Parsing Software Requirements with an Ontology-based Semantic Role Labeler

Michael Roth
University of Edinburgh
mroth@inf.ed.ac.uk

Ewan Klein
University of Edinburgh
ewan@inf.ed.ac.uk

Abstract

Software requirements describe functional and non-functional aspects of a software system and form the basis for the development process. Accordingly, requirements of existing systems can provide insights regarding the re-usability of already implemented software artifacts. To facilitate direct comparison between requirements of existing and to be developed systems, we propose to automatically map requirements in natural language text to structured semantic representations. For this task, we adapt techniques from semantic role labeling to a high-level ontology that defines concepts and relations for describing static software functionalities. The proposed method achieves a precision and recall of 77.9% and 74.5%, respectively, on an annotated software requirements dataset and significantly outperforms two baselines that are based on lexical and syntactic patterns.

1 Introduction

During the process of software development, developers and customers typically discuss and agree on requirements that specify the functionality of a system that is being developed.[1] Such requirements play a crucial role in the development lifecycle, as they form the basis for implementations, corresponding work plans, cost estimations and follow-up directives (van Lamsweerde, 2009). In general, software requirements can be expressed in various different ways, including the use of UML diagrams and storyboards. Most commonly, however, expectations are expressed in natural language (Mich et al., 2004), as shown in Example (1):

(1) A user should be able to login to his account.

While requirements expressed in natural language have the advantage of being intelligible to both clients and developers, they can of course also be ambiguous, vague and incomplete. Although formal languages could be used as an alternative that eliminates some of these problems, customers are rarely equipped with the mathematical and technical expertise for understanding highly formalised requirements. To benefit from the advantages of both natural language and formal representations, we propose to induce the latter automatically from text in a semantic parsing task. Given the software requirement in Example (1), for instance, we would like to construct a representation that explicitly specifies the types of the entities involved (e.g., OBJECT(account)) and relationships among them (e.g., ACTS_ON(login, account)).

Though there exist ontologies and small-scale data sets with annotated concept instances, most previous approaches to inducing such annotations from text relied on hand-crafted grammars and heuristic postprocessing rules. In this paper, we propose to identify and classify instances of ontology concepts and relations using statistical techniques from semantic role labeling.

The remainder of this paper is structured as follows. In Section 2, we describe previous approaches on converting software requirements to formal representations. In Section 3, we provide a summary

[1] Although software engineering can also involve *non-functional* requirements, which describe general quality criteria of a system, this paper is only concerned with functional requirements, i.e., requirements that specify the behavior of a system.

of a previously developed ontology and annotated data set which we use for training and testing our own approach. In Section 4, we decribe the semantic role labeling architecture that we developed to automatically process software requirements. In Section 5, we evaluate our proposed approach and compare it to two pattern-based baselines. Finally, we conclude this paper in Section 6 with a discussion and outlook on future work.

2 Related Work

A range of methods have been proposed in previous work to (semi-)automatically process requirements written in plain, natural language text and map them to formal representations. To the best of our knowledge, Abbott (1983) was the first to introduce a technique for extracting data types, variables and operators from informal texts describing a problem. The proposed method follows a simple rule-based setup, in which common nouns are identified as data types, proper nouns as objects and verbs as operators between them. Booch (1986) described a method of similar complexity that extends Abbot's approach to object-oriented development. Saeki et al. (1989) implemented a first prototype that automatically constructs object-oriented models from informal requirements. As proposed by Abbott and Booch, the system is based on automatically extracted nouns and verbs. Although Saeki et al. found resulting object diagrams of reasonable quality, they concluded that human intervention was still necessary to distinguish between words that are relevant for the model and irrelevant nouns and verbs. Nanduri and Rugaber (1995) proposed to further automate object-oriented analysis of requirement texts by applying a syntactic parser and a set of post-processing rules. In a similar setting, Mich (1996) employed a full NLP pipeline that contains a semantic analysis module, thus omitting the need for additional post-processing rules. More recent approaches include those by Harmain and Gaizauskas (2003) and Kof (2004), who relied on a combination of NLP components and human interaction. Whereas most previous approaches aim to derive class diagrams, Ghosh et al. (2014) proposed a pipeline architecture that induces logical expressions from syntactic parses via a set of heuristic post-processing rules.

Despite this seemingly long tradition, methods for processing software requirements have tended to depend on domain-specific heuristics and knowledge bases or have required additional user intervention. In contrast, we propose to utilize annotated data to learn how to perform semantic parsing of requirements automatically.

3 Ontology and Dataset

As training and testing material for our semantic role labeling approach, we use a high-level ontology of static software functionalities and an existing data set of software requirements with annotated ontology instances (Roth et al., 2014). The ontology by Roth et al. covers general concepts for describing static software functionalities. The main concepts and their associated relations are as follows:

`Action` An `Action` describes an operation that is performed by an `Actor` on any number of `Object`(s).[2] The participants of an `Action` are indicated by the relations HAS_ACTOR and ACTS_ON, respectively.

`Actor` (HAS_ACTOR) A `Actor` is an active participant of an `Action` and can be the user of a system or a software system itself.

`Object` (ACTS_ON) A `Object` is any kind of entity involved in an `Action` other than the `Actor`.

`Property` (HAS_PROPERTY) A `Property` is an attribute of an `Object`, a characteristic of an `Action` or an optional specification of an `Actor`. The domain of the relation HAS_PROPERTY will be the set of entities which possess a given `Property`.

[2]Note that an `Action` is a kind of entity, so the approach is similar to a Davidsonian event-based semantics.

Concept	Instances	Relations	Instances
Action	435		
Actor	305	HAS_ACTOR	355
Object	613	ACTS_ON	531
Property	698	HAS_PROPERTY	690
Total	2,051	Total	1,576

Table 1: Counts of annotated instances of concepts and relations in the dataset from Roth et al. (2014)

The annotated data is based on a collection of functional software requirements from software engineering classes at universities and industrial prototypes of a software company (for details, cf. Roth et al., 2014).[3] The collection contains 325 requirement sentences, over 2,000 annotated instances of ontology concepts and more than 1,500 instances of relations between concepts. All annotations refer to concepts and relations described in the previous paragraphs. Table 1 provides counts of annotations per concept and relation type. Note that instances of Actor can be involved in multiple instances of Action and some instances of Object are not involved in any, hence the number of relations can differ from the number of associated concepts. Since all annotations are provided as mark-up for tokens in text, we can directly use the annotated data set for our role labeling approach, which we describe in the next section.

4 Ontology-based Semantic Role Labeling

The goal of this work is to automatically identify and extract instances of ontology concepts and relations from software requirements expressed in natural language text. Based on the ontology and dataset described in Section 3, we developed a parser that learns statistical models for this task and can be applied to new data. In practice, the parsing task involves several steps: first, instances of concepts need to be identified and then mapped to the correct class and second, relations between instances of concepts need to be identified and labeled accordingly.

Inspired by previous work on semantic role labeling, we use a parsing pipeline consisting of a syntactic dependency parser and several semantic analyses modules. We specifically chose this kind of architecture as our task closely resembles previous semantic role labeling (SRL) tasks, in which such systems achieve state-of-the-art performance (Hajič et al., 2009). To meet the constraints and characteristics of the software requirements domain, we have taken as our starting point techniques for labeling roles of the kind standardly used in domain-independent semantic analyses and extended them to the concepts and relations defined in the ontology (cf. Section 3).

The following subsections describe our implementation in more detail. In Section 4.1, we introduce the preprocessing pipeline that we apply to compute a syntactic analysis for each requirement expressed as a sentence in English. Section 4.2 describes the semantic analysis modules that we implemented to map words and constituents in a sentence to instances of concepts and relations from the ontology. Finally, we define the features and learning techniques applied to train each statistical model in subsections 4.3. We illustrate each step of our analysis using the following running examples:

(a) "The user must be able to upload photos."

(b) "Any user must be able to search by tag the public bookmarks of all RESTMARKS users."

4.1 Syntactic Analysis

The syntactic analysis stage of our pipeline architecture consists of the following steps: tokenization, part-of-speech tagging, lemmatization and dependency parsing. Given an input sentence, this means that

[3] We thank the authors for making their data available to us.

the pipeline separates the sentence into word tokens, identifies the grammatical category of each word (e.g., "user" → noun, "upload" → verb) and determines their uninflected base forms (e.g., "users" → "user"). Finally, the pipeline identifies syntactic dependents of each word and their respective grammatical relation (e.g., ⟨"user","must"⟩ → subject-of, ⟨"upload","photos"⟩ → object-of).

For all syntactic analysis steps, we rely on components and pre-trained models from a system called Mate tools (Björkelund et al., 2010; Bohnet, 2010), which is freely available online.[4] This choice is based on two criteria. First, the system achieves state-of-the-art performance on a benchmark data set for syntactic analysis (Hajič et al., 2009). Second, the output of the syntactic analysis steps has successfully been used as input for the related task of PropBank/NomBank-style semantic role labeling (Palmer et al., 2005; Meyers et al., 2008).

4.2 Semantic Analysis

Our actual semantic role labeling pipeline consists of four main steps to extract instances of ontology concepts and relations from requirements written in natural language text: (1) identifying instances of the concepts `Action` and `Object`, (2) assigning the respective concept type, (3) determining instances of related concepts, and (4) labeling relationships between pairs of concept instances. Our implementation is based on the semantic role labeler from Mate tools and uses the built-in re-ranker to find the best joint output of steps (3) and (4). We extend Mate tools with respect to continuous features and arbitrary label types. We describe each component of our implementation in the following paragraphs.

Step (1) The first component of our pipeline identifies words in a text that instantiate the ontology concepts `Action` and `Object`. The motivation for identifying these two concept types first is that only they govern relationships to all other ontology concepts through the three relations `ACTS_ON`, `HAS_ACTOR` and `HAS_PROPERTY`. We hence expect the corresponding linguistic units to behave similarly to PropBank/NomBank predicates and can apply similar features as used in the *predicate identification* step implemented in Mate tools. Our implementation considers each verb and each noun in a sentence and performs binary classification based on lexical semantic and syntactic properties.

Step (2) This step determines which ontology concept is applicable to each instance identified in Step (1). That is, for each verb and noun in a sentence classified as a potential instance of `Action` and `Object`, the component predicts and instantiates the actual ontology concept (e.g., "upload"→`action`, "search"→`action`). As in the previous component, lexical semantic and syntactic properties are exploited to perform classification. This step corresponds to the *predicate disambiguation* step applied in PropBank/NomBank semantic role labeling but, in contrast to the former, the available set of labels is predefined in the ontology and hence does not depend on the identified "predicate".

Step (3) The component for determining related concept instances detects words and phrases in a text that are related to the instances previously identified in Step (1). The main goal of this step is to identify the `Actor` of an `Action` and affected `Objects` as well as instances of `Property` that are related to any of the former. As such, this step is similar to *argument identification* in semantic role labeling. Accordingly, we take as input potential 'arguments' of a concept instance from Step (1) and perform binary decisions that indicate whether a word or phrase instantiates a (related) ontology concept. In example (a), both "the user" and "photos" are ontology instances that are related to the `Action` expressed by the word "upload". In example (b), instances related to "search" are: "any user", "by tag" and "the public bookmarks of all RESTMARKS users". In this example, "of all RESTMARKS users" further denotes a `Property` related to the instance of `Object` expressed by the phrase "the public bookmarks".

[4] `http://code.google.com/p/mate-tools/`

	`Action` and `Object`		Related concepts	
	identification	classification	identification	classification
Affected word forms	•	•	•	•
Affected word lemmata	•	–	–	–
Word part-of-speech	•	–	•	•
Word vector representation	•	•	•	•
Relation to parent	•	–	•	•
Parent part-of-speech	•	•	–	–
Set of dependent relations	–	•	–	–
Single child words	•	–	–	–
Single child part-of-speech	•	–	–	–
Dependencies between words	–	–	•	•
Order of affected words	–	–	•	•
Distance between words	–	–	•	–

Table 2: Linguistic properties that are used as features in statistical classification

Step (4) The component for labeling relationships determines which relations hold between a pair of ontology instances as identified in Steps (1) and (3). Generally, each instance can be involved in multiple relations and hence more than one concept type can apply to a single entity. To represent this circumstance appropriately, the component performs classification on pairs of related instances (e.g., ⟨"the user", "upload"⟩ → ⟨`Actor`, `Action`⟩, ⟨"by tag", "search"⟩ → ⟨`Property`, `Action`⟩). This step roughly corresponds to the *argument classification* step of the semantic role labeler implemented in Mate tools. As with concept labels, however, our set of potential relations is predefined in the ontology. For classification, our implementation relies on lexical semantic and syntactic properties as well as additional characteristics that hold between the linguistic expressions that refer to the considered instances (e.g., their order in text).

4.3 Features and Learning

In practice, each step in our pipeline is implemented as a logistic regression model that uses linguistic properties as features, for which appropriate features weights are learned based on annotated training data. The majority of features applied in our models are already implemented in Mate tools (Björkelund et al., 2010). Given that the number of annotations available for our task is about one order of magnitude smaller than those in PropBank/NomBank, we utilize a subset of features from previous work, as summarized in Table 2, which we greedily selected based on classification performance.

To compensate for sparse features in our setting, we define additional features based on distributional semantics. The motivation for such features lies in the fact that indicator features and feature combinations (e.g., the affected word type plus its part-of-speech) can be too specific to provide robust generalization for semantic analysis. To overcome the resulting gap in coverage, we represent each word in a classification decision by a low-rank vector representation that is computed based on word-context co-occurrence counts and can be computed over large amounts of unlabeled text. As distributional representations tends to be similar for words that are similar in meaning, this allows word type information to be utilized at test time, even if a specific word has not occurred in the training data.

As indicated in Table 2, we apply vector representations of words for identifying instances of `Action` and `Object` as well as for classifying instances of related concepts. Following a recent comparison of different word representations for semantic role labeling (Roth and Woodsend, 2014), we use a set of publicly available vectors that were learned using a neural language model (Bengio et al., 2003).[5]

[5] `http://github.com/turian/neural-language-model`

Model	Precision	Recall	F_1-score
Baseline 1 (word-level patterns)	62.8	35.2	45.1
Baseline 2 (syntax-based patterns)	78.3	62.1	69.3
Full SRL model	77.9	74.5	76.2

Table 3: Performance of our full model and two simplified baselines; all numbers in %

5 Evaluation

We evaluate the performance of the semantic role labeling approach described in Section 4, using the annotated dataset described in Section 3. As evaluation metrics, we apply labeled precision and recall. We define *labeled precision* as the fraction of predicted labels of concept and relation instances that are correct, and *labeled recall* as the fraction of annotated labels that are correctly predicted by the parser. To train and test the statistical models underlying the semantic analysis components of our pipeline, we perform evaluation in a 5-fold cross-validation setting. That is, given the 325 sentences from the annotated data set, we randomly create five folds of equal size (65 sentences) and use each fold once for testing while training on the remaining other folds.

As baselines, we apply two pattern-based models that are similar in spirit to earlier approaches to parsing software requirements (cf. Section 2). The first baseline simply uses word level patterns to identify instances of ontology concepts and relations. The second baseline is similar to the first but also takes into account syntactic relationships between potential instances of ontology concepts. For simplicity, we train both baseline models using the same architecture as our proposed method but only use a sub-set of the applied features. In the first baseline, we only apply features indicating word forms, lemmata and parts-of-speech as well as the order between words. For the second baseline, we use all features from the first baseline plus indicator features on syntactic relationships between words that potentially instantiate ontology concepts.

The results of both baselines and our full semantic role labeling model are summarized in Table 3. Using all features described in Section 4.3, our model achieves a precision and recall of 77.9% and 74.5%, respectively. The corresponding F_1-score, calculated as the harmonic mean between precision and recall, is 76.2%. The baselines only achieve F_1-scores of 45.1% and 69.3%, respectively. A significance test based on random approximate shuffling (Yeh, 2000) confirmed that the differences in results between our model and each baseline is statistically significant ($p<0.01$).

6 Conclusions

We conclude this paper with an outlook on how the work presented here contributes to computer-assisted software engineering. The main aim of the latter is to semi-automate the process of getting from software specifications to actual implementations. Ontologies and semantically annotated requirements can help achieve this goal by providing a meaningful and structured representations of software components. To truly assist software engineering, the mapping from requirements to ontology instances needs to be performed computationally. Towards this goal, we developed a semantic role labeling approach that automatically induces ontology-based representations from text. Our model achieves a high precision on this task and significantly outperforms two pattern-based baselines. In future work, we will empirically validate the usefulness of our proposed approach in downstream applications.

Acknowledgements

Parts of this work have been supported by the FP7 Collaborative Project S-CASE (Grant Agreement No 610717), funded by the European Commission.

References

Abbott, R. J. (1983). Program design by informal English descriptions. *Communications of the ACM 26*(11), 882–894.

Bengio, Y., R. Ducharme, P. Vincent, and C. Jauvin (2003). A neural probabilistic language model. *Journal of Machine Learning Research 3*, 1137–1155.

Björkelund, A., B. Bohnet, L. Hafdell, and P. Nugues (2010). A high-performance syntactic and semantic dependency parser. In *Coling 2010: Demonstration Volume*, Beijing, China, pp. 33–36.

Bohnet, B. (2010). Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, Beijing, China, pp. 89–97.

Booch, G. (1986). Object-oriented development. *IEEE Transactions on Software Engineering* (2), 211–221.

Ghosh, S., D. Elenius, W. Li, P. Lincoln, N. Shankar, and W. Steiner (2014). Automatically extracting requirements specifications from natural language. *arXiv preprint arXiv:1403.3142*.

Hajič, J., M. Ciaramita, R. Johansson, D. Kawahara, M. A. Martí, L. Màrquez, A. Meyers, J. Nivre, S. Padó, J. Štěpánek, et al. (2009). The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, pp. 1–18.

Harmain, H. M. and R. Gaizauskas (2003). Cm-builder: A natural language-based case tool for object-oriented analysis. *Automated Software Engineering 10*(2), 157–181.

Kof, L. (2004). Natural language processing for requirements engineering: Applicability to large requirements documents. In *19th International Conference on Automated Software Engineering, Workshop Proceedings*.

Meyers, A., R. Reeves, and C. Macleod (2008). *NomBank v1.0*. Linguistic Data Consortium, Philadelphia.

Mich, L. (1996). NL-OOPS: From natural language to object oriented requirements using the natural language processing system LOLITA. *Natural Language Engineering 2*(2), 161–187.

Mich, L., F. Mariangela, and N. I. Pierluigi (2004). Market research for requirements analysis using linguistic tools. *Requirements Engineering 9*(1), 40–56.

Nanduri, S. and S. Rugaber (1995). Requirements validation via automated natural language parsing. In *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences*, Volume 3, pp. 362–368.

Palmer, M., D. Gildea, and P. Kingsbury (2005). The Proposition bank: An annotated corpus of semantic roles. *Computational Linguistics 31*(1), 71–106.

Roth, M., T. Diamantopoulos, E. Klein, and A. Symeonidis (2014). Software requirements: A new domain for semantic parsers. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, Baltimore, Maryland, USA, pp. 50–54.

Roth, M. and K. Woodsend (2014). Composition of word representations improves semantic role labelling. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, Doha, Qatar, pp. 407–413.

Saeki, M., H. Horai, and H. Enomoto (1989). Software development process from natural language specification. In *Proceedings of the 11th International Conference on Software Engineering*, pp. 64–73.

van Lamsweerde, A. (2009). *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley.

Yeh, A. (2000). More accurate tests for the statistical significance of result differences. In *Proceedings of the 18th International Conference on Computational Linguistics*, Saarbrücken, Germany, pp. 947–953.