From Software APIs to Web Service Ontologies: A Semi-automatic Extraction Method

Marta Sabou

Dept. of AI, Vrije Universiteit, Amsterdam, The Netherlands, marta@cs.vu.nl

Abstract. Successful employment of semantic web services depends on the availability of high quality ontologies to describe the domains of these services. As always, building such ontologies is difficult and costly, thus hampering web service deployment. Our hypothesis is that since the functionality offered by a web service is reflected by the underlying software, domain ontologies could be built by analyzing the documentation of that software. We verify this hypothesis in the domain of RDF ontology storage tools. We implemented and fine-tuned a semi-automatic method to extract domain ontologies from software documentation. The quality of the extracted ontologies was verified against a high quality hand-built ontology of the same domain. Despite the low linguistic quality of the corpus, our method allows extracting a considerable amount of information for a domain ontology.

1 Introduction

The promise of the emerging Semantic Web Services field is that machine understandable semantics augmenting web services will facilitate their discovery and integration. Several projects used semantic web service descriptions in very different application domains (bioinformatics grid[17], Problem Solving Methods[10]). A common characteristic of these descriptions is that they rely on a generic description language, such as OWL-S[3], to specify the main elements of the service (e.g. inputs, outputs) and on a ontology containing knowledge in the domain of the service such as the type of offered functionality (e.g. TicketBooking, CarRental) or the types of service parameters (e.g. Ticket, Car).

The quality of the domain ontologies used influences the complexity of reasoning tasks that can be performed with the semantic descriptions. For many tasks (e.g. matchmaking) it is preferable that web services are described according to the same domain ontology. This implies that the domain ontology used should be *generic* enough to be used in many web service descriptions. Domain ontologies also formally depict the complex relationships that exist between the domain concepts. Such *rich* descriptions allow performing complex reasoning tasks such as flexible matchmaking. We conclude that building quality (i.e. generic and rich) domain ontologies is at least as important as designing a generic web service description language such as OWL-S.

The acquisition of semantic web service descriptions is a time consuming and complex task whose automation is desirable, as signaled by many researchers in

S.A. McIlraith et al. (Eds.): ISWC 2004, LNCS 3298, pp. 410–424, 2004.

this field, for example [16]. Pioneer in this area is the work reported in [6] which aims to learn web service descriptions from existing WSDL[1] files using machine learning techniques. They classify these WSDL files in manually built task hierarchies. Complementary, we address the problem of building such hierarchies, i.e. domain ontologies of web service functionalities (e.g. TicketBooking). This task 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 domain ontologies is motivated by the observation that, since web services are simply exposures of existing software to web-accessibility, there is a large overlap (often one-to-one correspondence) between the functionality offered by a web service and that of the underlying implementation. Therefore we propose to build domain ontologies by analyzing application programming interfaces(APIs). We investigate two research questions:

1. *Is it possible and useful to build a domain ontology from software APIs?*
2. *Can we (semi-)automatically derive (part of) a domain ontology from APIs?*

This paper reports on work performed in the domain of RDF based ontology stores. Section 2 tackles the first question by presenting an ontology which was manually built from API documentation and reporting on using this ontology to describe existing web services. To address the second question, we present a (semi-) automatic method to derive (part of) a domain ontology in Section 3 and describe experimental results in Section 4. We use the manually built ontology as a Golden Standard for evaluating the result of the extraction process. We list related work in Section 5, then conclude and point out future work in Section 6.

2 Constructing a 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 [5] reported on the existence of 14 such 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 domain ontology by analyzing only the available web services. Nevertheless, a good domain ontology is clearly a must since we expect that many of these tools will become web services soon. We attempted to build a domain ontology by analyzing the APIs of three tools (Sesame [1], Jena [9], KAON RDF API[7]). We report on this ontology[2] in Section 2.1 then show that we could use it to describe web services in Section 2.2.

2.1 Manually Building the Domain Ontology

Method hierarchy. We identified overlapping functionalities offered by the APIs of these tools and modelled it in a hierarchy (see Fig. 2). The class *Method* depicts one specific functionality (similar to the OWL-S *Profile* concept).

[1] WSDL is the industry standard for syntactic web service descriptions.
[2] Available at http://www.cs.vu.nl/~marta/apiextraction.

According to our view, there are four main categories of methods for: adding data (*AddData*), removing data (*RemoveData*), retrieving data (*RetrieveData*) and querying (*QueryMethod*). 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, we also describe elements of the RDF Data Model (e.g. *Statement, Predicate, ReifiedStatement*) and their relationships.

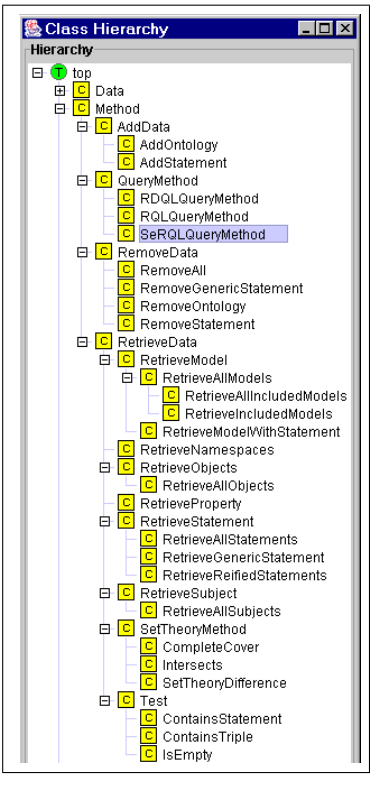

Fig. 1. The Method hierarchy.

During ontology building we introduced a main functionality category (a direct subclass of *Method*) if at least two APIs offered methods with such functionality (e.g. querying is only supported by Sesame and Jena). Functionalities offered just by one API were added as more specialized concepts with the goal of putting them in the context of generally offered functionality (e.g. SeRQL querying is only provided by Sesame).

Ontology richness. We enriched our ontology by identifying knowledge useful for several reasoning tasks. We enhanced the definition of methods in multiple ways such as: imposing restrictions on the type and cardinality of their parameters, describing their effects and types of special behavior (e.g. *idempotent*). We also identified methods which have the same effect, i.e *equivalent*. Knowledge about equivalent methods is important for tasks such as matchmaking. For more information on this ontology the reader is referred to [13]. We conclude that APIs are rich enough to make the building of a rich domain ontology *possible*.

2.2 Using the Domain Ontology for Web Service Description

We used the domain ontology to describe the web-interface of Sesame. The domain ontology offered all required concepts to describe the functionality of the web service and also indicated how the concepts we have used fit in a larger spectrum of possible functionalities. This is a clear advantage in comparison to the domain ontology[3] we have built in a previous project solely for the Sesame web

[3] Available at http://www.cs.vu.nl/~marta/apiextraction.

service [12]. While that ontology was satisfactory for the goals of that project, it was qualitatively inferior to the one we have obtained by analyzing the APIs of multiple tools.

We checked the generality of our domain ontology by using it to describe a web-interface which was not considered during ontology building. This is the recent W3C RDF Net API submission[4] which proposes a generic interface to be implemented by any service that wishes to expose RDF to client applications, therefore making the first step towards a standard RDF based web service interface. The submission distills six functionalities that any RDF engine should offer. Four of them (*query*, *getStatements*, *insertStatements*, *removeStatements*) are instances of the *QueryMethod, RetrieveGenericStatement, AddStatement* and *RemoveStatement* concepts respectively. The other two (*putStatements*, *updateStatements*) are complex operations and they correspond to sequences of concepts (*RemoveAll, AddStatement* and *RemoveStatement, AddStatement* respectively).

The manually built ontology is also an integrated part of the KAON Application Server, a middleware system which facilitates the interoperability of semantic web tools (e.g. ontology storages, reasoners). Many middleware tasks can be enhanced by reasoning with the semantic descriptions of the functionality and implementation details of registered tools [14]. Also, the middleware can expose the functionality of any registered tool as a web service and generate the semantic description of the web service from that of the underlying tool.

Summarizing Section 2, we note that by analyzing software APIs we were able to build a rich domain ontology and successfully employ it to describe (1) a web service whose API served as material for defining the ontology, (2) an emerging standard in this domain and (3) RDF storage tools registered with a semantic middleware. The manually built ontology serves as a Golden Standard for evaluating the semi-automatic extraction process presented next.

3 Semi-automatically Extracting Concepts

In this section we present our semi-automatic ontology extraction process (Section 3.1) and two possible refinements of this method (Sections 3.3 and 3.2).

3.1 The Extraction Process

The goal of the extraction process is to identify, by analyzing software API documentation, a set of concepts describing generally offered functionalities in the application domain of that software. It consists of several steps.

Step1 - Tokenise/POS tag the corpus - Automatic. This step preprocesses the corpus. After tokenisation we use the QTAG probabilistic Part Of Speech (POS) tagger[5] to determine the part of speech of each token.

Step2 - Extract Pairs - Automatic. This step extracts a set of potentially useful verb-noun pairs from the POS tagged documents. We extract all

[4] http://www.w3.org/Submission/2003/SUBM-rdf-netapi-20031002/
[5] http://web.bham.ac.uk/o.mason/software/tagger/index.html

pairs where the verb is in present tense, past tense or gerund form. Between the verb and the noun(phrase) we allow any number of determiners(e.g. "the", "a") and/or adjectives. The choice for such a pattern is straightforward since in javadoc style documentation verbs express the functionality offered by the methods and the following noun phrase (often their object) usually identifies the data structure involved in this functionality. We reduce the number of extracted pairs by lemmatization, so if "removes model" and "remove models" are extracted, the resulting pair is "remove model". We define the resulting pairs as *distinct*.

Step3 - Extract Significant Pairs - Manual, with some support. This step identifies all the significant pairs from the set of previously extracted distinct pairs. We define a pair to be *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. Manually inspecting these pairs is time consuming, especially in cases when the set of extracted pairs contains many pairs of low importance for the ontology engineer. To solve this problem we adopted two strategies. First, we fine-tuned the extraction process so that the output contains a minimal number of insignificant pairs (Section 3.2). Second, we developed a set of ranking schemes to order the pairs according to their significance (Section 3.3). We present these refinement methods in different subsections due to their complexity, but we consider them closely related to the extraction process itself.

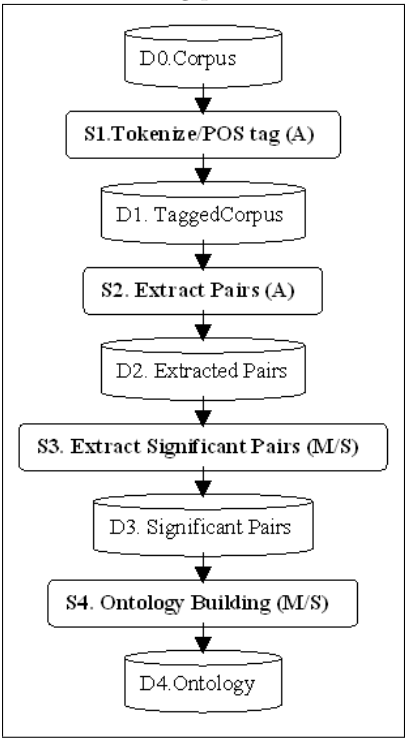

Fig. 2. The Extraction process.

Step4 - Ontology Building - Manual with support. Finally, the ontology engineer derives a concept from each significant pair. Often different pairs represent the same concept. For example, both "load graph" and "add model" correspond to the *AddOntology* concept. These concepts are the bases of the final ontology. Even if arranging them in a hierarchy is still a manual task, we support the ontology engineer with visual methods in discovering subclass relationships (Section 4.2). Further, we present the two refinement methods.

3.2 Refining the Extraction by Evaluation

Using a multi-stage evaluation method we aim to fine-tune the extraction process so that the output contains a minimum number of insignificant pairs. We present the evaluation metrics here and exemplify their use for fine-tuning in Section 4.3.

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 and the pair extraction steps. We denote with all_{pairs} all manually identified pairs that we wish to extract and with $valid_{pairs}$ the subset of these pairs that are extracted. Extracted pairs that do not fulfill the extraction pattern are denoted by $invalid_{pairs}$. We define two metrics, adapted from the well-known information retrieval recall and precision.

$$Recall = \frac{valid_{pairs}}{all_{pairs}} \quad \text{and} \quad Precision = \frac{valid_{pairs}}{valid_{pairs} + invalid_{pairs}}$$

Quality criteria. We consider a pair extraction successful if both metrics have a high value. High recall testifies that many valid pairs were extracted from the corpus and high precision shows a low number of invalid pairs.

Stage 2 - evaluating pair significance. Evaluating the extraction process from the point of view of pair significance targets the improvement of the third ontology extraction step. For this, we count all pairs that were classified as significant during the manual inspection of the corpus (all_sign_{pairs}), all extracted significant ($sign_{pairs}$) and all extracted insignificant ($insign_{pairs}$) pairs. Similar to the previous stage, we compute recall and precision for the significant pairs as follows.

$$SRecall = \frac{sign_{pairs}}{all\_sign_{pairs}} \quad \text{and} \quad SPrecision = \frac{sign_{pairs}}{sign_{pairs} + insign_{pairs}}$$

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).

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 [8]. 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, as introduced in [2], suffices. 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.

$$LO(O_1, O_2) = \frac{|L_{O_1} \cap L_{O_2}|}{|L_{O_2}|} \quad \text{and} \quad OI(O_1, O_2) = \frac{|L_{O_1} \setminus L_{O_2}|}{|L_{O_2}|}$$

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, as illustrated in Section 4.2. We introduce an ontology improvement (OI) metric which equals to the ratio of new concepts (expressed as the set difference between extracted and desired concepts) and all concepts of the manual ontology. As a quality criteria we aim to increase the value of both metrics.

3.3 Refining the Extraction by Measuring Significance

The previous subsection addressed the problem of decreasing the effort of the ontology engineer when determining significant pairs (step 3) by evaluating and fine-tuning the extraction so that the number of extracted insignificant pairs is minimal. In this section we adopt a different strategy: we use pair ranking schemes to order the extracted pairs according to their significance. We built these schemes on pair frequency, term weight and API relevance considerations.

Pair Frequency. Our first intuition was that significant pairs are pairs that appear often and also in many different method descriptions. Denoting pf_{pair} the frequency of the pair in the corpus and df_{pair} the number of documents in the corpus in which the pair appears, we compute the rank of a pair as:

$$rank_{pair} = pf_{pair} * df_{pair}$$

Term Weight. This weighting scheme considers that the rank of a pair is directly proportional with the weight of its components. We give the same importance to the weight of the component verb and noun. Other variations in which one of these terms is considered more important could be investigated in the future.

$rank_{pair} = w_{verb} * w_{noun}$ where the weight of a term in the corpus is:

$$w_i = \sum_k wd_{i,k} = \sum_k (tf_{i,k} * df_i) = df_i * \sum_k tf_{i,k} = df_i * cf_i$$

- $wd_{i,k}$ - is the weight of term i in document k. It is defined as $wd_{i,k} = tf_{i,k} * df_i$;
- $tf_{i,k}$ - is the frequency of term i in document k;
- df_i - is the number of documents in which term i appears;
- cf_i - is the frequency of term i in the whole corpus.

API Relevance. Our final ranking scheme filters the pairs based on the number of APIs in which they appear. Intuitively, this corresponds to a different ontology building strategy than the one supported by the previous two schemes. Before we derived the "union" of all functionalities while here the focus is on their "intersection". Accordingly, pairs found in the maximum number of APIs are given the highest rank. Pairs belonging to the same number of APIs, are ordered using the Term Weight ranking scheme.

4 Experimental Results

4.1 Experimental Setup

The goal of our experiments is to test the basic extraction process and the enhancements achieved with the refinement methods. We conduct three sets of experiments. First, we apply the basic extraction process and examine the extracted concepts (4.2). Second, we use the evaluation method (1) to get an insight

in the internal working of the basic extraction, (2) to suggest improvements for the extraction method and (3) to evaluate if the enhanced extraction is superior to the basic one (4.3). Finally, we perform a comparative evaluation of the pair ranking schemes on the output of the basic extraction process (4.4).

Corpora. We used two distinct corpora[6] in our experiments. The first corpus, *Corpus 1*, contains the documentation of the tools we used to build the manual ontology (Jena, KAON RDF API, Sesame) and accounts to 112 documents. The second corpus, *Corpus 2*, contains 75 documents collected from four tools: InkLing[7], the completely rewritten API of Sesame, the Stanford RDF API[8] and the W3C RDF model[9]. Each document in the corpora contains the javadoc description of one method. This description consists of a general description of the method functionality (termed "text"), followed by the description of the parameters, result type and the exceptions to be thrown (termed "parameters"). See for example the *add* method of the Jena API. We exclude the syntax of the method because it introduces 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
```

4.2 Extracting Concepts with the Basic Extraction Process

From Corpus 1 we extracted 180 pairs (80 distinct after lemmatization in Step 2) from which 31 distinct concepts were distilled. The first half of Table 1 lists these concepts, divided in 18 concepts already identified in the manual ontology (first column) and 13 new concepts which were ignored during the manual ontology building due to several reasons, as follows (second column). First, concepts that denote implementation related details (transactions, repositories) are tool specific functionalities and were not interesting for our task to determine RDF based functionalities. Then, concepts related to the creation of elements were ignored because only Jena offers such methods. "ModifyModel" actually denotes the effect of many methods that we ignored while modelling.

To check the applicability of our method to data sets that did not influence its design, we have applied it to Corpus 2. The extraction resulted in 79 pairs (44 distinct) synthesized in 14 concepts shown in the second half of Table 1. We conclude that our method worked on a completely new corpus allowing us to extract concepts for each four main categories identified manually.

The significant pairs identified in the third extraction step are currently only used to derive a set of concepts, however it is possible to determine their (and the

[6] The corpora are available on http://www.cs.vu.nl/~marta/apiextraction.
[7] http://swordfish.rdfweb.org/rdfquery/
[8] http://www-db.stanford.edu/ melnik/rdf/api.html
[9] http://dev.w3.org/cvsweb/java/classes/org/w3c/rdf/

Table 1. Extracted existing and new concepts from both experimental corpora.

Corpus 1		Corpus 2	
Concept	**New Concept**	**Concept**	**New Concept**
AddData	AbandonChanges	AddData(4)	CreateOntology
AddOntology (2)	BeginTransaction	AddOntology	VerifyData
AddStatement(2)	CommitTransaction	AddStatement	
ContainsStatement	CreateOntology	ContainsTriple	
ContainsTriple	CreateProperty	EvalauteQuery (3)	
QueryMethod	CreateResource	RemoveAll	
RDQLQueryMethod	CreateStatement (2)	RemoveStatement(3)	
RQLQueryMethod	GetURL	RetrieveAllStatements	
SerQLMethod	GetUsername	RetrieveData	
RemoveAll	ModifyModel	RetrieveObject	
RemoveOntology	RetrieveRepositories	RetrieveProperty	
RemoveStatement(2)	SupportSetOperations	RetrieveSubject	
RetrieveAllStatements	SupportsTransactions		
RetrieveData (2)			
RetrieveObject(2)			
RetrieveOntology(2)			
RetrieveProperty			
RetrieveResource			

corresponding concepts') hierarchical relationships. We experimented with the Cluster Map [4] visualization technique, developed by Aduna (www.adnua.biz), to highlight potential subclass relationships. The Cluster Map was developed to visualise the *instances* of a number of *classes*, organized by their classifications.

Figure 3 displays API methods (small spheres - playing the role of *instances*) grouped according to the significant pairs that were extracted from their descriptions (bigger spheres with an attached label stating the name of the pair - considered *classes* here). Balloon-shaped edges connect instances to the class(es) they belong to. We observe that all *instances* of the rql, rdql and serql query pairs (i.e. all methods from which these pairs were extracted) are also *instances* of the query pair, hence intuitively indicating that the concepts resulting from these pairs are in a subclass relationship, where the concept derived from the "evaluate query" pair is the most generic concept.

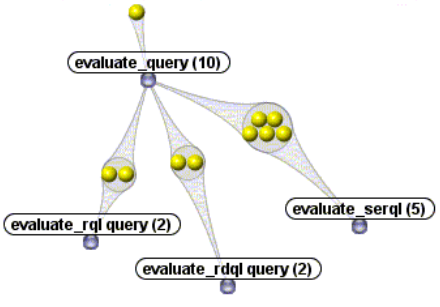

Fig. 3. Hierarchy visualisation.

We observe that, in both corpora, the number of distinct pairs (80, 44) is mush higher than the finally derived concepts (31, 14). To better understand this behavior of the extraction process we employ the evaluation method defined in Section 3.2, as described in the next Section.

4.3 Fine-Tuning the Extraction Process by Evaluation

In this section we describe the fine-tuning of the extraction process by using the evaluation method. Methodologically, there are three main steps in a fine tuning process. First (A), we evaluate the performance of the extraction process for the evaluation metrics defined. Second (B), according to its performance we decide on several modifications that could enhance the performance. Finally (C), we evaluate the enhanced process (on the original and a new corpus) to check if the predicted improvements took place, i.e. if the fine-tuning process was successful.

A) Evaluating the extraction process. The original extraction process was applied on Corpus 1. We observed that, in this corpus, the text section is grammatically more correct than the parameters section. Also, the predominant number of verbs were in present form, especially those that describe 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. A summary of the performance evaluation for the original extraction is shown in the third column of Table 2. Readers interested in the detailed evaluation data are referred to [13].

Stage 1 - evaluating pair extraction. We have counted the extracted valid and invalid pairs and computed the precision and recall of the pair extraction per verb category. The recall of pairs with present tense verbs is quite low (0.65) because, in the text area, these verbs appear in an unusual position - the first word in the sentence - a position usually being attributed to nouns. The POS tagger often fails in such cases, especially when the verbs can be mistaken for nouns (e.g. lists). In contrast, the precision of extracting present tense verbs is high. Both the recall and the precision of the extraction on the corpus as a whole are low (see Table 2) due to the existence of many present tense verbs which are often mistaken for nouns (recall) and the past tense verbs (precision).

Stage 2 - evaluating pair significance. Low significance recall (0.64 for text, 0.69 for parameters, 0.65 globally) 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 (e.g. many of the pairs which are not extracted are significant). The significance precision is almost double in the text (0.59) versus the parameter (0.28) section. Therefore pairs extracted from the textual part are much more likely to be significant than pairs extracted from the parameter part. This heavily affects the precision of the whole corpus (0.5).

Stage 3 - evaluating ontology coverage. The first half of Table 1 shows all distinct extracted concepts. Eighteen concepts from the manual ontology were identified yielding in a lexical overlap of $LO = 0.5$, a very good result given that a lot of significant pairs were not extracted from the corpus. Besides this, 13 new concepts were identified resulting in an ontology improvement of $OI = 0.36$. In other words, we extracted half of the concepts that we identified manually and we suggested improvements that could enlarge the ontology with 36%.

B) Conclusions for enhancements. We decided (1) to ignore the parameter section, because it heavily lowers the SPrecision of the whole corpus. Also, (2)

we will only extract pairs with present tense verbs, because there are only a few verbs in other forms but they negatively influence the pair retrieval precision of the corpus. These measures serve the decrease of insignificant pairs extracted from the corpus in step 2 but do not solve the recall problem. For that we need to train the POS tagger, a task regarded as future work.

C) Quality increase of the enhanced process. The particularities of Corpus 1 influenced the fine-tuning of the process. Therefore we check the improvement of the performance on both corpora. Table 2 summarizes the evaluation results for the original and the enhanced extraction process. The enhanced process uses a modified linguistic pattern (only extracts present tense verbs) and is only applied on the textual part of the method descriptions.

Table 2. Comparison of the original and the enhanced extraction for two corpora.

		Corpus 1		Corpus 2	
Ev.Step	**Ev. Matrics**	**Original**	**Enhanced**	**Original**	**Enhanced**
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

For Corpus 1 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 reduce the number of insignificant pairs extracted in step 2. 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 Corpus 2, we did not perform any manual inspection and therefore could only compute the SPrecision and the ontology comparison metrics. Similarlly, the pair significance increased (with 20%) resulting in a small decrease of ontology overlap (with 5%).

4.4 Refining the Extraction by Measuring Significance

Pair ranking schemes order the extracted pairs according to their significance. We evaluated which of them is the best discriminator for pair significance. To evaluate the performance of ranking schemes, we have manually determined the type of all pairs, extracted in the second step of the basic extraction process from Corpus 1, as: (1) insignificant, (2) significant leading to a new concept and (3) significant leading to an already identified concept in the manual ontology. We represented the output of each ranking scheme by plotting the rank position assigned to pairs against their significance. We drew a linear trendline (and

computed its equation) to determine the behavior of the scheme. Considering the highest rank positions the most significant, in the optimal case, the trendline would have an increasing tendency, assigning high ranks to high significance. A high slope of the trendline indicates a good performance of the ranking scheme.

Pair Frequency. Fig. 4 shows the performance of the Pair Frequency ranking scheme. The slope of the linear trendline has a low value (0,0044) denoting a suboptimal behavior. Indeed, several significant pairs still appear on the lowest rank positions and then insignificant pairs predominate until the highest positions. The very top of the ranked set consists indeed of significant pairs. The reason for this behavior is that often very frequent pairs do not reflect functionality while rare pairs do. Often rare pairs interrelate a significant verb (e.g. "add") with a significant noun (e.g. "model"). This prompted us to consider the weights of the constituent terms, as in the next scheme.

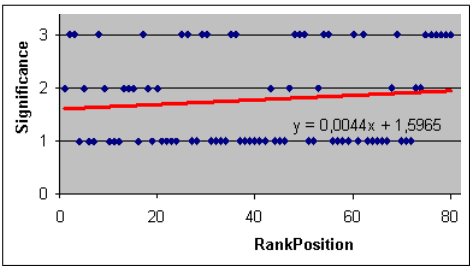

Fig. 4. Pair Frequency Scheme.

Term Weight. Observe in Fig. 5 that, comparatively to the previous scheme, the behavior of the Term Weight scheme is closer to the desired one: the slope of the linear trendline is higher (0,0137). Also, many insignificant pairs are identified for low rank positions and no significant pair is placed on the lowest 20 position (bottom quarter). However, there is a mixed amount of significant and insignificant pairs for a large interval of rank positions. On top positions fewer insignificant and more significant pairs exist.

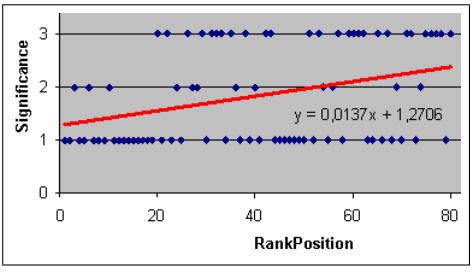

Fig. 5. Term Weight Scheme.

API Relevance. The API Relevance ranking performs slightly better comparatively to the Term Weight method, having a higher slope for the linear trendline (0,0151). The number of shared pairs within APIs is surprisingly low. There is a single pair shared by all three APIs ("add statement"), and three pairs shared by two of the three APIs ("support transaction", "remove statement" , "contain statement"). Note that we only measured the intersection of lexical pairs, however, the intersection of concepts derived from the lexical pairs would yield in a much larger set (since different lexical pairs often denote the same concept).

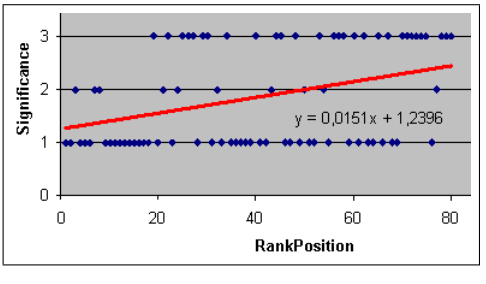

Fig. 6. API Relevance Scheme.

4.5 Discussion

The basic extraction process proved very helpful: it extracted half of the concepts existent in the manually built ontology and also suggested some new additions which were ignored during ontology building. We checked the applicability of the extraction method on a different corpus than the one which served for its design and obtained concepts from all major categories of functionalities. While the extracted concepts are already a valuable help in ontology building, early experiments show that hierarchical information could be derived automatically as well. However, many of the pairs extracted in the second step were insignificant causing considerable time delays when filtering out the significant pairs. Also, we had no insight in the internal working of the process to understand its behavior. Our refinement methods address these two issues.

The evaluation refinement method 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 much less insignificant pairs were extracted in the second step and the loss of ontological information was minimal. The improvements were verified on the original and a new corpus as well.

Our experiments show that ranking schemes based on simple frequency measures can filter some of the significant pairs, making step 3 less time consuming. The Term Weight based scheme performs better than the Pair Ranking scheme, and the API relevance achieves the best performance. We experimented with many more possible schemes, but none of them performed significantly better. We conclude that there is a limit to the performance of simple frequency based methods, however additional background knowledge, specifically of synonymy between terms, could lead to enhanced results. For example knowing which actions (verbs) and data structures (nouns) are synonyms would help determining which lexical pairs are conceptually equivalent and allow us to work with concepts rather than lexical pairs. Our intuition is that synonymy between verbs could be determined from generic knowledge (e.g. WordNet) but more specialized domain knowledge is required for the synonymy of data structures.

5 Related Work

The problem of automating the task of web service semantics acquisition was addressed by the work of two research teams. Heß[6] employs 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 training data as the machine learning techniques do. Patil[11] employ graph similarity techniques to determine a relevant domain for a WSDL file and to annotate the elements of the WSDL file. Currently they determine the semantics of the service parameters and plan to concentrate on functionality semantics in the future. They use existent domain ontologies and acknowledge that their work was hampered by the lack of these ontologies.

There has been several efforts in the ontology learning community to learn concept hierarchies (e.g. [2], [15]), but none of them used sources similar to API documentation. 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 the Formal Concept Analysis theory. As a result, their metrics for filtering the best pairs are different from ours. They define and evaluate three such metrics. Finally, they compare their extracted ontology to a manually built one. We adopted one of their ontology comparison metrics.

6 Conclusions and Future Work

The hypothesis of the presented work is that, given the similarity in functionality between web services and their underlying implementation, functionality focused domain ontologies could be derived from the underlying software's documentation. In this paper we answer two research questions. First, we prove that it is *possible* and *useful* to build domain ontologies from software documentation, by manually building such an ontology from API documentation and using it to describe several web services. Second, we investigate the possibility of automating the ontology extraction process and present a low-investment method for extracting concepts denoting functionality semantics. Even if these concepts represent only part of the available information in the APIs, extracting them is a considerable help for the ontology engineer. We plan to check the applicability of our method in other domains than that of RDF ontology stores.

We plan several enhancements to our extraction method. First, we wish to fine-tune the used linguistic tools (POS tagger) to the grammatical characteristics of our corpus. Second, we want to automate the conceptualization step possibly using synonymy considerations. Finally, we will investigate automatic learning of hierarchical relationships. The visualisation techniques we used suggested several good heuristics in determining such relationships and promise to become intuitive tools for non professional ontology builders.

We are aware of some limitations of our work. First, our corpora are too small for applying statistical techniques. We plan to enlarge them by adding other API documentations or including other sources such as internal code comments, user manuals etc. Also, we consider using extraction methods better suited for small corpora, for example clustering methods. Second, we ignore all existent structural information, therefore loosing a lot of semantics. We plan to build methods which leverage on the structured nature of the javadoc descriptions. For example, terms appearing in the textual description should have a higher weight than those appearing in the parameter section. Also, we wish to take into account the structure of the code (e.g. package/class hierarchies) to determine the main data structures and their relationships.

We conclude that software APIs are rich sources for ontology building and even simple statistical methods are able to extract much of the contained knowledge. The promising results published in this paper prompt us to further pursue our work towards boosting a web of semantic web services.

Acknowledgements. We thank P. Cimiano and J. Tane for their support with the TextToOnto tool and W. van Atteveldt, F. van Harmelen, P. Mika and H. Stuckenschmidt for their review and comments on earlier versions of this paper.

References

1. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In I. Horrocks and J. A. Hendler, editors, *Proceedings of the First International Semantic Web Conference*, 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. C. Fluit, M. Sabou, and F. van Harmelen. *Handbook on Ontologies in Information Systems*, chapter Supporting User Tasks through Visualisation of Light-weight Ontologies. International Handbooks on Information Systems. Springer-Verlag, 2003.
5. A. Gomez Perez. A survey on ontology tools. OntoWeb Delieverable 1.3, 2002.
6. A. Heß and N. Kushmerick. Machine Learning for Annotating Semantic Web Services. In *AAAI Spring Symposium on Semantic Web Services*, March 2004.
7. A. Maedche, B. Motik, and L. Stojanovic. Managing Multiple and Distributed Ontologies in the Semantic Web. *VLDB Journal*, 12(4):286–302, 2003.
8. A. Maedche and S.Staab. Measuring similarity between ontologies. In *Proceedings of EKAW*. Springer, 2002.
9. B. McBride. Jena: A Semantic Web Toolkit. *IEEE Internet Computing*, 6(6):55–59, November/December 2002.
10. E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A Framework and Infrastructure for Semantic Web Services. In *Proceedings of the Second International Semantic Web Conference*, LNCS. Springer-Verlag, 2003.
11. A. Patil, S. Oundhakar, A. Sheth, and K. Verma. METEOR-S Web service Annotation Framework. In *Proceeding of the World Wide Web Conference*, 2004.
12. D. Richards and M. Sabou. Semantic Markup for Semantic Web Tools: A DAML-S description of an RDF-Store. In *Proceedings of the Second International Semantic Web Conference*, LNCS, pages 274–289, Florida, USA, 2003. Springer.
13. M. Sabou. Semi-Automatic Learning of Web Service Ontologies from Software Documentation. Technical report. http://www.cs.vu.nl/ marta/papers/techreport/olws.pdf, 2004.
14. M. Sabou, D. Oberle, and D. Richards. Enhancing Application Servers with Semantics. In *Proceedings of AWESOS Workshop*, Australia, April 2004.
15. C. Sporleder. A Galois Lattice based Approach to Lexical Inheritance Hierarchy Learning. In *Proccedings of the Ontology Learning Workshop, ECAI*, 2002.
16. C. Wroe, C. Goble, M. Greenwood, P. Lord, S. Miles, J. Papay, T. Payne, and L. Moreau. Automating Experiments Using Semantic Data on a Bioinformatics Grid. *IEEE Intelligent Systems*, 19(1):48–55, 2004.
17. C. Wroe, R. Stevens, C. Goble, A. Roberts, and M. Greenwood. A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. *Journal of Cooperative Information Science*, 2003.