# Automatic Tag Identification in Web Service Descriptions

Jean-Rémy Falleri, Zeina Azmeh, Marianne Huchard, Chouki Tibermacine

# AUTOMATIC TAG IDENTIFICATION IN WEB SERVICE DESCRIPTIONS *

Jean-Rémy Falleri, Zeina Azmeh, Marianne Huchard, Chouki Tibermacine
*LIRMM, CNRS and Montpellier II University - 161, rue Ada 34392 Montpellier Cedex 5 France*
*{falleri,azmeh,huchard,tibermacin}@lirmm.fr*

Keywords: Tags, web services, text mining, machine learning.

Abstract: With the increasing interest toward service-oriented architectures, the number of existing Web services is dramatically growing. Therefore, finding a particular service among this huge number of services is becoming a time-consuming task. User tags or keywords have proven to be a useful technique to smooth browsing experience in large document collections. Some service search engines, like Seekda, already propose this kind of facility. Service tagging, which is a fairly tedious and error prone task, is done manually by the providers and the users of the services. In this paper we propose an approach that automatically extracts tags from Web service descriptions. It identifies a set of relevant tags extracted from a service description and leaves only to the users the task of assigning tags not present in this description. The proposed approach is validated on a corpus of 146 services extracted from Seekda.

## 1 INTRODUCTION

Service-oriented architectures (SOA) are achieved by connecting loosely coupled units of functionality. The most common implementation of SOA uses Web Services. One of the main tasks is to find the relevant Web services to use. With the increasing interest toward SOA, the number of existing Web services is dramatically growing. Finding a particular service among this huge amount of services is becoming a time-consuming task.

Web services are usually described with a standard XML-based language called WSDL. A WSDL file includes a documentation part that can be filled with a text indicating to the user what the service does. Unfortunately, this documentation part is often not filled by the creators of the services. In this case, the potential users of the service spend time to understand its functionality and to decide whether or not to use it. A selected service may become unavailable after a period of time, and therefore, a mechanism that may facilitate the discovery of similar services becomes indispensible. *Tagging* is a mechanism that has been introduced in search engines and digital libraries to fulfill exactly this objective.

Tagging is the process of describing a resource by assigning some relevant keywords (tags) to it. The tagging process is usually done manually by the users of the resource to be tagged. Tags are useful when browsing large collections of documents. Indeed, unlike in traditional hierarchical categories, documents can be assigned an unlimited number of tags. It allows cross-browsing between the documents. Seekda[2], one of the main service search engines, already allows its users to tag its indexed services. Tags are also useful to have a quick understanding of a particular service and service classification or clustering.

In this paper, we present an approach that automatically extracts a set of relevant tags from a WSDL. We use a corpus of user-tagged services to learn how to extract relevant tags from untagged service descriptions. Our approach relies on text mining techniques in order to extract candidate tags out of a description, and machine learning techniques to select relevant tags among these candidates. We have validated this approach on a corpus of 146 user-tagged Web services extracted from Seekda. Results show that this approach is significantly more efficient than the traditional (but fairly efficient) *tfidf* weight.

The remaining of the paper is organized as follows. Section 2 introduces the context of our work. Then, Section 3 details our tag extraction process. Section 4 presents a validation of this process and discusses the obtained results. Before concluding and presenting the future work, we describe the related work in Section 5.

*France Télécom R&D has partially supported this work (contract CPRE 5326).

[2] http://www.seekda.com

## 2 CONTEXT OF THE WORK

Our work focuses on extracting tags from service descriptions. In the literature, we found a similar problem: *keyphrase extraction*, which aims at extracting important and relevant short phrases from a plain-text document. It is mostly used on journal articles or on scientific papers in order to smooth browsing and indexing of those documents in digital libraries. Before starting our work, we analyzed one assessed approach that performs keyphrase extraction: Kea (Frank et al., 1999) (Section 2.1). We concluded that a straightforward application of this approach is not possible on service descriptions instead of plain-text documents (Section 2.2).

### 2.1 Description of Kea

Kea (Frank et al., 1999) is a keyphrase extractor for plain-text documents. It uses a Bayesian classification approach. Kea has been validated on several corpora (Jones and Paynter, 2001; Jones and Paynter, 2002) and has proven to be an efficient approach. It takes a plain-text document as input. From this text, it extracts a list of candidate keyphrases. These candidates are the $\bigcup_{i=1}^{k}$ $k$-grams of the text. For instance, let us consider the following sample document: "*I am a sample document*". The candidate keyphrases extracted if $k = 2$ are: *(I,am,a,sample,document,I am,am a,a sample,sample document)*. To choose the most adapted value of $k$ for the particular task of extracting tags from WSDL files, we made some measurements and found that 86% of the tags are of length 1. It clearly shows that one word tags are assigned in the vast majority of the cases. Therefore we will fix $k = 1$ in our approach (meaning that we are going to find one word length tags). Nevertheless, our approach, like Kea, is easily generalizable to extract tags of length $k$.

Kea then computes two features on every candidate keyphrase. First, *distance* is computed, which is the number of words that precede the first observation of the candidate divided by the total number of words of the document. For instance, for the sample document, $distance(am\ a) = \frac{1}{5}$. Second, *tfidf*, a standard weight in the information retrieval field, is computed. It measures how much a given candidate keyphrase of a document is specific to this document. More formally, for a candidate $c$ in a document $d$, $tfidf(c,d) = tf(c,d) \times idf(c)$. The metric $tf(c,d)$ (*term frequency*) corresponds to the frequency of the term $c$ in $d$. It is computed with the following formula: $tf(c,d) = \frac{occurences\ of\ c\ in\ d}{size\ of\ d}$. The metric $idf(c)$ (*inverse document frequency*) measures the general importance of the term in a corpus $\mathcal{D}$. $idf(c) = log(\frac{|\mathcal{D}|}{|\{d:\ c\ \in d\}|})$.

Kea uses a naive Bayes classifier to classify the different candidate keyphrases using the two previously described features. The authors showed that this type of classifier is optimal (Domingos and Pazzani, 1997) for this kind of classification problem. The two classes in which the candidate keyphrases are classified are: *keyphrase* and *not keyphrase*. Several evaluations on real world data report that Kea achieves good results (Jones and Paynter, 2001; Jones and Paynter, 2002). In the next section, we describe how WSDL files are structured and highlight why the Kea approach is not directly applicable on them.

### 2.2 WSDL service descriptions

We extract tags from the following WSDL elements: *services*, *ports*, *port types*, *bindings*, *types* and *messages*. Each element has an identifier, which can optionally come with a plain-text documentation. Figure 3 (left) shows the general outline of a WSDL file.

One simple idea to extract tags from services would be to use Kea on their plain-text documentations. Unfortunately, an analysis of our service corpus shows that about 38% of the services are not documented at all. Another important source of information to discover tags are the identifiers contained in the WSDL. For instance *weather* would surely be an interesting tag for a service named *WeatherService*. Unfortunately, identifiers are not easy to work with. Firstly because identifiers are usually a concatenation of different words. Secondly because they are associated with different kinds of elements (services, ports, types, ...) that have not the same importance in a service description. Therefore, extracting candidate tags from WSDL files is not straightforward. Several preprocessing and text-mining techniques are required. Moreover, the previously described features (*tfidf* and *distance*) are not easy to adapt on words from WSDL files. First because WSDL deals with two categories of words (the documentation and the identifiers) that are not necessary related. Second because the *distance* feature is meaningless on the identifiers, which are defined in an arbitrary order.

## 3 TAG EXTRACTION PROCESS

Similarly to Kea, we model the tag extraction problem as the following classification problem: classifying a word into one of the two *tag* and *no tag* classes. Our overall process is divided into two phases: a *training* phase and a *tag extraction* phase.

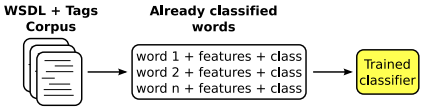


Figure 1: The training phase.

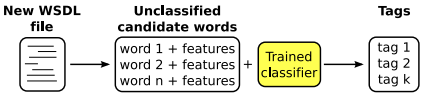


Figure 2: The tag extraction phase

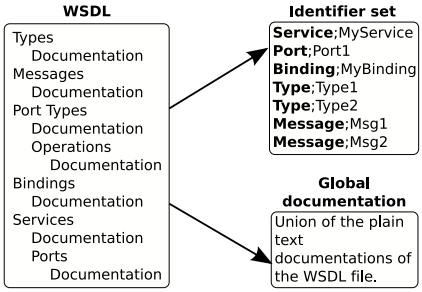


Figure 3: WSDL pre-processing

Figure 1 summarizes the behavior of the training phase. In this phase we have a corpus of WSDL files and associated tags, extracted from Seekda, (Section 3.1). From this training corpus, we first extract a list of candidate words by using text-mining techniques, (Sections 3.2 and 3.3). Then several *features* (metrics) are computed on every candidate. A *feature* is a common term in the machine learning field. As an example, it may be the frequency of the words in their WSDL file. Finally, since manual tags are assigned to those WSDL files, we use them to classify the candidate words coming from our WSDL files. Using this set of candidate words, computed features and assigned classes, we train a classifier.

Figure 2 describes the *tag extraction phase*. First, like in the *training phase*, a list of candidate words is extracted from an untagged WSDL file and the same features are computed. The only difference with the training phase is that we do not know in advance which of those candidates are true tags. Therefore we use the previously trained classifier to automatically perform this classification. Finally the tags extracted from the WSDL file are the words that have been classified in the *tag* class.

## 3.1 Creation of the training corpus

As explained above, our approach requires a training corpus, denoted by $\mathcal{T}$. Since we want to extract tags from WSDL files, $\mathcal{T}$ has to be a set of couples $(wsdl, tags)$, with *wsdl* a WSDL file, and *tags* a set of corresponding manually assigned tags. We created a corpus using Seekda. Indeed, Seekda allows its users to manually assign tags to its indexed services. We created a program that parses the Seedka result pages to extract WSDL files together with their user tags. To ensure that the services of our corpus were significantly tagged, we only retain the WSDL files that have at least five tags. Using this program, we extracted 150 WSDL files. Then, we removed from $\mathcal{T}$ the WSDL files that triggered parsing errors. Finally, we have a training corpus containing 146 WSDL files together with their associated tags.

To clean the tags of the training corpus, we performed three operations. We removed the non alpha numeric characters from the tags (we found several tags like *_onsale* or *:finance*). We also removed a meaningless and highly frequent tag (the *_unkown* tag). Finally, we divided the tags with length $n > 1$ into $n$ tags of length 1, in order to have only tags of length 1 (the reason has been explained in section 2.1). The length of a tag is defined as the number of words composing this tag.

Finally, we have a corpus of 146 WSDL files and 1393 tags (average of 9.54 tags per WSDL file). An analysis of $\mathcal{T}$ shows that about 35% of the user tags are already contained in the WSDL files.

## 3.2 Pre-processing of the WSDL files

As we have seen before, a WSDL file contains several element definitions optionally containing a plain-text documentation. The left side of figure 3 shows such a data structure. In order to simplify the WSDL XML representation in a format more suitable to apply text mining techniques, we decided to extract two documents from a WSDL description. First, a set of couples $(type, ident)$ representing the different elements defined in the WSDL. We have $type \in$ *(Service,Port,PortType,Message,Type,Binding)* the type of the element and *ident* the identifier of the element. We call this set of couples the *identifier set*. Second, a plain text containing the union of the plain-text documentations found in the WSDL file, called the *global documentation*. This pre-processing operation is summarized in the figure 3.

## 3.3 Selection of the candidate tags

As seen in the previous section, we have now two different sources of information for a given WSDL: an *identifier set* and a *global documentation*. Unfortunately, those data are not yet usable to compute meaningful metrics. Firstly because the identifiers are

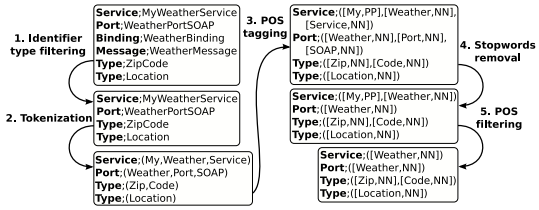


Figure 4: Processing of the identifiers

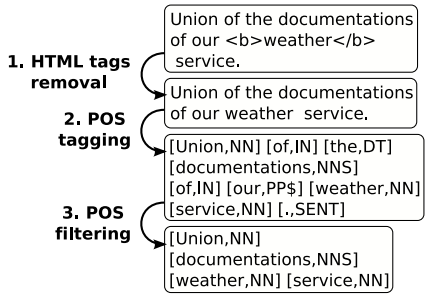


Figure 5: Processing of the global documentation

names of the form *MyWeatherService*, and therefore are very unlikely to be tags. Secondly because these data contain a lot of obvious useless tags (like the *you* pronoun). Therefore, we will now apply several text-mining techniques on the identifier set and the global documentation.

Figure 4 shows how we process the *identifier set*. Here is the complete description of all the performed steps:

1. **Identifier type filtering**: during this step, the couples *(type,ident)* where $type \in$ *(PortType,Message,Binding)* are discarded. We applied this filtering because very often, the identifiers of the elements in those categories are duplicated from the identifiers in the other categories.
2. **Tokenization**: during this step, each couple $(type, ident)$ is replaced by a couple $(type, tokens)$. *tokens* is the set of words appearing in *ident*. For instance, *(Service,MyWeatherService)* would be replaced by *(Service,[My,Weather,Service])*. To split *ident* into several tokens, we created a tokenizer that uses common clues in software engineering to split the words. Those clues are for instance a case change, or the presence of a non alpha-numeric character.
3. **POS tagging**: during this step each couple $(type, tokens)$ previously computed is replaced by a couple $(type, ptokens)$. *ptokens* is a set of couples $(token_i, pos_i)$ derived from *tokens* where $token_i$ is a token from *tokens* and $pos_i$ the part-of-speech corresponding to this token. We used the tool *tree tagger* (Schmid, 1994) to compute those part-of-speeches. Example: *(Service,[My,Weather,Service])* is replaced by *(Service,[(My,PP),(Weather,NN),(Service,NN)])*. *NN* means noun and *PP* means pronoun.
4. **Stopwords removal**: during this step, we process each couple $(type, ptokens)$ and remove from *ptokens* the elements $(token_i, pos_i)$ where $token_i$ is a *stopword* for *type*. A stopword is a word too frequent to be meaningful. We manually established a stopword list for each identifier type. Example: *(Service,[(My,PP),(Weather,NN),(Service,NN)])* is replaced by *(Service,[(My,PP)(Weather,NN)])* because *Service* is a stopword for service identifiers.
5. **POS filtering**: during this step, we process each couple $(type, ptokens)$ and remove from *ptokens* the elements $(token_i, pos_i)$ where $pos_i \notin$ *(Noun,Adjective,Verb,Symbol)*. Example: *(Service,[(My,PP),(Weather,NN))* is replaced by *(Service,[(Weather,NN)])* because pronouns are filtered.

Figure 5 shows how we process the *global documentation*. Here is the complete description of all the performed steps:

1. **HTML tags removal**: the HTML tags (words begining by < and ending by >) are removed from the global documentation.
2. **POS tagging**: similar to the POS tagging step applied to the identifier set.
3. **POS filtering**: similar to the POS filtering step applied to the identifier set.

The union of the remaining words in the identifier set and in the global documentation are our candidate tags. When defining those processing operations, we took great care that no correct candidate tags (i.e. a candidate tag that is a real tag) of the training corpus have been discarded. The next section describes how we adapted the Kea features to these candidate tags.

## 3.4 Computation of the features

We have now different well separated words. Therefore we can now compute the $tfidf$ feature. But words appearing in documentation or in the identifier names are not the same. We decided (mostly because it turns out to perform better) to separate the $tfidf$ value into a $tfidf_{ident}$ and a $tfidf_{doc}$ which are respectively the $tfidf$ value of a word over the identifier set and over the global documentation. Like in

| Word | $TFIDF_{id}$ | $TFIDF_{doc}$ | *IN_SERVICE* | ... | *IN_DOC* | *POS* | *IS_TAG* |
|---|---|---|---|---|---|---|---|
| Weather | $[0, 0.01]$ | $]0.01, 0.04]$ | × | | | *NN* | × |
| Location | $]0.03, 0.1]$ | $]0.04, 0.15]$ | | | × | *JJ* | |
| Code | $]0.03, 0.1]$ | $]0.01, 0.04]$ | | | × | *VV* | |

Table 1: Excerpt of the ARFF file enriched with the words

Kea, we used the method in (Fayyad and Irani, 1993) to discretize those two real-valued features.

The *distance* feature still has no meaning over the identifier set, because the elements of a WSDL description are given in an arbitrary order. Therefore we decided to adapt it by defining five different features: *in_service*, *in_port*, *in_type*, *in_operation* and *in_documentation*. Those features take their values in the $(true, false)$ set. A *true* value indicates that the word has been seen in an element identifier of the corresponding type. For instance *in_service(weather)* = *true* means that the word *weather* has been seen in a service identifier. *in_documentation(weather)* = *true* means that the word *weather* has been seen in the global documentation.

In addition of these features, we compute another feature called *pos*, not used in Kea, which significantly improves the results. *pos* is simply the part-of-speech that has been assigned to the word during the POS tagging step. If several parts-of-speech have been assigned to the same word, we choose the one that has been assigned in the majority of the cases. The different values of *pos* are: *NN (noun), NNS (plural noun), NP (proper noun), NPS (plural proper noun), JJ (adjective), JJS (plural adjective), VV (verb), VVG (gerundive verb), VVD (preterit verb), SYM (symbol)*.

## 3.5 Training and using the classifier

We applied the previously described technique to all the WSDL files of $\mathcal{T}$. In addition to the previously described features, we compute the *is_tag* feature over the candidates. This feature takes its values in the $(true, false)$ set. $is\_tag(word) = true$ means that *word* has been assigned as a tag by Seekda users for its service description. We have serialized all those results in an *ARFF* file compatible with the Weka tool (Witten and Frank, 1999). Weka is a machine learning tool that defines a standard format for describing a training corpus and provides the implementation of many classifiers. One can use Weka in order to train a classifier or compare the performances of different classifiers regarding a given classification problem. Table 1 shows an excerpt of the ARFF file we produce, enriched with the words for the sake of clarity.

With this ARFF file, we used Weka to train a naive Bayes classifier, shown as optimal for our kind of classification task (Domingos and Pazzani, 1997). This trained classifier can now be used in the tag extraction phase. As previously said, the beginning of this phase is the same as the one of the training phase. It means that the WSDL file goes through the previously described operations (pre-processing, candidates selection and features computation). Only this time, the value of the *is_tag* feature is not available. This value will be automatically computed by the previously trained classifier.

# 4 VALIDATION OF THE PROPOSED WORK

This section provides a validation of our technique on real world data from Seekda to assess the precision and recall of our trained classifier.

**Methodology:** We conducted two different experiments. In the first one, the trained classifier is applied on the training corpus $\mathcal{T}$ and its output is compared with the tags given by Seekda users (obtained as described in Section 3.1). After having conducted the first experiment, a manual assessment of the tags produced by our approach revealed that many tags not assigned by the user seemed highly relevant. This phenomenon has also been observed in several human evaluations of Kea (Jones and Paynter, 2001; Jones and Paynter, 2002), that inspired our approach. It occurs because tags assigned by the users are not the *absolute truth*. Indeed, it is very likely that users have forgotten many relevant tags, even if they were in the service description. To show that the real efficiency of our approach is better than the one computed in the first experiment, we perform a second experiment, where we manually augmented the user tags of our corpus with additional tags we found relevant and accurate by analyzing the WSDL descriptions of the services.

**Metrics:** In the evaluation, we used precision and recall. First, for each web service $s \in \mathcal{T}$, where $\mathcal{T}$ is our training corpus, we consider: $A$ the set of tags produced by the trained classifier, $M$ the set

of the tags given by Seekda users and $W$ the set of words appearing in the WSDL. Let $I = A \cap M$ be the set of tags assigned by our classifier and Seekda users. Let $E = M \cap W$ be the set of tags assigned by Seekda users present in the WSDL file. Then we define $precision(s) = \frac{|I|}{|A|}$ and $recall(s) = \frac{|I|}{|E|}$, which are aggregated in $precision(\mathcal{T}) = \frac{\sum_{s \in \mathcal{T}} precision(s)}{|\mathcal{T}|}$ and $recall(\mathcal{T}) = \frac{\sum_{s \in \mathcal{T}} recall(s)}{|\mathcal{T}|}$. The recall is therefore computed over the tags assigned by Seekda users that are present in the descriptions of concerned services.

**Evaluation:** Figure 6 (left) gives results for the first experiment where the output of the classifier is compared with the tags of Seekda users, while in Figure 6 (right), enriched tags of Seekda users are used in the comparison (curated corpus). In this figure, our approach is called *ate* (*Automatic Tag Extraction*). To clearly show the concrete benefits of our approach, we decided to include in these experiments a straightforward (but fairly efficient) technique. This technique, called *tfidf* in Figure 6, consists in selecting, after the application of our text-mining techniques, the five candidate tags with the highest *tfidf* weight.

In Figure 6 (left), the precision of *ate* is 0.48. It is a significant improvement compared to the *tfidf* method that achieves only a precision of 0.28. Moreover, there is no significant difference between the recall achieved by the two methods. To show that the precision and recall achieved by *ate* are not biased by the fact that we used the training corpus as a testing corpus, we performed a 10 folds cross-validation. In a 10 folds cross-validation, our training corpus is divided in 10 parts. One is used to train a classifier, and the 9 other parts are used to test this classifier. This operation is done for every part, and then, the average recall and precision are computed. The results achieved by our approach using cross-validation ($precision = 0.44$ and $recall = 0.42$) are very similar to those obtained in the first experiment.

In Figure 6 (right), we see that the precision achieved by *ate* in the second experiment is much better. It reaches 0.8, while the precision achieved by the *tfidf* method increases to 0.41. The recall achieved by the two methods remains similar. The precision achieved by our method in this experiment is good. Only 20% of the tags discovered by *ate* are not correct. Moreover, the efficiency of *ate* is significantly higher than *tfidf*.

**Threats to validity:** Our experiments use real world services, obtained from the Seekda service search engine. Our training corpus contains services

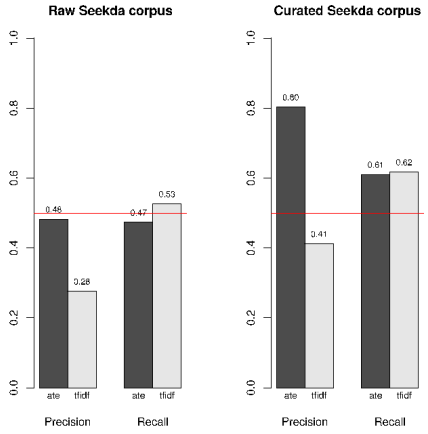


Figure 6: Results on the original and manually curated Seekda corpus

extracted randomly with the constraint that they contain at least 5 user tags. We assumed that Seekda users assign correct tags. Indeed, our method admits some noise but would not work if the majority of the user tags were poorly assigned. In the second experiment, we manually added tags we found relevant by examining the complete description and documentation of the concerned services. Unfortunately, since we are not "real" users of those services, some of the tags we added might not be relevant.

# 5 RELATED WORK

In this section, we will present the related work according to two fields of research: keyphrase extraction and web service discovery.

**Keyphrase extraction and assignment** According to (Turney, 2003), there are two general approaches that are able to supply keyphrases for a document: *keyphrase extraction* and *keyphrase assignment*. Both approaches are using supervised machine learning approaches, with training examples being documents with manually supplied keyphrases.

In the *keyphrase assignment* approach, a list of predefined keyphrases is treated as a list of classes in which the different documents are classified. Text categorization techniques are used to learn models for assigning a class (*keyphrase*) to a document. Two main approaches of this category are (Dumais et al., 1998; Leung and Kan, 1997).

In the *keyphrase extraction* approach, a list of candidate keyphrases are extracted from a document and classified into the classes *keyphrase* and *not keyphrase*. There are two main approaches that fall

in this category: one using a genetic algorithm (Turney, 2000) and one using a naive Bayes classifier (Kea (Frank et al., 1999)).

**Web service discovery** Web service discovery is a wide research area with many underlying issues and challenges. A quick overview of some of the works can be acquired from (Brockmans et al., 2008; Lausen and Steinmetz, 2008)[3]. Here, we describe a selection of works, classified using their adapted techniques.

Many approaches adapt techniques from machine learning field, in order to discover and group similar services. In (Crasso et al., 2008a; Heßand Kushmerick, 2003), service classifiers are defined depending on sets of previously categorized services. Then the resulting classifiers are used to deduce the relevant categories for new given services. In case there were no predefined categories, unsupervised clustering is used. In (Ma et al., 2008), CPLSA approach is defined that reduces a service set then clusters it into semantically related groups.

In (Lu and Yu, 2007), a web service broker is designed relying on approximate signature matching using XML schema matching. It can recommend services to programmers in order to compose them. In (Günay and Yolum, 2007), a service request and a service are represented as two finite state machines then they are compared using various heuristics to find structural similarities between them. In (Dong et al., 2004), the Woogle web service search engine is presented, which takes the needed operation as input and searches for all the services that include an operation similar to the requested one. In (Bouillet et al., 2008), tags coming from folksonomies are used to discover and compose services.

The vector space model is used for service retrieval in several existing works as in (Platzer and Dustdar, 2005; Wang and Stroulia, 2003; Crasso et al., 2008b). Terms are extracted from every WSDL file and the vectors are built for each service. A query vector is also built, and similarity is calculated between the service vectors and the query vector. This model is sometimes enhanced by using WordNet, structure matching algorithms to ameliorate the similarity scores as in (Wang and Stroulia, 2003), or by partitioning the space into subspaces to reduce the searching space as in (Crasso et al., 2008b).

A collection of works (Aversano et al., 2006; Peng et al., 2005; Azmeh et al., 2008), adapt the formal concept analysis method to retrieve web services more efficiently. Contexts obtained from service descriptions are used to classify the services as a concept lattice. This lattice helps in understanding the different relationships between the services, and in discovering service substitutes.

[3]The second one is edited by the responsible of Seekda's technical infrastructure

# 6 CONCLUSION AND FUTURE WORK

With the emergence of SOA, it becomes important for developers using this paradigm to retrieve Web services matching their requirements in an efficient way. By using Web service search engines, these developers can either search by keywords or navigate by tags. In the second case, it is necessary that the tags characterize accurately this service. Our work contributes in this direction and introduces a novel approach that extracts tags from Web service descriptions. This approach combines and adapts text mining as well as machine learning techniques. It has been experimented on a corpus of user-tagged real world Web services. The obtained results demonstrated the efficiency of our automatic tag extraction process. The proposed work is useful for many purposes. First, the automatically extracted tags can assist the users who are tagging a given service, or to "bootstrap" tags on untagged services. They are also useful to have a quick understanding of a service without reading the whole description. They can also be used in tasks such as service clustering (for instance by measuring the similarity of the tags of two given services), or classification (for instance by defining association rules between tags and categories).

With our approach, only tags appearing in the WSDL files are to be found. This way, we miss some interesting tags (such as associated words, synonyms or more general words). Nevertheless, the identified tags represent a good support to find other relevant tags by using ontological resources (like *WordNet*), or machine learning techniques. This is one of the perspectives of our work. We also plan to work on the extraction of tags composed of more than one word. Indeed, one-word tags are sometimes insufficient to describe concepts like *exchange rate* or *Web 2.0*.

# REFERENCES

Aversano, L., Bruno, M., Canfora, G., Penta, M. D., and Distante, D. (2006). Using concept lattices to support service selection. *International Journal of Web Services Research*, 3(4):32–51.

Azmeh, Z., Huchard, M., Tibermacine, C., Urtado, C., and Vauttier, S. (2008). WSPAB: A tool for

automatic classification & selection of web services using formal concept analysis. In *Proc. of (ECOWS 2008)*, pages 31–40, Dublin, Ireland. IEEE Computer Society.

Bouillet, E., Feblowitz, M., Feng, H., Liu, Z., Ranganathan, A., and Riabov, A. (2008). A folksonomy-based model of web services for discovery and automatic composition. In *IEEE International Conference on Services Computing (SCC)*, pages 389–396.

Brockmans, S., Erdmann, M., and Schoch, W. (2008). Service-finder deliverable d4.1. research report about current state of the art of matchmaking algorithms. Technical report, Ontoprise, Germany.

Crasso, M., Zunino, A., and Campo, M. (2008a). Awsc: An approach to web service classification based on machine learning techniques. *Inteligencia Artificial, Revista Iberoamericana de Interligencia Artificial*, 12, No 37:25–36.

Crasso, M., Zunino, A., and Campo, M. (2008b). Query by example for web services. In *SAC '08: Proc. of the 2008 ACM symposium on Applied computing*, pages 2376–2380.

Domingos, P. and Pazzani, M. J. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130.

Dong, X., Halevy, A., Madhavan, J., Nemes, E., and Zhang, J. (2004). Similarity search for web services. In *VLDB '04: Proc. of the Thirtieth international conference on Very large data bases*, pages 372–383.

Dumais, S. T., Platt, J. C., Hecherman, D., and Sahami, M. (1998). Inductive learning algorithms and representations for text categorization. In *CIKM*, pages 148–155. ACM.

Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI*, pages 1022–1029.

Frank, E., Paynter, G. W., Witten, I. H., Gutwin, C., and Nevill-Manning, C. G. (1999). Domain-specific keyphrase extraction. In *IJCAI*, pages 668–673.

Günay, A. and Yolum, P. (2007). Structural and semantic similarity metrics for web service matchmaking. In *EC-Web*, pages 129–138.

Heß, A. and Kushmerick, N. (2003). Learning to attach semantic metadata to web services. In *International Semantic Web Conference*, pages 258–273.

Jones, S. and Paynter, G. W. (2001). Human evaluation of kea, an automatic keyphrasing system. In *JCDL*, pages 148–156. ACM.

Jones, S. and Paynter, G. W. (2002). Automatic extraction of document keyphrases for use in digital libraries: Evaluation and applications. *JASIST*, 53(8):653–677.

Lausen, H. and Steinmetz, N. (2008). Survey of current means to discover web services. Technical report, Semantic Technology Institute (STI).

Leung, C.-H. and Kan, W.-K. (1997). A statistical learning approach to automatic indexing of controlled index terms. *JASIS*, 48(1):55–66.

Lu, J. and Yu, Y. (2007). Web service search: Who, when, what, and how. In *WISE Workshops*, pages 284–295.

Ma, J., Zhang, Y., and He, J. (2008). Efficiently finding web services using a clustering semantic approach. In *CSSSIA '08: Proc. of the 2008 international workshop on Context enabled source and service selection, integration and adaptation*, pages 1–8.

Peng, D., Huang, S., Wang, X., and Zhou, A. (2005). Management and retrieval of web services based on formal concept analysis. In *Proc. of the The Fifth International Conference on Computer and Information Technology (CIT'05)*, pages 269–275.

Platzer, C. and Dustdar, S. (2005). A vector space search engine for web services. In *Third IEEE European Conference on Web Services, 2005. ECOWS 2005.*, pages 62–71.

Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. In *Proc. of NeMLaP'94*, volume 12. Pages 44–49.

Turney, P. D. (2000). Learning algorithms for keyphrase extraction. *Inf. Retr.*, 2(4):303–336.

Turney, P. D. (2003). Coherent keyphrase extraction via web mining. In *IJCAI*, pages 434–442.

Wang, Y. and Stroulia, E. (2003). Semantic structure matching for assessing web service similarity. In *1st International Conference on Service Oriented Computing (ICSOC03*, pages 194–207. Springer-Verlag.

Witten, I. H. and Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*.