Extracting Ontologies from Software Documentation: a Semi-Automatic Method and its Evaluation

Marta Sabou [1]

Abstract. Rich and generic ontologies about web service functionalities are a prerequisite for performing complex reasoning tasks with web service descriptions. However, their acquisition is time-consuming and conditioned by the small number of web services available in certain domains. As a solution, we describe a semi-automatic method to extract such ontologies from software documentation, motivated by the observation that web services reflect the functionality of their underlying implementation. Further, we report on fine-tuning the extraction process by using a multi-stage evaluation method.

1 Introduction

Machine understandable semantics augmenting web services will facilitate their discovery and integration. OWL-S [3] is a major initiative in this direction: it is an ontology that provides basic concepts to describe web services. However, OWL-S provides only a generic frame for description which must be filled in with appropriate domain knowledge such as the type of offered functionality, the types of the corresponding input/output parameters and domain specific preconditions and effects. Such knowledge is provided by domain ontologies (DO) usually containing a hierarchy of tasks and several concepts which can participate as inputs/outputs in these tasks.

The quality of the used domain ontologies influences the complexity of reasoning tasks that can be performed with the semantic descriptions of web services. For many tasks (e.g. matchmaking) it is preferable that web services are described according to the same DO. This implies that the used DO should be *generic* enough to be used by many web service descriptions. By being generic, we do not mean that this ontology should contain all the knowledge needed by any possible web service, but rather that it can easily be extended with new domain knowledge whenever it is needed. Besides generality, DOs formally depict the complex relationships that exist between the domain concepts. Such *rich* descriptions allow performing complex tasks such as flexible matchmaking. We conclude that building quality DOs is at least as important as designing a generic web service description language such as OWL-S.

Building quality DOs is a real challenge since in many domains only a few web services are available. These are not sufficient for building generic and rich ontologies.

Our approach to the problem of building quality DOs is motivated by the observation that, since web services are simply exposures of existing legacy software to web-accessibility, there is a large overlap (often an one-to-one correspondence) between the functionality offered by a web service and that of the underlying software. Therefore we propose to build DOs by analyzing software programming interfaces (APIs). We wish to answer the following questions:

1. *Are APIs rich enough to serve as sources for building ontologies?*
2. *How to (semi-)automatically extract ontologies from APIs?*
3. *How to evaluate and fine-tune ontology extraction?*

We have conducted our work in the domain of RDF based ontology storage tools. Section 2 investigates the first question reporting on a manually built ontology from the APIs of three ontology storage tools. We regard this ontology as a "Golden Standard" for the extraction process which is described in Section 3. To enhance the extraction process we built a multi-stage evaluation method. Section 4 details this method, reports on how we have applied it in a specific case and shows how it helped us to fine-tune the extraction process. We present related work in Section 5, then conclude and point out future work in the last Section.

2 The Golden Standard

Tools for storing ontologies are of major importance for any semantic web application. While there are many tools offering ontology storage (a major ontology tool survey [4] reported on the existence of 14 tools), only very few are available as web services (two, according to the same survey). Therefore, in this domain it is problematic to build a good DO by analyzing only the available web services. Nevertheless, a good DO is clearly a must since we expect that many of these tools will become web services soon. Driven by this need, as well as willing to verify our hypothesis, we attempted to build a DO by analyzing the APIs of three tools in this domain: Sesame [1], Jena [7], KAON RDF API[8].

2.1 Manually Building the Domain Ontology

Method hierarchy. We identified overlapping functionality offered by the APIs of these tools and modelled it in a hierarchy (see Fig. 2). The process of building this domain ontology consisted in analyzing all methods of the three APIs, and introducing a main functionality category if at least two APIs offered methods with such functionality (for example KAON RDF API does not offer querying methods). Functionalities offered just by a single API were added as more specialized concepts with the goal of putting them in the context of generally offered functionality (for example the RQL querying is only provided by Sesame).

The class *Method* depicts one specific functionality (similar to the service Profile from OWL-S). According to our view, there are four main categories of methods for: adding data (*AddData*), removing data (*RemoveData*), retrieving data (*RetrieveData*) and querying (*QueryMethod*).

[1] Vrije Universiteit, Amsterdam, The Netherlands email: marta@cs.vu.nl

Naturally, several specializations of these methods exist. For example, depending on the granularity of the added data, methods exist for adding a single RDF statement (*AddStatement*) or a whole ontology (*AddOntology*). Note, that this hierarchy reflects our own conceptualization and does not claim to be unique. Indeed, from another perspective, one can regard query methods to be a subtype of data retrieval methods. We have chosen however to model them as a separate class as they require inputs of type *Query*. Besides the method hierarchy several domain concepts, in this case kinds of data structures, are described. We describe elements of the RDF Data Model (e.g. *Statement*, *Predicate*, *Object*, *ReifiedStatement*) and their relationships.

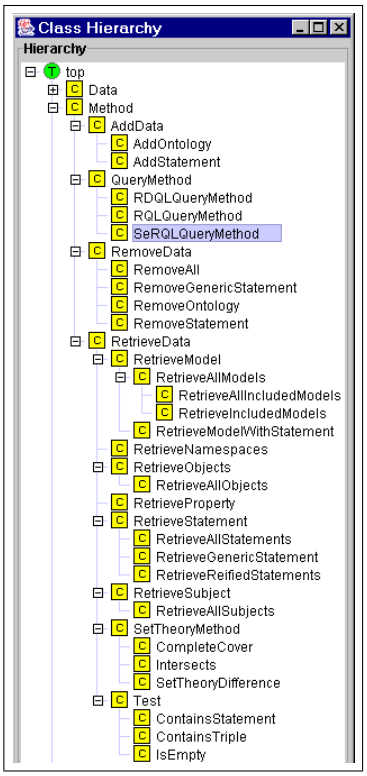

Figure 1: The Method hierarchy.

Ontology richness. To enrich our ontology we identified knowledge useful for several reasoning tasks. We enhanced the definition of methods in the following ways.

Characterizing methods by restricting their parameters. We define our methods by imposing restrictions on the type and cardinality of their parameters. For example, *AddOntology* methods are defined as methods which should accept an input of *Ontology* type. Similarly, any method returning a boolean is considered to be a *Test* method. Note that a method instance which adds an ontology and returns a boolean value stating the success of the transaction will belong to both classes of methods (*AddOntology* and *Test*). Formalizing these restrictions is straightforward.

Characterizing methods by their effect. For each method we have identified its effect. For example, adding a statement S to an ontology O will result in O containing that statement. Or adding an ontology O1 to an empty ontology O2 results in O1 and O2 having the same content. Finally, *RemoveAll* applied on an ontology causes that ontology to become empty.

Some methods expose special behavior. For example, *idempotent* methods produce the same effect independently of the number of sequential executions. Such are *AddData*, *RemoveAll*, *RemoveStatement* etc. Further, there exist pairs of methods which executed together result in a null operation, i.e. they cause no changes. For example adding and then removing the same data to an ontology leaves the ontology unchanged.

Methods which have the same effect are *equivalent*. While a correct formalization of the effects of each method should be enough to automatically deduce the equivalent methods, for now we have identified some equivalent pairs manually. For example, *RetriveAllStatements* returns all the RDF statements of an RDF ontology. The *RetriveGenericStatement* accepts three inputs for the *Subject*, *Predicate* and *Object* of the statement to be retrieved. If any of these inputs has the value null it acts like a wildcard. Calling this method with all its inputs null results in retrieving all statements of the ontology, which is the same effect as we get with the first method. Therefore the first method type is equivalent to a certain execution of the second one.

Our ontology allows us to declare functionality that is not directly provided but can be achieved with special values supplied to certain methods. For example, Sesame offers a single method for data retrieval which returns the whole ontology. In contrast, Jena offers many methods to retrieve data with different granularity. However, using Sesame's Query method with certain queries most of the retrieval functionality available in Jena can be achieved by Sesame as well. For example the effect of two method types of Jena can be achieved by using two Sesame queries: one retrieves a certain statement allowing wildcards, the second retrieves all statements.

```
RetrieveGenericStatement(S, P, O) <=>
 SeRQLQueryMethod(SELECT * FROM{X} Y {Z} where X=S, Y=P, Z=O)

RetrieveAllStatements()<=>
 SeRQLQueryMethod(SELECT * FROM{X} Y {Z})
```

From our analyzes it yielded that it is *possible* to build rich domain ontologies from software API documentation. These DO's contain complex relationships between the types of defined methods representing much stronger semantics than simple concept hierarchies of method types. Ideally one would want to automatically extract as much of these ontological relationships as possible. For now, our current efforts, presented next, address identifying the main types of method functionalities.

3 The ontology extraction method

The ontology extraction process consists of the following steps and involves the following data structures:

Data0 - The Corpus. Each document in the corpus contains the javadoc description of one method. Typically, the javadoc documentation of a method consists of a general description of the method functionality, followed by the description of the parameters, result type and exceptions to be thrown. See for example the *add* method of the Jena API. We exclude the syntax of the method because it introduces many irrelevant, technical terms such as *java*, *com*, *org*.

```
add
 Add all the statements returned by an iterator to this
 model.

 Parameters:
   iter - An iterator which returns the statements to be
   added.
 Returns:this model
 Throws: RDFException - Generic RDF Exception
```

Step 1 - Tokenise/POS tag the corpus - Automatic. After tokenisation we use the QTAG[2] probabilistic POS tagger to determine the part of speech of each token.

Step2 - Pair Extraction - Automatic. Using the POS tagged documents, we identify verb-noun pairs. We filter out these pairs using the regular expression identifier offered by the java.regex package. The choice for such a pattern is straightforward since in javadoc style documentation verbs express the functionality offered by the methods and their object usually identifies the data structure which is involved in this functionality. In a next step we lemmatize both verbs and nouns and present only the lemmatized pairs (if "removes model" and "remove models" are extracted, after lemmatization the resulting pair is "remove model"). This step results in a set of pairs extracted from the corpus.

[2] http://web.bham.ac.uk/o.mason/software/tagger/index.html

Step3 - Identifying Significant Pairs - Manual, with some support. We regard a pair significant for our task if it reflects the functionality of the method from whose description it was extracted. The ontology-engineer has to decide by himself which of the extracted pairs are significant. We support this decision in two ways. First, we provide an intuitive display of the pairs in which the engineer can inspect the method descriptions from which each pair was extracted and therefore decide on the significance of that pair. Second, we have experimented with several pair ranking schemes to determine the most significant pairs and to present the end-user with the pairs sorted according to their significance. These schemes range from measuring pair frequency to combining information about the weight of the constituting terms as well as the number of APIs in which a pair appears. Describing these schemes is beyond the scope of this paper. Step 3 filters out all the significant pairs.

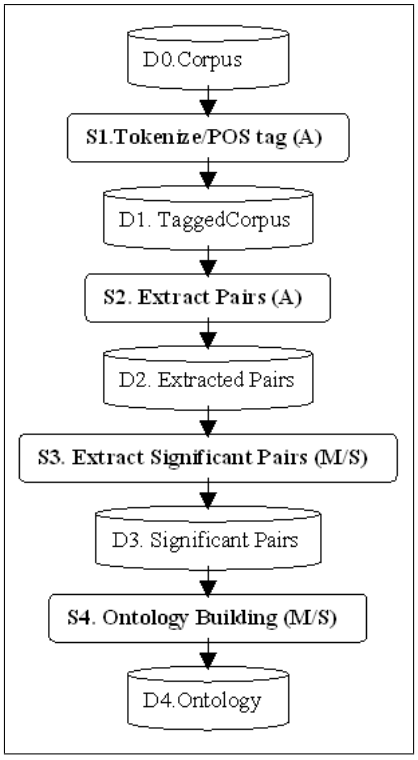

Figure 2: The Extraction process.

Step4 - Ontology Building - Manual with support. The ontology engineer attributes a concept to each significant pair. It is often the case that different pairs represent the same concept. For example, both "load graph" and "add model" can be attributed the same semantic category, *AddOntology*. The set of derived concepts is a basis for building the domain ontology. Arranging these concepts in a hierarchy is still a manual process.

3.1 Results

We applied the extraction mechanism on the collection of all methods of the three tools. The corpus consisted of 112 documents and resulted in the extraction of 180 pairs (80 distinct) from which 31 distinct concepts were derived. Table 1 shows all the individual extracted concepts (the number between brackets represents the number of times the corresponding concept was derived), divided in concepts that correspond to the ones in the manual ontology and concepts that were newly introduced because they were neglected while building the manual ontology. Eighteen concepts from the manual ontology were identified (out of 36).

A number of new concepts were extracted as well, which were not considered during the manual ontology building due to several reasons. First, concepts that denote implementation details (related to transactions, repositories) are tool specific functionalities and were not interesting for our task to determine RDF based functionality. Second, concepts related to the creation of ontology elements were ignored because only Jena offers such methods. Finally, "ModifyModel" actually denotes the effect of many methods and we overlooked it during modelling.

Nr.	Concept	New Concept
1	AddData	AbandonChanges
2	AddOntology (2)	BeginTransaction
3	AddStatement(2)	CommitTransaction
4	ContainsStatement	CreateOntology
5	ContainsTriple	CreateProperty
6	QueryMethod	CreateResource
7	RDQLQueryMethod	CreateStatement (2)
8	RQLQueryMethod	GetURL
9	SerQLQueryMethod	GetUsername
10	RemoveAll	ModifyModel
11	RemoveOntology	RetrieveRepositories
12	RemoveStatement(2)	SupportSetOperations
13	RetrieveAllStatements	SupportsTransactions
14	RetrieveData (2)	
15	RetrieveObject(2)	
16	RetrieveOntology(2)	
17	RetrieveProperty	
18	RetrieveResource	

Table 1. Extracted Pairs - Original Corpus

To verify the validity of the extraction process we applied it to a different corpus than the one which inspired its design. The new corpus contains the methods of four other ontology tools: InkLing[3], the completely rewritten API of Sesame, the Standford RDF API[4] and the W3C RDF model[5]. The corpus containing 75 methods resulted in 79 pairs (44 individual pairs) synthesized in 14 concepts shown in Table 2. We conclude that our method worked on a new corpus and allowed us to extract methods from each four main categories identified by the Golden Standard.

Nr.	Concept	New Concept
1	AddData(4)	CreateOntology
2	AddOntology	VerifyData
3	AddStatement	
4	ContainsTriple	
5	QueryMethod (3)	
6	RemoveAll	
7	RemoveStatement(3)	
8	RetrieveAllStatements	
9	RetrieveData	
10	RetrieveObject	
11	RetrieveProperty	
12	RetrieveSubject	

Table 2. Extracted Pairs - New Corpus

One of the inconveniences of the extraction method is that many of the extracted pairs were not significant and did not lead to any concepts. Inspecting all these pairs (as in step S3) becomes a huge overhead. Therefore we wish to maximize the number of extracted significant pairs in comparison to the insignificant ones.

[3] http://swordfish.rdfweb.org/rdfquery/
[4] http://www-db.stanford.edu/ melnik/rdf/api.html
[5] http://dev.w3.org/cvsweb/java/classes/org/w3c/rdf/

4 Fine-tuning by Evaluation

Addressing the conclusion of the previous Section, our goal is to fine-tune the extraction process so that the output contains a minimum number of insignificant pairs. The success of the ontology extraction process as a whole depends on the success of all the individual steps of the process. These steps are: 1) extracting pairs from the corpus (S1 and S2), 2) filtering out significant pairs (S3) and 3) attributing a concept to each significant pair (S4). Therefore we have devised an evaluation method to assess the quality of each extraction step. This helped us not only to provide a better output, but also to get an understanding of the process' performance and to improve it from other perspectives as well.

In this Section we present the main evaluation stages (4.1) and their employment for fine-tuning the extraction process (4.2).

4.1 A three stage evaluation process

We distinguish several stages of evaluation.

Stage 1 - evaluating pair extraction. At this stage we evaluate the quality of the first two steps of the ontology extraction process, i.e. the POS tagging (S1) and the pair extraction (S2) steps. We use the following terminology:

- all_{pairs} - all the manually identified pairs that we wish to extract with the pattern applied in S2.
- $valid_{pairs}$ - all pairs that fulfill the extraction pattern and were extracted from the corpus.
- $invalid_{pairs}$ - all pairs that do not fulfill the extraction pattern but were extracted from the corpus.

Further, we define the following metrics, adapted from the well-known information retrieval measures: recall and precision.

$$Recall = \frac{valid_{pairs}}{all_{pairs}}$$

$$Precision = \frac{valid_{pairs}}{valid_{pairs} + invalid_{pairs}}$$

Quality criteria. We consider a pair extraction successful if both metrics have a high value: high recall indicates that many valid pairs are extracted from the corpus, while high precision indicates the predominance of valid pairs.

Stage 2 - evaluating pair significance. Evaluating the extraction process from the point of view of pair significance targets the improvement of the third step of the ontology extraction process. We use the following terminology:

- $all\_significant_{pairs}$ - all valid pairs that were classified as significant during the manual inspection of the corpus.
- $significant_{pairs}$ - all extracted significant pairs.
- $insignificant_{pairs}$ - all extracted insignificant pairs.

Similar to the previous evaluation stage, we will compute recall and precision for the significant pairs. We define two metrics:

$$SRecall = \frac{significant_{pairs}}{all\_significant_{pairs}}$$

$$SPrecision = \frac{significant_{pairs}}{significant_{pairs} + insignificant_{pairs}}$$

Quality criteria. An extraction is successful if the ontology builder is presented with a high ratio of significant pairs from those existent in the corpus (high SRecall), and if there are only few insignificant pairs (high SPrecision) in the set of all extracted pairs.

Stage 3 - evaluating ontology coverage. At this stage we compare the extracted ontology with the manually built ontology. There has been little work in measuring similarity between two ontologies, one of the recent advances being the work published in [6]. We are interested to know how many of the concepts contained in the Golden Standard could be extracted, and therefore a simple lexical overlap measure suffices. We have adopted this measure from [2]. Let L_{O_1} be the set of all extracted concepts and L_{O_2} the set of concepts of the Golden Standard. The lexical overlap (LO) equals to the ratio of the number of concepts shared by both ontologies and the number of concepts we wish to extract:

$$LO(O_1, O_2) = \frac{|L_{O_1} \cap L_{O_2}|}{|L_{O_2}|}$$

Human ontology building is not perfect. We encountered cases when the extraction process prompted us to introduce new concepts which were overlooked during the manual process (see some examples in 3.1). Let us therefore introduce an ontology improvement (OI) metric which equals to the ratio of new pairs (expressed as the set difference between extracted and desired pairs) and all pairs of the manual ontology.

$$OI(O_1, O_2) = \frac{|L_{O_1} \setminus L_{O_2}|}{|L_{O_2}|}$$

As a quality criteria we aim to increase the value of both metrics.

4.2 Fine-tuning the Ontology Extraction Process

Methodologically, there are three main steps in a fine tuning process. First, we evaluate the performance of the extraction process for the evaluation metrics defined (4.2.1). Second, according to its performance we decide on several modifications that could enhance the performance of the extraction (4.2.2). Finally, we evaluate the enhanced process (on the original and a new dataset) to check if the predicted improvements took place, i.e. if the fine-tuning process was successful (4.2.3).

4.2.1 Evaluating the extraction process

We ran our extraction process on all the available javadoc documentation for a method. We used an extraction pattern to identify all verb-noun pairs, where the verb is in present tense, past tense or gerund (ing-form). Between the verb and the noun(phrase) there can be any number of determiners(e.g. "the", "a") and/or adjectives.

To fine-tune this extraction process we wanted to determine how it performs in the two different parts of the method description. Our observation was that the first textual description of the method (referred to as "text") was grammatically much more correct than the rest of the method description dealing with the parameters of the method (referred to as "parameters"). We have also observed that the text part contains the pairs describing the functionality of the method, while in the parameters part of the descriptions verbs mostly describe the parameters of the method. Another observation was that the predominant number of verbs are in their base (present) form, especially those that describe the method functionalities. Accordingly, to verify our observations, we have evaluated the ontology extraction process

in these two different parts of the method description and for the three different verb forms.

Stage 1 - evaluating pair extraction. To perform this evaluation step we manually counted the number of extractable pairs in the corpus, in both parts of the description (text and parameter) and for all three verb tenses.

Doc part/verb form	Text	Parameters	Corpus
Present	130	41	171(88%)
Past Tense	0	1	1(1%)
Gerund	10	12	22(11%)
Total	140(72%)	54(28%)	194(100%)

Table 3. Distribution of pairs over description parts and verb forms.

Analyzing the results depicted in Table 3, we conclude that there are much more pairs in the textual description (72% of all pairs in the corpus) compared to the parameter section (28%). Also, most pairs contain verbs in present form, in both areas of the documents. Globally, pairs with present tense verbs represent 88% of the pairs in the corpus, compared to 1% for pairs having a past tense verb and 11% for gerund based pairs.

After the extraction process, we have counted the valid and invalid pairs and computed the precision and recall of the pair extraction, as shown in Table 4.

		Text	Parameter	Corpus
Present	Valid	85	28	113
	Invalid	8	1	9
	Recall	0.65	0.68	0.66
	Precision	0.91	0.96	0.93
Past Tense	Valid	0	1	1
	Invalid	19	11	30
	Recall	-	1	1
	Precision	0	0.08	0.03
Gerund	Valid	9	12	21
	Invalid	1	3	4
	Recall	0.9	1	0.95
	Precision	0.9	0.8	0.84
Total	Valid	94	41	135
	Invalid	28	15	43
	Recall	**0.76**	**0.67**	**0.69**
	Precision	**0.73**	**0.77**	**0.76**

Table 4. Evaluation of the extraction process for stage 1.

We derive several observations. First, the recall of pairs with present tense verbs is quite low (0.66 globally). This is due to the fact that, often, present tense verbs appear on an unusual position - the first word in the sentence - a position being attributed to nouns. The used POS tagger often fails in such cases, especially when the verbs can be mistaken for nouns (e.g. lists, uploads). Second, the precision of extracting present tense verbs is high (0.93 globally). Finally, both the recall (0.69) and the precision(0.76) of the extraction on the corpus as a whole are low. The decrease of recall is caused by the existence of many present tense verbs which are often mistaken for nouns. The precision is lowered by the past tense verbs. We conclude that, to increase recall we need a specialised POS tagger that would recognise verbs on the first position in a sentence. Second, to increase precision we can simply give up filtering past tense verbs.

Stage 2 - evaluating pair significance. As a first step of this stage we have identified all significant pairs in the corpus, for all of the six categories of verbs. The value between brackets in Table 5 shows the ratio of significant and all pairs in the corresponding categories. We refer to it as significance.

Doc part/verb tense	Text	Parameter	Corpus
Present	109(0.84)	20(0.49)	129(0.75)
Past Tense	0(0)	0(0)	0(0)
Gerund	5(0.5)	3(0.25)	8(0.36)
Total	114(0.81)	23(0.43)	137(0.7)

Table 5. Significance of different pair categories.

We can derive a set of observations based on this data. First, the significance in the category of pairs based on present tense verbs is much higher than for pairs containing verbs in other forms. This is true for both parts of the description, and for the whole corpus as well. Second, the significance in the textual description (0.81) is much higher than in the parameter part (0.43). Third, the significance of the corpus is affected by the low significance in the parameter section.

Given that only a few significant pairs have past tense and gerund verbs, we have decided to measure the significance recall and precision only for the text and parameter part of the description, this time taking all verb forms globally into account (see Table 6).

	Text	Parameter	Corpus
Significant	73	16	89
SRecall	0.64	0.69	**0.65**
SPrecision	0.59	0.28	**0.5**

Table 6. Evaluation of the second stage.

Low significance recall (0.65) shows that a lot of significant pairs are not extracted from the corpus. This is a direct consequence of the low extraction recall of pairs in general (i.e. many of the pairs which are not extracted are significant). The significance precision is almost double in the textual description (0.59) versus the parameter description (0.28). This reflects that pairs extracted from the textual part are much more likely to be significant than pairs extracted from the parameter part. Globally, the significance precision of the corpus is quite low (0.5), mostly due to the low significance precision of the extraction from the parameter part.

Stage 3 - evaluating ontology coverage. The individual extracted concepts are presented in Table 1. Eighteen concepts from the manual ontology were identified (out of 36). This yields in an lexical overlap of $LO = 0.5$, a very good result given the fact that lot of significant pairs were not extracted at all from the corpus. Besides this, 13 new concepts were identified, therefore resulting in an ontology improvement of $OI = 0.36$. In other words we were able to extract half of the concepts that we extracted manually and we suggested improvements that could enlarge the manual ontology with 36%.

4.2.2 Conclusions for Enhancements

Based on the performance of the extraction process we decided to perform two major modifications in order to achieve our fine-tuning

goal. First, we decided to ignore the parameter section, because it contains a small number of verbs (28%), and only 43% of the existing pairs are significant. This translates in a negative influence over the significance recall of the corpus. Also, we will only extract pairs with present tense verbs, because they represent the majority of verbs (88%) and are predominantly significant, especially in the text section. These measures serve the decrease of insignificant pairs but do not solve the recall problem. For that we need to train the POS tagger, a task regarded a future work.

4.2.3 Quality increase of the enhanced process

The particularities of the corpus influenced the fine-tuning of the process. Therefore we check the improvement of the performance both on the original corpus and a new corpus. Table 7 summarizes the evaluation results for the original and then the enhanced extraction process applied on the two data sets.

		Original Corpus		**New Corpus**	
Ev.Step	**Ev. Matrics**	**Orig.**	**Enh.**	**Orig.**	**Enh.**
1	Recall	0.69	0.65	-	-
	Precision	0.76	0.98	-	-
2	SRecall	0.65	0.62	-	-
	SPrecision	0.5	0.78	0.48	0.67
3	LO	0.5	0.39	0.3	0.25
	OI	0.36	0.36	0.05	0.03

Table 7. Comparison of extraction performance of the original and the enhanced process for two data sets

For the first corpus, the enhanced process did not improve the recall. However the precision of the output was highly increased in both evaluation stages: 22% for the first stage and 28% for the second. This serves our goal to present the ontology engineer with as few insignificant pairs as possible. Despite the heavy simplifications the ontological loss was minimal (11%). The enhanced process only missed 4 concepts because their generating pairs appeared in the parameter section (*QueryMethod*) or had verbs in past tense form (*RetrieveObject*, *RetrieveProperty*, *RetrieveResource*).

For the second corpus, we did not perform any manual inspection of the corpus and therefore could only compute the SPrecision and the ontology comparison metrics. Similar to the previous corpus, the pair significance increased (almost 20%) and resulted in a small decrease of ontology overlap (only 5%).

This Section described an evaluation method that allowed us to get a better understanding of the corpus and the behavior of the extraction process on this corpus. The derived observations helped us to fine-tune to extraction so that, despite heavy simplification of the input and the extraction pattern, a much "cleaner" output was achieved without loosing significant ontological information. The improvements were verified on the corpus used for fine-tuning and a new corpus as well.

5 Related Work

The problem of automating the task of web service semantic acquisition, to our knowledge, was only tackled by the work of Heß[5]. They employ the Naive Bayes and SVM machine learning algorithms to classify WSDL files (or web forms) in manually defined task hierarchies. Our work is complementary, since we address the acquisition of such hierarchies. Also, our method does not rely on any manually built annotation for training as the machine learning techniques do.

From the ontology learning community, the work that is methodologically closest to our solution is the work of Cimiano[2]. They identify a set of verb-noun pairs with the intention to build a taxonomy by using Formal Concept Analysis theory. As a result, their metrics for filtering the best pairs are different than ours. They define and evaluate three such metrics. Finally, they compare their extracted ontology to a manually built one. We have adopted one of the metrics they use for ontology comparison.

6 Conclusions

The hypothesis which drove the work in this paper was that, given the similarity in functionality between web services and their underlying implementation, domain ontologies used to semantically describe web services could be derived from the documentation associated with the underlying software. In this paper we show that APIs of tools are rich sources for building complex domain ontologies. Currently we only investigated the domain of RDF based ontology stores but, in the future, we will extend our work to other domains as well.

We also describe a low-investment method for extracting types of method functionalities. Even if this is just a small part of the derivable information from APIs, it is already a substantial support for the ontology engineer. One of the limitations of our work is that our corpus is very small for applying statistical techniques. To address this issue we plan to enlarge our corpus and also to use extraction methods better suited for small corpora.

Finally, we use a multi-stage evaluation method to asses the quality of the extraction process and to enhance it accordingly. The enhanced process outperforms the original process on the original and a new corpus as well.

Our final conclusion is that the results of this work are encouraging to further investigate ontology extraction from software APIs.

ACKNOWLEDGEMENTS

We thank P. Cimiano and J.Tane for their support with the Text-ToOnto tool, which served as a starting point for our implementation.

REFERENCES

[1] J. Broekstra, A. Kampman, and F. van Harmelen, 'Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema', in *Proceedings of the First International Semantic Web Conference*, eds., I. Horrocks and J. A. Hendler, LNCS, Sardinia, Italy, (2002).

[2] P. Cimiano, S. Staab, and J. Tane, 'Automatic Acquisition of Taxonomies from Text: FCA meets NLP', in *Proceedings of the ECML/PKDD Workshop on Adaptive Text Extraction and Mining, Cavtat–Dubrovnik, Croatia*, (2003).

[3] The OWL-S Services Coalition. OWL-S: Semantic Markup for Web Services. White Paper. http://www.daml.org/services/owl-s/1.0/owl-s.pdf, 2003.

[4] A. Gomez Perez. A survey on ontology tools. OntoWeb Delieverable 1.3, 2002.

[5] A. Heß and N. Kushmerick, 'Machine Learning for Annotating Semantic Web Services', in *AAAI Spring Symposium on Semantic Web Services*, (March 2004).

[6] A. Maedche and S.Staab, 'Measuring similarity between ontologies', in *Proceedings of EKAW*. Springer, (2002).

[7] B. McBride, 'Jena: A Semantic Web Toolkit', *IEEE Internet Computing*, **6**(6), 55–59, (November/December 2002).

[8] D. Oberle, R. Volz, B. Motik, and S. Staab, *An extensible open software environment*, chapter III: Ontology Infrastructure, 311–333, International Handbooks on Information Systems, Springer, 2003.