Cross-Lingual Web API Classification and Annotation

Maria Maleshkova, Lukas Zilka, Petr Knoth, Carlos Pedrinaci

Knowledge Media Institute (KMi)
The Open University, Milton Keynes, United Kingdom
{m.maleshkova, l.zilka, p.knoth, c.pedrinaci}@open.ac.uk

Abstract. Recent developments on the Web are marked by the growing support for the Linked Data initiative, which encourages government and public organisations, as well as private institutions, to expose their data on the Web. This results in a plentitude of multi-lingual document collections where the original resources are published in the language, in which they are available. The challenges of multilingualism present on the Semantic Web are also reflected in the context of services on the Web, characterised by the rapid increase in popularity and use of Web APIs, as indicated by the growing number of available APIs and the applications built on top of them. Web APIs are commonly described in plain-text as part of Web pages, following no particular guidelines and conforming to no standards, despite some initial approaches in the area [1, 2]. Therefore, API providers publish descriptions in any language they see fit, making the service discovery and the subsequent processing of the documentation challenging tasks. In this paper, we present a cross-lingual approach that calculates semantic similarity of text to help classify and annotate Web APIs, based on their textual descriptions. Furthermore, we show how our solution can be implemented as part of SWEET [3], which is a tool that enables the semi-automated creation of semantic Web API descriptions. In addition, we demonstrate how the cross-lingual approach can be adopted to support the language-independent discovery of Web APIs.

1 Introduction

In the research context, English has established itself as a de-facto standard language for conducting and publishing work. It is therefore easy to forget that multilingualism is actually one of the main characteristics of the Semantic Web. The importance of language diversity is made evident by the growing support for the Linked Data initiative, which encourages government and public organisations, as well as private institutions, to expose their data on the Web. Since the document collections are published in the language, in which the original sources are available, the result is an abundance of multi-lingual resources. In comparison, the situation is quite similar in the context of services on the Web, where the past few years have been marked by the increasing popularity and use of Web APIs. The growing importance of Web APIs, also referred to as RESTful

services [4] (especially when conforming to the REST [5] architectural principles) was initially triggered by popular Web 2.0 applications like Facebook, Google, Flickr and Twitter that offer easy-to-use, publicly available APIs as means for accessing their resources. Currently, Web APIs not only enable retrieval and manipulation of different resources but also facilitate building of versatile applications based on combining heterogeneous data coming from diverse services.

Despite their proliferation, Web APIs are facing a number of limitations. The majority of the Web APIs have only textual descriptions that are given directly as part of HTML Web pages, disregarding efforts towards a common formal language for describing Web APIs [1, 2]. Providers publish the documentation in any form and any language that they see fit and as a result, finding and using Web APIs can be quite challenging and requires extensive manual effort. API consumers need to search for suitable services, manually process and interpret the available documentation, which is sometimes in a different language, and produce custom implementation solutions that are rarely reusable.

In this paper, we focus in particular on supporting the Web API search and discovery tasks by enhancing the descriptions with: 1) information about the type of provided functionality (for example, a weather service or a shopping service) and 2) central concepts that can be used for determining the domain of the service or be taken directly for annotating service properties such as the inputs and outputs. For this purpose we present an approach that makes use of Cross-lingual Explicit Semantic Analysis [6] to classify and annotate APIs, given their textual description. As a result we are able to discover APIs with a particular functionality, or characterised by a set of keywords, across languages. Moreover, by including the computed classification and annotation details as part of the semantic Web API descriptions, we support service discovery as well as directly contribute to a Semantic Web that integrates Web APIs with multilingual documentation. We also validate the applicability of the devised approach by introducing a design of a system capable of supporting the creation of semantic Web API descriptions, enhanced with classification information and further annotations, and describe the implementation of its key components.

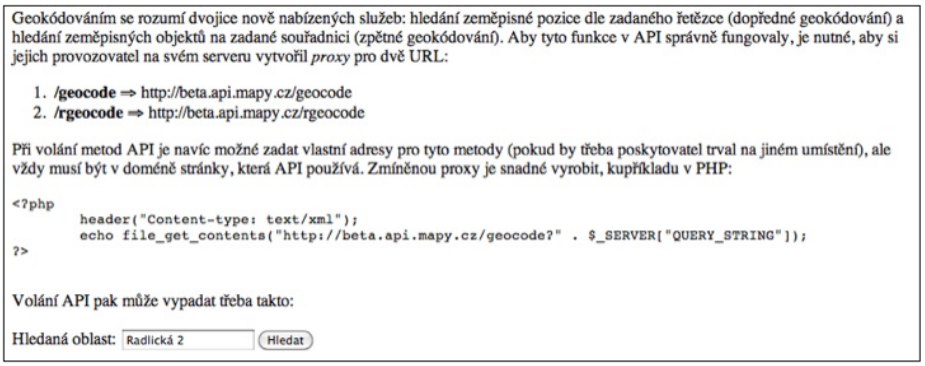
The remainder of this paper is structured as follows: Section 2 provides a motivating example that illustrates the challenges of searching for APIs with particular functionality or from a particular domain, while Section 3 lists related work and gives some background in the area of semantic Web API descriptions and details on the cross-lingual semantic relatedness approach. Our API classification and annotation approaches are given in Section 4. Section 5 describes in more detail the solution design and the implementation of the key components, and Section 6 concludes the paper.

2 Motivation

One of the most common service discovery tasks is discovery based on the functionality or the domain of the service (for example, “I am looking for an API that can map my travel route” or “I am looking for a shopping service”). Therefore,

in this paper we focus our work on supporting this basic but essential discovery type. Currently the search options for APIs are very limited. One possibility is to use conventional search engines such as Google or Yahoo and do keyword search and hope that one of the returned matches is a Web API description. It is important to point out that so far there is no way of automatically distinguishing between webpages that describe Web APIs and webpages that simply mention an API, such as a news article, so this differentiation has to be done manually. Another way is searching in Web API directories, such as ProgrammableWeb (`http://www.programmableweb.com`), which are based on manually collecting and registering APIs. A final option is looking in developer forums and asking other users for suitable APIs, i.e. the "word of mouth" approach.

Figure 1 visualises a simple example, which demonstrates the necessity of supporting cross-language Web API search. The presented API provides capabilities for geocoding and reverse geocoding. If we use Google to search for a geocoding API, the query will be language specific; therefore, we would either find a service such as the popular GeoNames (`http://www.geonames.org/export/web-services.html`), which is in English, or the example description in Czech (`http://ondras.zarovi.cz/smap/geokodovani/`). However, it would not be possible to find both descriptions with one and the same search keywords. Similarly, existing Web API directories are language specific, in particular restricted to English, as are developer sites and forums too.



Geokódováním se rozumí dvojice nově nabízených služeb: hledání zeměpisné pozice dle zadaného řetězce (dopředné geokódování) a hledání zeměpisných objektů na zadané souřadnici (zpětné geokódování). Aby tyto funkce v API správně fungovaly, je nutné, aby si jejich provozovatel na svém serveru vytvořil *proxy* pro dvě URL:

1. **/geocode** ⇒ http://beta.api.mapy.cz/geocode
2. **/rgeocode** ⇒ http://beta.api.mapy.cz/rgeocode

Při volání metod API je navíc možné zadat vlastní adresy pro tyto metody (pokud by třeba poskytovatel trval na jiném umístění), ale vždy musí být v doméně stránky, která API používá. Zmíněnou proxy je snadné vyrobit, kupříkladu v PHP:

```
<?php
        header("Content-type: text/xml");
        echo file_get_contents("http://beta.api.mapy.cz/geocode?" . $_SERVER["QUERY_STRING"]);
?>
```

Volání API pak může vypadat třeba takto:

Hledaná oblast: Radlická 2 Hledat

Fig. 1. Example Web API Description in Czech

In summary, even though there are at least two geocoding Web APIs, given the existing search possibilities, we would find either one or the other, depending on which language we use to conduct the search. Therefore, we propose to employ a cross-language classification approach and to enhance the API descriptions with metadata about the service functionality. Furthermore, we propose to determine the key concepts, characterising the textual documentation, and to use those directly as tags or even use them to determine the domain of the service and specific annotations for individual service properties, such as inputs and outputs. In particular, we follow a lightweight semantic approach for enhancing existing API description with metadata, which supports the completion of tasks such as discovery, but also composition and invocation, on the level of semantics,

abstracting away from syntactic specifics, including the original language of the documentation [3, 7]. We provide more detail to the proposed approach in the following sections.

3 Background and Related Work

In this section we provide some background on the use of lightweight semantics for describing Web APIs, list existing annotation and tagging tools, and focus on providing details on common classification approaches and, in particular, on classification based on cross-lingual semantic relatedness.

3.1 Lightweight Semantic Web API Descriptions

Since the advent of Web service technologies, research on semantic Web services (SWS) has been devoted to reduce the extensive manual effort required for manipulating Web services. The main idea behind this research is that tasks such as discovery, negotiation, composition and invocation can have a higher level of automation, when services are enhanced with semantic descriptions of their properties. Similarly to "classical" Web services based on WSDL/SOAP, Web API-related tasks also require a lot of developer involvement and face even further difficulties, since there is no established common formalism for describing Web APIs. In order to address this, lightweight annotations over API descriptions have been proposed as means for achieving a higher-level of automation.

Currently, there are two main contributions aiming at using semantics to support the automation of common Web API service-related tasks. Both approaches rely on marking service properties within the HTML description and subsequently linking these to semantic entities. MicroWSMO [7] is a formalism for the semantic description of Web APIs, which is based on adapting the SAWSDL [8] approach for enhancing service properties with semantic information. MicroWSMO uses microformats for adding semantic information on top of HTML service documentation, by relying on hRESTS [9] for marking service properties. Another formalism is SA-REST [10], which also applies the grounding principles of SAWSDL but instead of using hRESTS relies on RDFa [11] for marking service properties. Similarly to MicroWSMO, SA-REST enables the annotation of existing HTML service descriptions by identifying service elements and linking these to semantic entities. The main differences between the two approaches are not the underlying principles but rather the implementation techniques. For the here presented work, we have adopted hRESTS and MicroWSMO that are already implemented as part of SWEET [3], which is a tool that enables the semi-automated creation of semantic Web API descriptions.

Currently, there are quite a few tagging tools that enable the tagging of web pages but also support the user in choosing the correct tags. Some of the main ones include TagAssist [12], collaborative tagging [13] and user-based collaborative tagging [14]. In the context of our work, there are also a number of application that are especially developed for supporting Web service and API annotation [15, 16]. However, since we propose a general approach for classifying

APIs and determining further annotations, any of the existing tagging or service description tools can be extended to include the computed results and present them to the user. In this paper, we verify the applicability of our approach by enhancing SWEET through integration with the developed cross-language classification and central concepts deriving components.

3.2 Cross-lingual Text Classification

Text classification has been successfully applied to many real world problems including spam detection, plagiarism detection or newspaper content classification, and its importance grew quickly with the amount of information available on the Web. Along with the widespread use of text classification methods comes the need for automated classification of new documents or web pages into hierarchies. This can be demonstrated on the Web by the existence of large web directories, such as Open Directory Project or ProgrammableWeb.

Over the past 20 years, text classification largely benefitted from the advances in the field of machine learning [17]. The machine learning approach, which aims at inducing a classifier given a set of training examples, already dominates over the knowledge engineering approach, which consisted of manually constructing the classifier. A common way to address the problem is to represent a textual document using a Vector Space Model [18], i.e. as a weighted vector of terms, and to automatically build a classifier from a set of training examples. While this approach often produces good results when applied to monolingual texts, it is not directly applicable in a multilingual environment.

There are two common approaches to address this problem:

- *Machine translation approach* - involves machine translation of texts to a common language or interlingua and then represents the documents as vectors in that language.
- *Mapping to a shared conceptual space* - represents the documents as term vectors in their source language and then projects them into a shared conceptual space. This is typically done in practice with the help of ontologies/vocabularies or by applying the distributional hypothesis [19].

An approach, which received much attention in the recent, years is to use Wikipedia terms as a shared conceptual space. Texts can be mapped into this space by performing Explicit Semantic Analysis (ESA) [20], hence this method is called Cross-language Explicit Semantic Analysis (CL-ESA) [6]. While there has been significant research involvement in monolingual text classification, the multilingual context has been addressed only recently. The Cross-Language Evaluation Forum (CLEF) has been, over the last decade, the main conference specialising in this research field.

In this paper we describe a Web API classification and annotation method that uses CL-ESA to classify the textual description of a Web API, given a background collection of APIs. The form of CL-ESA that we utilise is equivalent to [6], and lies in finding the correct cross-lingual mapping of the ESA concepts from the Wikipedia. Since CL-ESA uses Wikipedia concepts to represent documents in a multilingual shared vector space, the approach is applicable to the majority of languages.

4 Supporting the Cross-lingual Web API Classification and Annotation

In this section we describe in detail our approach for classifying Web APIs based solely on their textual documentation. We provide the devised algorithm as well as a specific application example. We take the cross-lingual processing one step further and use it to determine the key concepts of the description, which can be used directly as tags or can serve as the basis for deriving further API annotations.

4.1 Cross-lingual Web API Classification

Our approach towards Web API classification is based on comparing the description of an API, which is to be classified, with a set of APIs already classified according to a given taxonomy. The specific implementation of our approach is based on the ProgrammableWeb taxonomy, which comprises of 54 classes (`http://www.programmableweb.com/apis/directory`). We refer to the set of pre-classified services as *Background Collection*. In particular, we determine a number of representative service descriptions for each class in the taxonomy. These service descriptions are used as service models for the classification process. Moreover, the actual classification process is not based on the textual descriptions in the background collection but rather on the pre-computed ESA vector representations, thus saving computation time at runtime.

In addition to the background collection, we also define a set of *stop words*. Web API documentation use very limited vocabulary for describing the format of data and also for describing the behaviour of the Web API. For this reason, a stop-word file must be built to prevent the Explicit Semantic Analysis from focusing on the features of Web API descriptions that do not differentiate the services into classes. Therefore, a sufficiently large document collection in each of the input languages must be acquired and used to build the stop-word list. The stop-word list serves as an input for the pre-processing step of the Explicit Semantic Analysis.

Algorithm 1 formally describes the proposed API classification approach. In particular, the devised method includes the following steps. First we determine the language, in which the Web API description is written. This is currently not an issue and can be done easily by comparing the word distribution of the Web API description to average word distributions of other languages, or using one of the Web Services[1]. Second, we remove the web-service specific stopwords and project the Web API description into the concept space given by the particular language version of Wikipedia. After that we project the vector into the English Wikipedia concept space, to facilitate its comparison with our Web API background. In the following step we iterate over each document in the background and record its similarity with the previously determined vector

[1] `http://code.google.com/apis/language/translate/v1/using_rest_langdetect.html`

of the input Web API description. Finally, for each category, we add up the acquired similarity measure and divide it by the number of examples for the given category. We do this in order to derive a normalised similarity measure, which is not influenced by the number of representative services. There are a number of further ways for determining the similarity measure (selecting the category with best service score, selecting the category with best median, etc.). The output is a list of categories, sorted according to their score.

Algorithm 1 Assigning Class Labels to a Web API Description

```
Require: webAPIDescription, backgroundCollection
Ensure: Scored class suggestions
  language ← recognize_language(webAPIDescription);
  esa_vector ← esa_analyze(language, webAPIDescription);
  esa_vector_en ← esa_map_vector(esa_vector, language, "en");
  category_score ← new Map();
  category_cnt ← new Map();
  for ⟨background_api_vector, category⟩ ∈ backgroundCollection do
    doc_score ← vector_similarity(esa_vector_en, background_api_vector);
    category_score[category] ← category_score[category] + doc_score;
    category_cnt[category] ← category_cnt[category] + 1;
  end for
  for category, score ∈ category_score do
    result[category] ← score / category_cnt[category];
  end for
  sort(result);
  return result
```

Coming back to the example introduced in Section 2, independently of whether we want to classify the GeoNames API or the Czech geocoding API, both descriptions will be converted to English ESA vectors. Based on each vector a list (ideally, an identical list) of sorted categories will be produced. Therefore, independently of the language, both descriptions would in the end be mapped to the same category. We do not consider the case where a new category needs to be created but simply map the API to the closest of the existing categories. Previous approaches base classification on word matches or word stemming/similarity, therefore, they are not applicable to a multi-lingual context.

4.2 Cross-Lingual Web API Annotation

Central Concepts Detection We assume that two APIs can be described with the same central concepts if their descriptions are semantically similar (their semantic relatedness measure is above some threshold). Our approach towards detecting the Central Concepts of a non-english Web API description is to find similar descriptions in a repository of English-based APIs (in this approach serving as background collection), and re-use its central concepts.

The Central Concepts for API descriptions in the repository. i.e. background collection, can be assigned either manually (e.g. by letting users assign keywords

to services), using a concept extraction method or a concept extraction Web service. We will use the concept extraction Web service AlchemyAPI[2]. It would be possible to extract concepts from the non-English WebAPI description directly using the aforementioned Web service, but from our experience the concept detection from an English text yields much better results.

Algorithm 2 Determining the Central Concepts for a Web API Description

```
Require: webAPIDescription, backgroundCollection
  language ← recognize_language(webAPIDescription);
  esa_vector ← esa_analyze(language, webAPIDescription);
  esa_vector_en ← esa_map_vector(esa_vector, language, "en");
  for ⟨background_api_vector, central_concepts⟩ ∈ backgroundCollection do
    score ← cosine_similarity(esa_vector_en, background_api_vector);
    results[score] ← central_concepts;
  end for
  return max(results)
```

Algorithm 2 represents the pseudo-code for our central concepts detection method. First, the language of the input API description is determined, and the description is projected into the ESA concept space of the particular language. Then, the ESA vector is mapped into the English concept space to facilitate its comparison with the ESA vectors of the services in the background API collection. The best matching service from the background collection is chosen and its central concepts are suggested as central concepts for the input API description.

If we use the algorithm to process the examples introduced in Section 2, the central concepts for the GeoNames API can be determined directly by using the AlchemyAPI. However, calculation of the central concepts for the Czech geocoding API is more challenging and is based on computing the cross-lingual similarity between its description and the descriptions in the background collection. The results, however, are comparable for both APIs.

The benefits of determining the central concepts for an API description are multifold. First, they can be used directly as tags for the Web API. These tags can be employed to enhance search within directories or as complementary information presented to the user as part of the API description. However, with some further processing, the central concepts can serve as the basis for determining semantic annotations for separate service parts, such as inputs and outputs, or for extrapolating the domain of the service. In particular, we propose to input the computed words into Watson [21] or Sindice (`http://sindice.com`) and to use the results as suggestions for semantic entities suitable for annotating the API. In our example, two of the central concepts are "latitude" and "longitude", which when posted in Watson return `http://www.w3.org/2003/01/geo/wgs84_pos#long` and `http://www.w3.org/2003/01/geo/wgs84_pos#lat`. These properties can directly be used to semantically describe the inputs of the API. Furthermore, the central concepts can be processed in order to determine

[2] `http://www.alchemyapi.com/`

the domain of the service and extrapolate a set of relevant domain ontologies. However, this work is beyond the scope of the paper but is envisioned as part of our future work.

4.3 Supporting Web API Search and Discovery

The here described methods for cross-lingual classification and determining central concepts can be employed for supporting Web API search and discovery, overcoming language boundaries. In particular, the benefits of enhancing Web API descriptions with classification information and specific key words can be implemented both directly on the level of the API documentation as well as on the semantic level. For instance, existing Web API directories could be extended with search functionality about the type of service or based on keywords describing the service, which in contrast to current solutions, would be language independent. This is a simple, yet effective way for enabling cross-language Web API search.

Furthermore, our work supports enhanced discovery by following the general approach outlined by semantic Web service technologies that aims to reduce the extensive manual effort required for performing tasks such as discovery, negotiation, composition and invocation by enriching services with semantic descriptions of their properties. In particular, the computed classification type and annotations can directly be included as part of lightweight semantic Web API descriptions given in MicroWSMO or SA-REST. These, in turn serve as a basis for applying automated discovery approaches. In the following section we describe the implementation of a system that enables precisely the semi-automatic creation of semantic API descriptions in MicroWSMO, where the user is presented with a list of suitable categories and annotations to choose from.

5 System Design and Implementation

In this section we validate our approach by presenting a system design and giving an implementation solution realised by extending the Semantic Web sErvice Editing Tool – SWEET [3]. SWEET[3] is a Web application developed using JavaScript and ExtGWT, which is started in a Web browser by calling the host URL. It takes as input an HTML Web page describing a Web API and offers functionalities, which enable users to annotate the service properties and to associate semantic information with them. As it can be seen in Figure 2, the architecture of SWEET consists of three main components, including the visualisation component, the data preprocessing component and the annotations recommender. In order to integrate the here presented work, we have extended the interface of the *Annotations Recommender*, to receive input from the *Cross-lingual Classification* and *Central Concept Detection* components.

The implementation of our cross-lingual Web API classification and annotation approach consists of three parts. The first one is the background builder,

[3] `http://sweet.kmi.open.ac.uk/`

which prepares the background collection for further classification, the second one proceeds with the actual classification, and the third one detects the central concepts. As background for the Explicit Semantic Analysis, we use different language versions of Wikipedia, in particular, English and Czech. The text analysis and its projection into ESA concepts space is done by our Java library, created by adapting the code from Wikiprep ESA implementation[4].

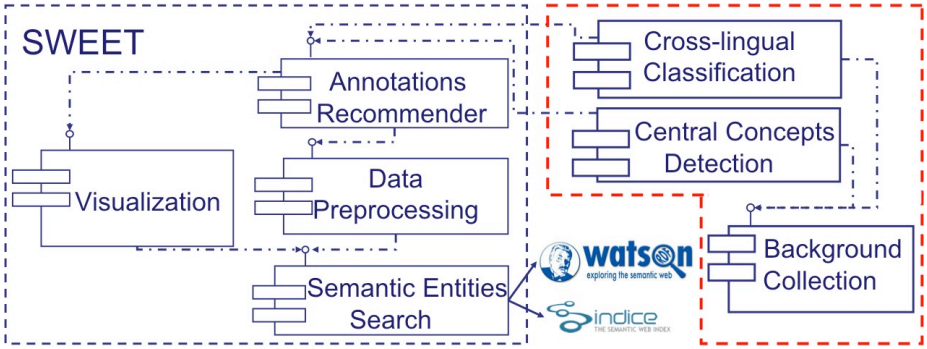


Fig. 2. SWEET Extended Architecture

The Web API background collection is built by getting APIs and categories from `http://www.programmableweb.com`. Five APIs are taken as an example for each category. Information about each API is saved to a database and after that the web pages describing each API are harvested. Subsequently, the HTML mark-up is removed and the text is normalised by removing stop-words and stemming. Then, the ESA vector is computed and stored in the database. Additionally, central concepts for each API in the background collection can be automatically determined by the AlchemyAPI. Before putting the Web API description into the AlchemyAPI engine, we remove the service-specific stop-words to get the Web API specific concepts.

Both, classification and central concept detection operate similarly, and differ only in the last step. They start with projecting the input API description into the Czech Wikipedia concept space. Then, the resulting Czech ESA vector is mapped into English ESA vector, using the concept mapping from Wikipedia. Afterwords, the ESA vector is compared with each API description ESA vector from the API background collection. The last step is the following:

– In case of classification, the results are aggregated and the best categories are suggested as candidates.

– Concept detection does not summarise the results but rather suggests the central concepts of the first few most semantically similar Web APIs as concept suggestions.

The so computed results can be represented to the user as annotation suggestions, aiding the process of creating the semantic Web API description. In the case of the classification of the service functionality, the top 3 results, for ex-

[4] `http://github.com/faraday/wikiprep-esa`

ample, can be automatically assigned to the API and the annotator would only need to validate them.

We also ran some preliminary evaluation and tests. In particular, we ran the concept detection system on the APIs from the geocoding domain. The first phase, which identifies the most similar service worked quite well, and was able to locate relevant similar Web APIs. Therefore the classification task was completed successfully. This evaluation needs to be extended to cover further domains, in order to be able to make statements about the precision of the classification approach in general. Our previous experiments with CL-ESA reported in [22] suggest that the method is able to detect semantically comparable text across languages with high precision (about 0.7 precision at top_{50}) from a 3.5 million large corpus. Given the fact that the size of ProgrammableWeb is smaller and we are classifying only into 54 classes, significantly better results can be expected.

In contrast, the concept extraction phase must be further refined because the returned central concepts were not always relevant. We discovered that the results greatly depend on the quality of the background collection. In particular, we are using the Web APIs from the ProgrammableWeb directory, where Web APIs are sometimes assigned to the wrong category or the link to the API documentation is inaccurate. We can overcome these limitations by hand-picking the APIs per category or by ensuring that the URLs pointing to the API documentation are correct. Even if improvements still remain to be done, the initial results show that the approach, especially in the context of the classification task, is quite promising.

6 Conclusions and Future Work

Nowadays, finding, interpreting and invoking Web APIs requires extensive human involvement due to the fact that the majority of the APIs have only textual documentation, not conforming to any particular standards and guidelines. Moreover, providers publish API description in any language that they see fit, making the discovery of suitable services a challenging tasks. In this paper, we present a cross-lingual approach, based on calculating semantic similarity, for classifying APIs and determining the central concepts of their descriptions, thus enabling language-independent search and discovery. We validate the applicability of the proposed method by implementing it as part of an extension to SWEET [3], which support users in creating semantic Web API descriptions. We also give some preliminary test results. Future work will mainly focus on extensively evaluating the system, staring off with improving the quality of the background collection and covering further domains in addition to the geocoding/mapping one.

Acknowledgment The work presented in this paper is partially supported by funding from the EC FP7, under grant agreement number 270001 – Decipher

References

1. M. J. Hadley: Web Application Description Language (WADL). Technical report, Sun Microsystems, November 2006. Available at `https://wadl.dev.java.net`.

2. Web Services Description Language (WSDL) Version 2.0. Recommendation, W3C, June 2007. Available at http://www.w3.org/TR/wsdl20/.
3. M. Maleshkova, C. Pedrinaci, J. Domingue: Semantic annotation of Web APIs with SWEET. 6th Workshop on Scripting and Development for the Semantic Web at ESWC, 2010.
4. L. Richardson, S. Ruby: RESTful Web Services. O'Reilly Media, May 2007.
5. R. T. Fielding: Architectural styles and the design of network-based software architectures. PhD thesis, University of California, 2000.
6. P. Sorg, P. Cimiano: Cross-lingual information retrieval with explicit semantic analysis. In Working Notes for the CLEF Workshop, 2008.
7. J. Kopecký, T. Vitvar, D. Fensel, K. Gomadam: hRESTS & MicroWSMO. Technical report, available at http://cms-wg.sti2.org/TR/d12/, 2009.
8. J. Kopecký, T. Vitvar, C. Bournez, J. Farrel. SAWSDL: Semantic Annotations for WSDL and XML Schema. IEEE Internet Computing, 11(6):60-67, 2007.
9. J. Kopecký , K. Gomadam, T.Vitvar: hRESTS: an HTML Microformat for Describing RESTful Web Services. In Proc of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence (WI-08), 2008.
10. A. P. Sheth, K. Gomadam, J. Lathem: SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. In IEEE Internet Computing, 2007.
11. RDFa in XHTML: Syntax and Processing. Proposed Recommendation, W3C, September 2008. Available at http://www.w3.org/TR/rdfa-syntax/.
12. S. C. Sood, K. J. Hammond. TagAssist: Automatic tag suggestion for blog posts. In Proc of International Conference on Weblogs and Social, 2007.
13. S. Lee, A. Chun: Automatic tag recommendation for the web 2.0 blogosphere using collaborative tagging and hybrid ANN semantic structures. 6th Conference on WSEAS International Conference on Applied Computer Science, 2007.
14. R. Jaeschke, R. Marinho, A. Hotho, L. Schmidt-Thieme, G. Stumme: Tag recommendations in folksonomies. In PKDD, pages 506514, Springer, 2007.
15. A. Hess, E. Johnston, N. Kushmerick: ASSAM: A tool for semiautomatically annotating semantic web services. In Proc of the 3rd International Semantic Web Conference (ISWC), 2004.
16. A. Patil, S. Oundhakar, A. Sheth, K. Verma: METEOR-S web service annotation framework. pages 553562. ACM Press, 2004.
17. F. Sebastiani: Machine learning in automated text categorization. ACM Computing Surveys, 34(1):1-47, 2002.
18. C. D. Manning, P. Raghavan, H. Schutze: Introduction to Information Retrieval. Cambridge Press, 2008.
19. C. D. Manning, H. Schutze: Foundations of Statistical Natural Language Processing. The MIT Press, 1999
20. E. Gabrilovich, S. Markovitch: Computing semantic relatedness using Wikipedia-based explicit semantic analysis. Proceedings of IJCAI, 1606-1611, 2007.
21. Watson - The Semantic Web Gateway: Ontology Editor Plugins. http://watson.kmi.open.ac.uk. Online November 2008.
22. P. Knoth, L. Zilka, Z. Zdrahal: Using Explicit Semantic Analysis for Cross-Lingual Link Discovery. Workshop: 5th International Workshop on Cross Lingual Information Access: Computational Linguistics and the Information Need of Multilingual Societies (CLIA) at The 5th International Joint Conference on Natural Language Processing (IJC-NLP 2011), Chiang Mai, Thailand, 2011.