Terminology #2
Client
Terminology #3
Client, Customer: someone who pays for goods or services
Client: a person who seeks the advice of a lawyer
Person […]: (a human being) […]
WordNet

Figure 2: Relations between Requirements Specification, Terminology and WordNet.

3 ONTOLOGY CONSTRUCTION

In order to use the classification facilities as they are described in the next section, we had to represent the requirements specifications and their links to

[3] see: `wordnet.princeton.edu/`

WordNet as a DL-model, using ontology languages like OWL (OWL, 2004) or KRSS (Patel-Schneider, 1993).[4]. Such languages provide facilities to define *concepts* and *roles* (or sometimes called *properties*) between two concepts. A concept (given with a unique name) indicates the set of individuals that belongs to it, and a role indicates a relationship between concepts. Concepts and roles are combined via constructors. Typical constructors are those of $\mathcal{ALCHIF}$, i.e. Attributive Language with universal restrictions, existential qualification, concept intersection ($\mathcal{ALC}$), role hierarchy ($\mathcal{H}$), inverse roles ($\mathcal{I}$), and functional properties ($\mathcal{F}$). As we will see in the next section, we use $\mathcal{ALC}$ for gaining a tractable description logic.

With these constructors, we can define a specialization relation between concepts, which provides *superconcepts* and *subconcepts* in a taxonomy. Furthermore, $\mathcal{AL}$ allows to distinguish between *primitive* (i.e. necessary) and *defined* (i.e. necessary and sufficient) concepts, indicated with `implies` and `equivalent`, respectively.

The question is now, how to use these ontology representation facilities in order to represent requirements specifications such that similar specifications have short taxonomical distances after classification. For this task, we examine particular aspects of requirements specifications in the following and describe how they are represented.

Mapping the RSL Metamodel The RSL metamodel consists of 127 classes, sub- and superclasses and several associations between them. For example, the class SVOSentence has a generalization relation to ConstrainedLanguageSentence and the association subject to class Subject and predicate to class Predicate (see Figure 1). The mapping is straightforward: classes of the metamodel are represented with concepts, generalization relations are represented with specialization relations, and associations are represented with roles. Furthermore, the associations of a class in the metamodel *define* the class, i.e. if an object with such relations exists, then it belongs to that class and if an object belongs to a class it has the relations of that class (see Figure 3, upper part).

Although navigation across associations is bidirectional in the RSL metamodel, we do not use inverse roles in the DL-model since the similarity computation only requires roles *directing down* from the concept representing the requirements specification to the synsets of WordNet. Likewise, we can avoid a role hierarchy because only a flat hierarchy of associations is given in the metamodel. Finally, functional properties are not needed for defining concepts because they relate a class to a primitive type not to another concept. Thus, we can use the description logic language $\mathcal{ALC}$ instead of the more complex language $\mathcal{ALCHIF}$.

The mapping described above provides concepts and roles, which have direct correspondence to the classes and relations of the RSL metamodel. The concepts representing the classes of the RSL metamodel form the *upper model* of our ontology. The concepts representing the elements of specific requirements specifications are modeled as specializations of these concepts, thus, they form the lower part of our ontology.

Mapping Requirements Specifications Requirements specifications of software cases are represented through instances of classes of the RSL metamodel. For example, each SVO sentence defined in a software case is an instance of the class SVOSentence of the metamodel and is related to instances of the classes Subject and Predicate. We map these instances to further *concepts* in the ontology. Please note that we map the elements of requirements specifications to concepts and not to individuals. This is necessary because taxonomical relationships cannot be computed between individuals. However, in our approach it is a key to compute taxonomical relationships between the elements of requirements specifications. Thus, there are several SVO sentences, several related subjects and predicate etc. in each requirements specification. Each of these instances is mapped to further *concepts* in the ontology.[5] For this reason, we represent the elements of a requirements specification as subconcepts of the corresponding concept of the upper model. Thus, each SVO sentence of a specific software case is mapped to a concept with a unique id as name and the concept `SVOSentence`[6] as superconcept. The concepts representing the elements of requirements specifications are related through roles defined in the upper model. Furthermore, the roles specified in the upper model *define* the software case concepts (see Figure 3, middle part). Figure 4 shows a part of a DL-model, which we created manually for illustration purposes. SVO sentences are coded by a software case id starting with C and a readable string reflecting the text of the sentence (e.g. `SVOSentenceC2ClientOpensWindow`).

This general mapping can be used to convert all elements of a requirements specification to a DL-model. However, this is not necessary since RSL specifications contain elements that are not relevant for our semantic-based similarity measure such as NaturalLanguageHypertextSentences, which only contain meaningless strings. A further example are all TermHyperlinks (e.g. NounLink). They are used to specify the inflection of a word within a particular sentence, which is of minor relevance for the similarity of sentences. The same holds for function words like Determiner. They constitute a significant number of

[4] For presentation, we prefer the human-readable form of KRSS instead of XML-based OWL notations. However, a direct mapping exists and is used in the experiments (Section 6).

[5] Note, that not individuals are used for representing requirements specifications, but concepts, because later on taxonomical relationships are computed between requirements specifications. These cannot be computed between individuals.

[6] This font denotes elements defined in the DL-model.

sentence element's but have little semantic content on their own. Mapping each determiner into a concept within the DL-model would require the DL-reasoner to identify the equivalence of most of the determiners. This, however, would cost significant amount of computing time and would not improve the similarity measure. For this reason, we only map those RSL elements that are relevant for our similarity calculation. TermHyperlinks e.g. can easily be skipped by relating Phrases directly with corresponding Terms (see Figure 1).

Mapping WordNet Elements The WordNet metamodel consists of several classes like Synonym, Synset, Wordform etc. Synsets are related via the relation hyponym and hypernym in a synset taxonomy. For the classification, only this taxonomical relation and the fact that synsets are distinct are needed. Thus, we only map synsets from WordNet. They are represented with concepts and the synset taxonomy is represented with the subconcept relation. The role `termLinksToWordNetEntry` relates each term of a requirements specification to one synset (see Figure 3). However, it is not necessary to map all synsets of WordNet (i.e. almost 117.000) into the ontology. For our purposes, it is only necessary to map the synsets that have relations to a term of a requirements specification and all the predecessors of these synsets in the synset taxonomy.

4 CLASSIFYING THE ONTOLOGY

Through the mapping described in Section 3, requirements specifications of software cases are represented with concepts and roles in a DL-model. Since the requirements specifications contain no information that directly relates the diverse concepts of distinct specifications, all concepts are direct subconcepts of upper model concepts (see Figure 4). Figure **??** shows a manually constructed DL-model, which was developed for illustration purposes. Figure 5 shows the same DL-model as Figure 4 but after classifying. In Figure 4, the diverse subjects (`SubjectC1Client`, `SubjectC2Administrator` etc.) are direct subclasses of the upper model concept `Subject`. In this DL-model, only the synset taxonomy provides a hierarchical structure. For example `AdministratorSynset` and `ClientSynset` are both of type `PersonSynset` (see Figure 4). The synset taxonomy provided by WordNet and the fact that the concepts representing a requirements specification are defined concepts and thus, strongly related between each other allow for the classification described in the following. In addition to this synset taxonomy provided by WordNet, the concepts representing one requirements specification are defined concepts and thus, strongly related between each other (see Figure 3).

This modeling enables a DL-reasoner to use the synset taxonomy in order to classify the concepts of the requirements specifications, i.e. to compute, which concepts are equivalent (see Figure 5, upper part in red) or, which concepts can be taxonomically structured (see Figure 5, lower part in blue). In Figure 5, relations of the synset taxonomy are propagated one by one to the subjects of requirements specifications of $C1$, $C2$, and $C7$. Namely, *Subject-*$C1$*Client*, *Subject*$C2$*Administrator* and *Subject*$C7$*-Fireman* are subconcepts of the concept *Subject-*$C8$*Person* due to the fact that *ClientSynset*, *AdministratorSynset* and *FiremanSynset* are subconcepts of *PersonSynset*. Thus, after classifying the ontology, the taxonomy defined in WordNet is also reflected in the concepts representing the specific requirements specifications. Even more interesting is the impact on structural concepts like `SVOSentence`. In Figure 5 for example, the SVO sentences of case $C1$ and $C2$ are classified as subconcept of the SVO sentence of case $C8$. This is due to the fact, that in both sentences a specific type of person (being either an administrator or a client) opens a window of an operating system while the fireman (of case $C7$) opens the window of a building. Please note that Figure 5 shows only parts of this information. For example, in Figure **??** the SVO sentences of case $C1$ and $C2$ are classified as subconcept of the SVO sentence of case $C8$ because it is specified that the administrator and client open the same window (i.e. the window of an operating system), while the fireman opens a window of a building. This distinction between the different meanings of *window* is given through the links to distinct synsets (*window: a framework of wood or metal that contains a glass windowpane [...]* or *window: ((computer science) a rectangular part of a computer screen [...])*.

This kind of classification takes the structure, i.e. the roles between the concepts, into account. Through this classification, a different taxonomical distance between the concepts of requirements specifications is introduced and will lead to different similarities. By using the WordNet link and the provided mapping our similarity measure considers the meaning of the requirements specifications.

5 SIMILARITY COMPUTATION

The goal of similarity computation is to get a value between 0 and 1 that indicates an similarity between a query and a software cases; 0 denoting no similarity one denoting that the query is completely contained in the software case. Please note, that this measure is asymmetric: if a query element is not found in a software case this results in a lower similarity value. In contrast, if a software case contains elements that are not part of the query, this has no negative impact on the similarity value. This is due to the fact that additional elements in the software case are potential for reuse and thus wanted.

The assumption made here is that the taxonomical distance between two concepts can be the basis of such a similarity value. By using the classification facilities of a DL-reasoner, implicit taxonomical relations are made explicit.

Our similarity measure compares pairs of concepts in the classified taxonomy. This comparison includes both the concept placement in the taxonomical hierarchy (*distance-based similarity*) and the roles and role fillers that the concepts define (*role-based similarity*) (see (González-Calero et al., 1999) for a similar approach).

The semantic interpretation of *distance-based similarity* is: the shorter the path between the two compared concepts, the more similar are they. This *intensional* approach relies on the assumption that links in the taxonomy represent uniform distances. Another possibility to compute the similarity of two concepts is the *extensional* approach: comparing the number of their common instances. The semantic interpretation of the number of common instances is: the more common instances both concepts share, the more similar they are. This approach relies on the assumption that all concepts in the taxonomy are evenly populated with instances. In our approach, we consider the *intensional* approach because each concept represents exactly one individual, i.e. one element of a requirements specification.

For computing the distance-based similarity between two concepts, the distance of both concepts to the least common subsumer (LCS) is computed, both values are added and a value describing that distance-based similarity is returned: 1 means both concepts are in fact the same concept, and the smaller the value, the further away the concepts are from each other.

```
ALGORITHM: distanceBasedSimilarity(c1, c2)
1. lcs = leastCommonSubsumer(c1, c2)
2. distance1 = pathLength(c1, lcs)
3. distance2 = pathLength(c2, lcs)
4. distance = distance1 + distance2
5. return $\frac{1}{\alpha+\text{distance}}$
```

Note that the α value is needed because comparing a concept to itself the taxonomic distance is 0 and division by 0 is not allowed.

The basic idea of *role-based similarity* is that the similarity of two concepts depends on the similarity of their subgraphs. When comparing roles, their most important aspects are their fillers: concepts or concrete domains that specify the value of a role. When comparing two concepts, the function of role-based similarity is recursively applied. The recursion terminates when two concepts without roles are compared; their similarity is given by the distance-based similarity function. Cyclic roles are taken into account during implementation.

The roles of both concepts are recursively compared and similarities are summed up for every concept. A value describing role-based similarity is returned: 1 means both concepts have the same roles, and the smaller the value, the less their roles have in common.

The role-based similarity is based on values that are computed by comparing every role of one concept to every role of the other concept. A matrix is filled with all similarity values and the maximum value in every row or column indicates the best matching roles for both concepts. In order to find a match for every role and not use the same role as a match more than once, the maximum value of a row is used. When $C1$ has less roles the maximum value of a column is used and when $C2$ has less roles the maximum value of a row is used.

```
ALGORITHM: roleBasedSimilarity(c1, c2)
1. if c1 OR c2 has roles, do
  (a) similarity = 0
  (b) roleValues = [ —roles(c1)— ], [ —roles(c2)— ]
  (c) for int i=0; i ¡ —roles(c1)—; i++, do
    i. s = roles(c1)[i]
    ii. for int j=0; j ¡ —roles(c2)—; j++, do
      A. t = roles(c2)[j]
      B. roleValues[i][j] = conceptBasedSimilarity(s, t)
  (d) if —roles(c1)— ¡ —roles(c2)—
    i. for int k=0; k ¡= —roles(c1)—; k++, do
      A. similarity += max(roleValues[k][*])
  (e) else
    i. for int k=0; k ¡= —roles(c2)—; k++, do
      A. similarity += max(roleValues[*][k])
  (f) similarity = $\frac{\text{similarity}}{\max_{(i,j)}}$
2. else, do
  (a) similarity = distanceBasedSimilarity(c1, c2)
3. return similarity
```

Both aspects can be computed independently from each other. Therefore, we defined two algorithms, one computing the distance-based similarity between two concepts and another one comparing the common roles of concepts. The similarity between two concept definitions is the sum of the distance-based similarity and the role-based similarity divided by two.

6 EXPERIMENTS

In the previous sections, we used a small running example for demonstrating our approach to ontology-based requirements comparison. In the following, we give an overview of the experiments we are currently running for applying the approach to larger examples and to industrial software cases. First, we sketch the technical setting.

Technical Setting The metamodel of RSL is created with a standard UML tool (Enterprise Architect) and an appropriate profile containing the metamodel. For creating requirements specifications according to the RSL metamodel, a specific software development tool is under development. This tool was used by some industrial partners and us for creating formalized requirement specifications. Internally, the tool uses a graph representation (see (Dahm and Widmann, 2003) for details), which is also based on the RSL metamodel. This way, a defined representation for requirements specifications is available for our experiments.

A JAVA-based converter maps the graph representation of requirements specifications to an OWL knowledge base by using the mapping described in this paper. For debugging the OWL knowledge base, Protégé is used. Protégé also provides interfaces to the DL-reasoners Pellet and Racer. For debugging inference behavior, Pellint is applied.[7]

The similarity computation is a further JAVA-component developed by us. It traverses the classified knowledge base for computing similarities as presented in Section 5. Together with the converter, this component will be integrated in the above mentioned software development tool together with other similarity measures based on text and graph structures. The tool is intended to provide sophisticated reuse mechanisms in combination with software transformations (Śmiałek, 2006).

Industrial Experiments We first created examples to test the similarity measure by slightly varying requirements specifications, and manually judge the similarity of software cases. These experiments show that the mapping described in Section 3 provides coherent DL-models. Furthermore, we could show that the taxonomical similarity measure computes the same similarity ranking for these test cases as considered plausible by us beforehand. For applying the approach in a broader scale, 16 industrial software cases have been created by international software organizations using the mentioned tool. The application domains are in the area of internet banking, investment funds management, emergency systems, forestry systems, financial contract systems, and funding systems. They include 261 requirements written in RSL in total, which map to more than 30.000 defined concepts. Those concepts are reduced to relevant concepts for similarity computations as described in Section 3 (see Table 1). Current experiments still use subsets of the industrial software cases for classification due to the complexity of the resulting ontologies. However, the general approach could be verified and further work will extend the number of tractable concepts.

Table 1: Sizes of industrial software cases

Upper model defined concepts	34
Software cases	16
Requirements	261
Defined concepts (complete mapping)	30184
Relevant defined concepts	8325
Relevant roles	36

7 DISCUSSION

In this paper, a new approach in the direction of reusing software on the basis of its semantical description is presented. For this task, a link between a formal requirements specification and an ontology is established. This link and the mapping of the formal requirements specifications to Description Logic concepts enable inferences that classify existing requirements specifications. This classification can be done offline before the reuse activities start. For reusing parts of former software cases, similarity computations on the basis of classified concepts are applied.

When using ontologies in industrial settings the ontology construction should be considered as an automatic process unless a knowledge engineer is continuously available within the organization. Automatic ontology construction can be achieved through learning ontologies (which is not considered in this paper), or by constructing ontologies programmatically from existing data and knowledge sources. In this paper, we examine the formal notation of requirements specifications as such a data source and developed an automatic mapping from RSL requirements specifications to an ontology (here called DL-model). By doing so, the modeling effort typically needed when ontology-based approaches are applied is reduced to linking words to WordNet elements in order to specify the intended meaning. This can be done by the requirements engineers who specify the requirements, i.e. no knowledge engineer is needed. Structural relations are implicit defined by using the tool, which automatically creates requirements specifications relying on the given metamodel.

The use of a given ontology like WordNet supplies an easy-to-use starting point. Although WordNet contains more than 155.000 words and about 117.000 synsets leading to almost 207.000 word-sense pairs, it does not contain all needed terms for any domain at hand. This holds especially for highly domain-specific terms which typically can be found in specifications. The requirements engineers define these terms as WordNet extensions, i.e. they specify a taxonomical relation. Linking words to WordNet elements and specifing extensions if necessary is the only knowledge modeling effort needed in our approach. This however, is essential for our approach since it provides the basis for the comparison of requirements specifications created for different domains, by different persons or even in different organizations. Without such a link, an alignment or comparison of different software cases would be difficult, one could only apply natural language processing in order to guess the intended meanings.

The approach of retrieving cases on the basis of a similarity measure originates in case-based reasoning (CBR) (Aamodt and Plaza, 1994).

[7] `protege.stanford.edu`, `clarkparsia.com/pellet`, `www.racer-systems.com`, `pellet.owldl.com/pellint`

Bergmann has described how to set up taxonomy similarities for various relationship types (Bergmann, 1998). In (González-Calero et al., 1999) and (Gomes et al., 2004) further combination of CBR and Description Logic are presented. In the paper at hand, we apply such taxonomical similarity measures to a taxonomy, which has been constructed by first mapping metamodelling-based requirements specifications to a DL-Model and then classifying this DL-model by a Description Logic reasoner.

As a further aspect of CBR, Memory Organization Packages (MOPs) [8] (Schank, 1982) are used for organizing cases into significant portions for allowing partial matching of cases. Also specializations are used for structuring those parts. In this respect, our approach is similar to MOPs, however, the structure we use is based on formal concepts of a description logic, which enable classification services provided by DL-reasoners, whereas MOPs use a frame-based strict hierarchy.

Through the mapping given in this paper, highly structured defined concepts are automatically created from RSL requirements specifications. Description Logic inferences provide the classification of these highly structured concepts. By this means, the given taxonomy of WordNet's synset is used for classifying all structured elements of requirements specifications (e.g. SVOSentences) and implicitly existing taxonomical relations even between different software cases are made explicit. We can not somehow relax the relatively strong modeling with defined concepts, because moving from defined to primitive concepts would not allow this classification of structured concepts.

(Borgida et al., 2005) describes similarity measures for Description Logics, where diverse types of concept definitions like nested roles are taken into account. But composite concepts consisting of multiple roles (like SVOSentences) are not yet considered. However, examining concept definitions in such detail could be used for improving our similarity measure.

The current mapping of the metamodel takes generalization relations and associations into account. Attributes to datatypes like strings or numbers are not considered. Including those would need Description Logics that handle concrete domains, and would need a similarity measure for comparing strings (e.g. based on information retrieval). Since the tool under development already provides such mechanisms, which will be combined with our ontology-based similarity measure, we will not go into that direction.

Compared to other specification technologies as e.g. provided by the specification language Z (Woodcock and Davies, 1996), our approach allows specifications on the much higher requirements level, instead of specifiying computer programs itself. However, our requirements specification language RSL is also used for creating architecture models and detailed design models through transformations (Kalnins et al., 2005). Furthermore, the link of the specification to ontologies enables the support through reasoners, which is not given in other specification languages.

Finally, our approach extends simple WordNet-based similarity measures as described e.g. in (Pedersen et al., 2004). These measures provide similarity values for synset pairs only and cannot compare structured elements like SVO sentences or whole scenarios. However, this is essential in our approach.

Our approach relies on the implicit existence of taxonomical relations between software cases modeled in the ontology. For example, in Figure **??** the software case containing the concept `Person` is needed for classifying software cases below it. Instead of relying on the existence of such software cases, it is possible to create such superconcepts automatically by a DL-reasoner. In an easy case, such created superconcepts would be disjunctions of the subconcepts. Our future work will include the consideration of such approaches for not being dependent on the existence of general concepts.

Most of these measures are based on pathlengths between synsets defined by semantic relations. Two synsets are considered similar when there is a short distance between them. Because the meaning of words is given the paraphrase and ambiguity problem do not emerge – both problems can be handled with this approach.

WordNet has been applied in combination with different approaches in order to improve retrieval results. For an overview, we refer the interested reader to (Pedersen et al., 2004). A matter of current research in Information Retrieval is the application of WordNet, or more generally semantic lexica, in order to solve the ambiguity and paraphrase problem (Varelas et al., 2005; Liu et al., 2005). (Gomes et al., 2004) used WordNet in combination with CBR approaches for retrieval of UML class diagrams.

Also, future work will show if the currently

[8] copweb.ecn.purdue.edu/engelb/abe565/cbr.html

used role-based similarity is really needed when a classification is applied in advance. The classification explicitly uses the role definitions for establishing new taxonomical relations. Thus, the impact of roles is mapped to the taxonomy, which might enable us to only use taxonomical similarity and discard role-based similarity.

8 SUMMARY

In this paper, a new combination of formal requirements specifications and description-logic based inferences is used for facilitating software retrieval. A formal mapping of requirements specifications represented by means of a metamodel to a highly differentiated DL-model is given. This model enables a reasoner to classify former requirements specifications. A taxonomy-based similarity computation uses the classified taxonomy as basis for comparing requirements specifications. Our experience so far showed that the approach works. We were able to map several industrial software cases into one ontology. Further application to industrial environments is currently under development. The principal functioning of such an approach could be shown. Also the mapping of several industrial cases into one ontology could be achieved. However, further applications to industrial environments is currently under development.

ACKNOWLEDGEMENTS

This work is partially funded by the EU: Requirements-driven Software Development System (ReDSeeDS) (contract no. IST-2006-33596). The project is coordinated by Infovide, Poland with technical lead of Wasaw University of Technology and with University of Koblenz-Landau, Vienna University of Technology, Fraunhofer IESE, University of Latvia, HITeC e.V. c/o University of Hamburg, Heriot-Watt University, PRO DV, Cybersoft and Algoritmu Sistemos.

REFERENCES

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. (2003). *The Description Logic Handbook*. Cambridge University Press.

Bergmann, R. (1998). On the use of taxonomies for representing case features and local similarity measures. In *6th German Workshop on CBR*.

Bergmann, R., Breen, S., Manago, M., Wess, S., Göker, M., Althoff, K.-D., and Traphöner, R. (1999). *Developing Industrial Case-Based Reasoning Applications - The INRECA Methodology*, volume 1612 of *Lecture Notes in Artificial Intelligence*. Springer.

Borgida, A., Walsh, Thomas, J., and Hirsh, H. (2005). Towards measuring similarity in description logics. In *International Workshop on Description Logics*, Edinburgh, Scotland. DL2005.

Dahm, P. and Widmann, F. (2003). GraLab - Das Graphenlabor. Projektbericht 4.3.0, University of Koblenz-Landau, Institute for Software Technology.

Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database*. MIT Press.

Fuchs, N. E., Höfler, S., Kaljurand, K., Rinaldi, F., and Schneider, G. (2005). Attempto controlled english: A knowledge representation language readable by humans and machines. *Lecture Notes in Computer Science*, 3564:213–250.

Gomes, P., Pereira, F. C., Paiva, P., Seco, N., Carreiro, P., Ferreira, J. L., and Bento, C. (2004). Using wordnet for case-based retrieval of uml models. *AI Commun.*, 17:13–23.

González-Calero, P. A., Gómez-Albarran, M., and Díaz-Agudo, B. (1999). Applying DLs for retrieval in case-based reasoning. In *Proc. of the 1999 International Workshop on Description Logics (DL'99)*.

Kaindl, H., Smialek, M., Mukasa, K. S., Wolter, K., Bildhauer, D., and Pooley, R. J. (2008). A comprehensible requirements specification language based on a metamodel. submitted to: 16th IEEE Requirements Engineering Conference (RE'08).

Kaindl, H., Śmiałek, M., Svetinovic, D., Ambroziewicz, A., Bojarski, J., Nowakowski, W., Straszak, T., Schwarz, H., Bildhauer, D., Brogan, J. P., Mukasa, K. S., Wolter, K., and Krebs, T. (2007). Requirements specification language definition. Project Deliverable D2.4.1, ReDSeeDS Project. www.redseeds.eu.

Kalnins, A., Celms, E., and Sostaks, A. (2005). Model transformation approach based on MOLA. In *Workshop on Model Transformations in Practice (MTIP), MoDELS/UML '2005*.

Liu, S., Yu, C., and Meng, W. (2005). Word sense disambiguation in queries. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 525–532, New York, NY, USA. ACM Press.

OMG (2006). *Meta Object Facility Core Specification, version 2.0, formal/2006-01-01*. Object Management Group.

OMG (2007). *Unified Modeling Language: Superstructure, version 2.1.1 (non-change bar), formal/07-02-05*. Object Management Group.

OWL (2004). *OWL Web Ontology Language Overview.* W3C. http://www.w3.org/TR/owl-features.

Patel-Schneider, P. F. (1993). Description-logic knowledge representation system specification from the KRSS group of the ARPA knowledge sharing effort. Technical report, DARPA Knowledge Representation System Specification (KRSS) Group of the Knowledge Sharing Initiative.

Pedersen, T., Patwardhan, S., and Michelizzi, J. (2004). WordNet::similarity - measuring the relatedness of concepts. In *Proc. 19th National Conference on Artificial Intelligence (AAAI-04)*, page 3p.

Salotti, S. and Ventos, V. (1998). Study and formalization of a case-based reasoning system using a description logic. In *EWCBR*, pages 286–297.

Schank, R. C. (1982). *Dynamic Memory: A Theory of Learning in Computers and People.* Cambridge University Press.

Śmiałek, M. (2006). Mechanisms for requirements based model reuse. In *International Workshop on Model Reuse Strategies - MoRSe*, pages 17–20.

Śmiałek, M., Ambroziewicz, A., Bojarski, J., Nowakowski, W., and Straszak, T. (2007). Introducing a unified requirements specification language. In Madeyski, L., Ochodek, M., Weiss, D., and Zendulka, J., editors, *Proc. CEE-SET'2007, Software Engineering in Progress*, pages 172–183. Nakom.

Śmiałek, M., Bojarski, J., Nowakowski, W., and Straszak, T. (2005). Writing coherent user stories with tool support. *Lecture Notes in Computer Science*, 3556:247–250.

Varelas, G., Voutsakis, E., Raftopoulou, P., Petrakis, E. G. M., and Milios, E. E. (2005). Semantic similarity methods in WordNet and their application to information retrieval on the web. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 10–16, New York, NY, USA. ACM Press.

Wolter, K., Krebs, T., and Hotz, L. (2008). Combined similarity measure for determining similarity of model-based and descriptive requirements. In *Artificial Intelligence Techniques in Software Engineering (ECAI 2008 Workshop)*, pages 11–15.

Woodcock, J. and Davies, J. (1996). *Using Z: Specification, Refinement, and Proof.* Prentice Hall International.

Definition of an upper-model concept:

```
(equivalent SVOSentence
   (and ConstrainedLanguageSentence
         (some subject Subject)
         (some predicate Predicate)))
```

Parts of the definition of the sentence:
"Client Opens a PC Window"

```
(equivalent SVOSentenceC2ClientOpensWindow
   (and SVOSentence
         (some subject SubjectC2Client)
         (some predicate PredicateC2OpenWindow)))
(equivalent PredicateC2OpensWindow
   (and Predicate
      (some verbPhrase SimpleVerbPhraseC2Open)))
(equivalent SimpleVerbPhraseC2Open
   (and SimpleVerbPhrase
         (some verb PhraseVerbLinkC2Open)
         (some object NounPhraseC2Window)))
(equivalent PhraseVerbLinkC2Open
   (and PhraseVerbLink
         (some linkedVerb VerbC2Open)))
(equivalent VerbC2Open
   (and Verb
         (some termLinksToWordnetEntry OpenSynset)))
(equivalent NounPhraseC2Window
   (and NounPhrase
         (some noun NounLinkC2Window)))
(equivalent NounLinkC2Window
   (and NounLink
         (some linkedNoun NounC2Window)))
(equivalent NounC2Window
   (and Noun
         (some termLinksToWordnetEntry WindowPCSynset)))
```

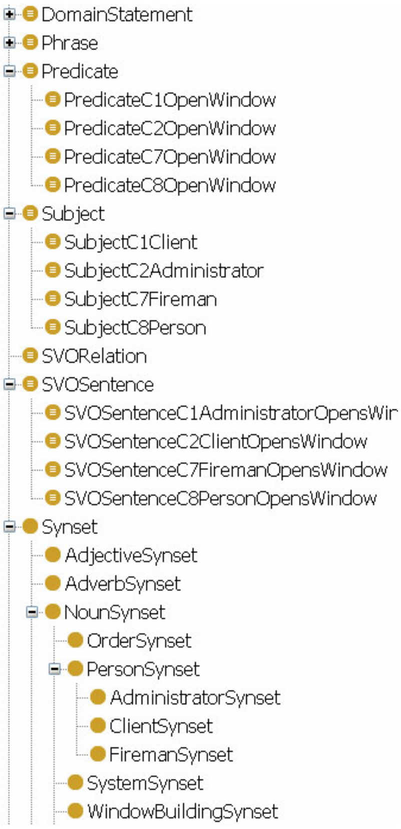
Parts of the definition of the sentence:
"Fireman opens a Building Window"

```
(equivalent SimpleVerbPhraseC7Open
   (and SimpleVerbPhrase
         (some verb PhraseVerbLinkC7Open)
         (some object NounPhraseC7Window)))
(equivalent NounPhraseC7Window
   (and NounPhrase
         (some noun NounLinkC7Window)))
(equivalent NounLinkC7Window
   (and NounLink
         (some linkedNoun NounC7Window)))
(equivalent NounC7Window
   (and Noun
         (some termLinksToWordnetEntry WindowBuildingSynset)))
```

Definition of some synset concepts:

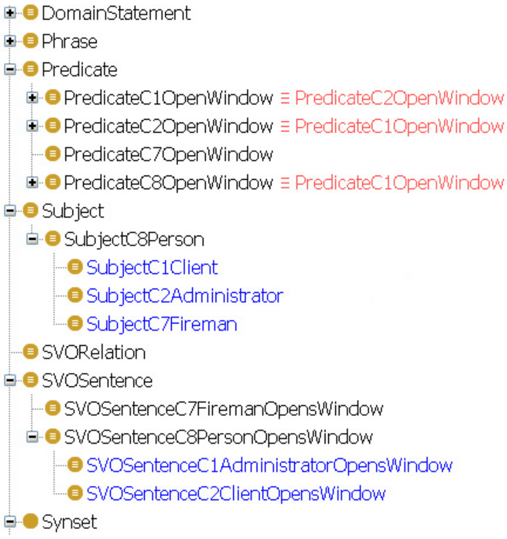
```
(implies OpenSynset VerbSynset)
(implies WindowBuildingSynset NounSynset)
(implies WindowPCSynset NounSynset)
(implies PersonSynset NounSynset)
(implies AdministratorSynset PersonSynset)
(implies ClientSynset PersonSynset)
(implies FiremanSynset PersonSynset)
```

Figure 3: Examples for mappings of the definition of the upper model concept `SVOSentence`, a specific SVO sentence of a software case `C2`, and a WordNet synset with the description logic representation language KRSS. Instead of `implies` and `equivalent` also `define-primitive-concept` and `define-concept` are used in the literature respectivley.



```
DomainStatement
Phrase
Predicate
    PredicateC1OpenWindow
    PredicateC2OpenWindow
    PredicateC7OpenWindow
    PredicateC8OpenWindow
Subject
    SubjectC1Client
    SubjectC2Administrator
    SubjectC7Fireman
    SubjectC8Person
SVORelation
SVOSentence
    SVOSentenceC1AdministratorOpensWir
    SVOSentenceC2ClientOpensWindow
    SVOSentenceC7FiremanOpensWindow
    SVOSentenceC8PersonOpensWindow
Synset
    AdjectiveSynset
    AdverbSynset
    NounSynset
        OrderSynset
        PersonSynset
            AdministratorSynset
            ClientSynset
            FiremanSynset
        SystemSynset
        WindowBuildingSynset
```

Figure 4: Asserted hierarchy



```
DomainStatement
Phrase
Predicate
    PredicateC1OpenWindow ≡ PredicateC2OpenWindow
    PredicateC2OpenWindow ≡ PredicateC1OpenWindow
    PredicateC7OpenWindow
    PredicateC8OpenWindow ≡ PredicateC1OpenWindow
Subject
    SubjectC8Person
        SubjectC1Client
        SubjectC2Administrator
        SubjectC7Fireman
SVORelation
SVOSentence
    SVOSentenceC7FiremanOpensWindow
    SVOSentenceC8PersonOpensWindow
        SVOSentenceC1AdministratorOpensWindow
        SVOSentenceC2ClientOpensWindow
Synset
```

Figure 5: Cutout of inferred hierarchy