# A Topic Modeling Approach for Web Service Annotation

Leandro Ordonez-Ante*[1], Ruben Verborgh†[2] and Juan Carlos Corrales‡[1]

[1]Grupo de Ingeniería Telemática (GIT), Universidad del Cauca , Calle 5 No 4-70, Popayán, Colombia
[2]Multimedia Lab, Ghent University , Gaston Crommenlaan 8/102, 9050 Ghent, Belgium

April 9, 2014

**Abstract**

## 1 Introduction

During the last two decades we have been witnessing how the Web has evolved from being a text and image repository on its early stages, to provide a huge offer of both information-providing and world-altering services. The current Web has posed a paradigm that revolutionized the generation and consumption dynamics of this kind of resources, encouraging its users, not merely to consume these services, but also to build and publish them.

This dominant paradigm of the current Web, has inspired the conception of initiatives into other communities such as the Telco providers, which include the GSMA's One API [1], and the ECMA-348 [2] and ECMA-323 [3] standards. Such initiatives promote for network operators to expose their capabilities and information via Web service interfaces, easing this way for users (service designers and developers) to create and deploy new telecom services, with a reduced time-to-market and tailored to their specific needs.

Thus, the service offering inside the Web is diversifying and steadily growing, so it is necessary to provide the users with increasingly intelligent mechanisms for services search and retrieval, identifying in a truthful way the functionality

*leandro@unicauca.edu.co
†ruben.verborgh@ugent.be
‡jcorral@unicauca.edu.co

provided by such resources while being able to deliver relevant services to the customer. The above has meant the transition from the traditional keyword-based or table-based search methods [4], to approaches supported on semantic Web technologies which provide meaning for both the services specifications, and user queries, through a formal and machine-readable specification of knowledge (e.g. ontologies, taxonomies, lexical database and so on).

In practical terms, however, the actual implementation of semantic-based mechanisms for service retrieval has been restricted precisely due to the expensive procedure involved in the formal specification of services. Such a procedure comprises a time-consuming task of semantic annotation, performed by hand by service developers, who additionally require specialized knowledge on models for semantic description of services (e.g. OWL-S, WSMO), as well as the aforementioned formal specifications of knowledge.

In order to overcome this limitation, currently some approaches are considered to tackle the problem of semantic service annotation, by applying knowledge discovery and emergent semantics techniques over a huge corpus of service descriptors, which in some cases already contains annotations made by consumers in a collaborative way. Those approaches however, has serious limitations in terms of the reliability of the users feedback they are built upon, which impacts the precision of search and selection tasks. Therefore it's considered necessary to develop mechanisms that enable the automation of semantic service annotation tasks.

This paper introduces our proposal for service annotation, based on processing existing web service documentation resources for extracting information regarding its offered capabilities. By uncover the hidden semantic structure of such information through statistical analysis techniques, we are able to associate meaningful annotations to the services operations, and to group those operations into non-exclusive semantic related categories.

Based on this approach we have build Topicalizer, a tool that allows the user to process a bunch of SOAP API descriptors (WSDL documents), in order to group the technical information they contain into semantic categories, and specifying such categorization as RDF statements stored in a Sesame triple-store, to which users may access and issue SPARQL queries.

## 2 Motivation and Background

The Web is emerging as a medium for connecting distributed applications, becoming—more than an information system—into a platform that supports the operation of a huge ecosystem of services [5], which are built under different architectures and design philosophies. Leonard Richardson has proposed in [6] a schema for classifying services on the Web, defining three maturity levels (plus a zero level). Each of the levels represents one element of what Richardson calls *the technology stack for web services*: URI, HTTP, Hypermedia (see Figure 1). This way, services are classified according to the technologies that supports their operation.

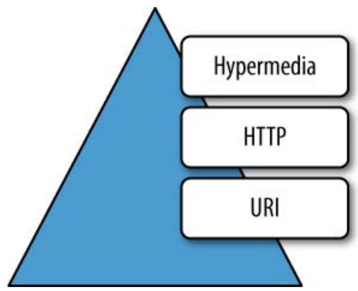


Figure 1: Richardson's service maturity model. source: [5]

- *Level zero services*: services at this level are characterized by having an unique URI and using only one HTTP method (typically `POST`). At this level we find the XML-RPC services [7] and most of the SOAP services.

- *Level one services (URI support)*: At this level, services employ various URIs but only one HTTP method. In contrast to level zero services—which tunnel all the interactions through a unique and rather complex resource—services at level one expose multiple logical resources. However services at this level tunnel their actions by inserting parameters and values into a URI, which is then transmitted to a remote service (via `HTTP GET` typically). According to Richardson, most of the services out there that claim to be RESTful are actually level one services.

- *Level two services (HTTP support)*: this level deals with services that host many resources, each of which is addressable through its own URI. Additionally level two services support various HTTP methods on their resources. This level includes CRUD-like (*Create*, *Read*, *Update*, *Delete*) services, such as the Amazon cloud storage system (*Amazon S3*: http://aws.amazon.com/es/s3/).

- *Level three services (Hypermedia support)*: At this level, we find real RESTful services: those having the features of level two services, plus supporting the notion of *hypermedia as the engine of application state* (HATEOAS), that is to say, the representations of the resources hosted by the service contain controls that enable consumers to access related resources. This way the service leads its users through a trail of resources, causing application state changes in consequence. Examples of this kind of services include the Web and the REST API of *Netflix* (http://developer.netflix.com/docs/REST_API_Conventio

A research conducted by Maleshkova et al. [8] reports that, despite the apparent spreading of RESTful services in the Web, there are actually few services that supports all the tenets and constraints of REST. The authors of this study have analyzed by hand 222 web APIs, randomly chosen from the *Programmable Web* API directory [1]. The results that arose from their analysis, evidence that

[1]Available at: http://www.programmableweb.com/

only 32% of services could be considered—at least approximately—REST services (i.e., services from levels two and three in the Richardson maturity model), while the remaining 68% was RPC and hybrid services (i.e., services from levels zero and one, according to the same model).

The study of Maleshkova also states that service development is driven by the particular criteria of its creators, rather than well-established standards and rules. Similarly, service documentation (specially REST service documentation) is not supported on interface description languages such as WSDL (for SOAP services), but it is provided as HTML pages, which have no regular or standard structure. Therefore, the use of web services requires a cumbersome manual process which additionally hinders the execution of discovery, composition and invocation procedures. In this regard, some initiatives have been fostered, seeking the definition of standard formats for describing REST services. That is the case of WADL (*Web Application Description Language*) [9], a language intended for specifying HTTP-based web services and applications.

A WADL descriptor (or contract) is a document that specifies the set resources hosted by a service, as well as their associated URI templates, the HTTP methods the service supports and the representations it is able to receive and deliver. Just like WSDL, WADL enables automatic building of services clients, making them easier to consume and accessible to developers. Nonetheless, WADL descriptors merely describe the static view of services and applications, neglecting the user-resources interaction dynamics, which is better specified by hypermedia and media types. Consequently, as stated in [10], this kind of descriptor is suitable only for CRUD-like REST services (Level two services) whose functionality is limited to manipulate records from a data set.

So far, WADL has been poorly adopted as description language for REST services. Instead other studies have been conducted for defining service descriptors that include semantic metadata, which aim to enable the automatic discovery and composition of services. Semantic annotations make it possible for intelligent agents to understand the services functionality, and establishing service relationships at the semantic level (e.g., similarity, partial matching, and membership[11]).

In this regard the academic community has came up with proposals like hRESTS [12]—an HTML-based description microformat that allows specifying the services functional attributes, like its operations, inputs and outputs—, along with its extensions SA-REST [13] and MicroWSMO [14], which enable the semantic annotation of hREST descriptors.

Another approach to REST API description is *RESTdesc*, proposed by Verborgh et al. in [15]. RESTdesc provides a functionality-centered format for semantic description of REST services, as well as an automatic discovery mechanism based on inference with logic rules specifying the capabilities of these resources. According to the authors of RESTdesc, the approach they propose is supported on well known technologies such as HTTP and RDF/Notation3 and is grounded on the hypermedia and *Linked Data* concepts, from which defines and leverages relationships between different services specifications, enabling intelligent agents to automatically discover and compose services.

Since, in general terms, there is a sort of barrier regarding the adoption of new formats for specifying the web service semantics, our proposal intends to use the information currently available in service description documents—i.e., WSDL interfaces for SOAP services and HTML documents for XML-RPC and REST services—in order to abstract a knowledge representation based on the content of such documentation, from which it would be possible to establish semantic similarity relations between services. This proposal is founded on three main processes:

1. Extraction of technical information related to service functionality.
2. Analysis of the extracted information for identifying conceptual categories the services they comprise.
3. Deriving a taxonomy from the categories obtained in process (2).

# 3 Related Work

At the end of the last section we reviewed some works regarding description of REST APIs. This section continues with that review but focuses on works addressing semantic annotation of Web services and resources in general.

In [16] Loutas et al. explore alternative approaches for semantic annotation of available services and resources in the Web. Such an approach consists of recognizing the information constructs from collaborative tagging systems (also known as folksonomies) as specifications of shared knowledge, which can be suitable for associate semantic annotations to service interfaces, dispensing with the use of ontologies. The main goal of these proposals, however, is to assist the process of semantic enrichment, still requiring human intervention (developers, users, providers, etcetera) for fulfilling the complete process.

The authors of [12] and [13] address two works regarding to semantic annotation of folksonomies, for various kinds of online available resources. In contrast to aforementioned works, the proposals of Angeletou in [12] and the one described by Siorpaes in [13] argue that it is required to formalize the knowledge generated within folksonomies, by using ontologies, in order to overcome their limitations in terms of organizing, searching and retrieving resources based on tags.

The work of Angeletou differs from the current proposal, as long as the former is focused on an image folksonomy. In turn, the project addressed in [13], although it takes into account the services as part of its working resources, its scope is limited to promote collaborative tagging thereof.

The approaches outlined in [14, 15, 16, 17] pose the use of techniques of machine learning such as Formal Concept Analysis (FCA) and most recently Relational Concept Analysis (RCA), for extracting and representing the knowledge covered by documental corpus, as conceptual hierarchies or taxonomies. This way, the approaches described in these works are suitable for composing formal models of knowledge, such as core ontologies, avoiding the intervention of domain experts. However, none of the aforesaid proposals had considered the automation of such a process.

From observations made on related proposals, the present work aims to automate the process of semantic annotation of web services descriptors, through an approach that combines techniques of text mining, unsupervised machine learning (Latent Dirichlet Allocation–LDA) for enabling automatic and incremental generation of formal models of knowledge from service descriptors. Such models are meant to be used in annotating and categorizing services, through a platform that implements the above techniques.

Next section will address the description of our proposal, by outlining the architecture of the platform for automatic semantic annotation of service descriptors.

# 4 Proposal

## 4.1 Analysis of Web Service documentation sources

As evidenced in the study by Maleshkova et al. [8], web service documentation is often limited by the content that API developers provide on their websites. SOAP services are a special case, whose main descriptor is a WSDL document, which defines an abstract service interface (information regarding operations, messages and types) and concrete details about transport and location of the service. Following subsections deal with the description of mechanisms for extraction of technical information regarding service functionality, from various documentation sources: WSDL descriptors for SOAP services, and HTML pages for XML-RPC and REST services.

### 4.1.1 SOAP Services

WSDL is an XML standard format for Web service description [32]. A WSDL document describes service interface abstractly and provides concrete technical details about service operation. This may be visualized in figure 2, which shows the structure of a WSDL descriptor.

The diagram of figure 2, shows the separation between service's abstract description and concrete details. The later refer to element that specify service endpoints, and communication and transport protocols used for message exchange. These concrete details are required for service invocation, however they provide little information about service functionality, this is why descriptor analysis focuses on the components of the abstract description of the service interface, namely:

- *Types (schema, element, complexElement, sequence)*: define the data types composing the messages exchanged in service invocation.
- *Message* : represents an abstract definition of data transmitted when invoking a particular service operation.

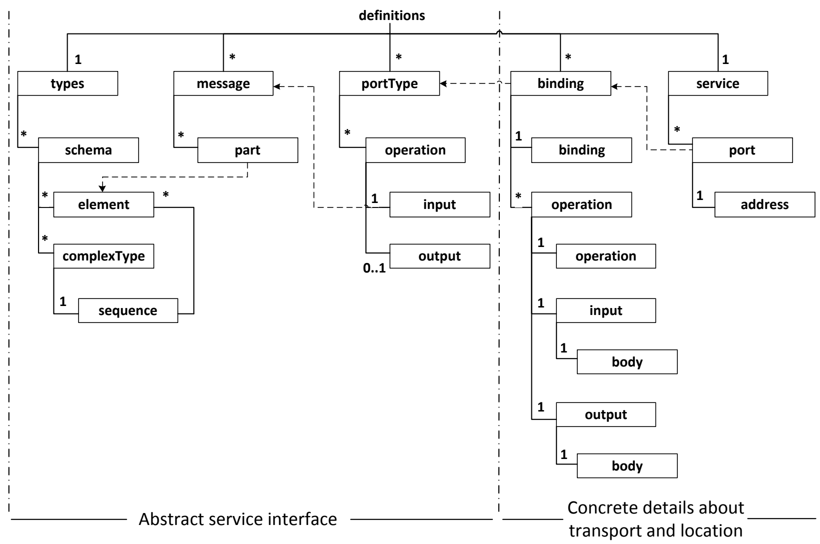


Figure 2: Structure of WSDL descriptor.

- *PortType*: it is defined as a grouping of abstract service operations along with their associated messages.
- *Operation*: abstracts service functional units. This descriptor element is associated with a set of input/output messages.

Additionally, WSDL allows natural language description for some of the service interface elements, including: service, binding, portType, operation, message and types. Such a description is provided by service developers and is enclosed within a special tag called documentation. As argued by Falleri in [8], typically there is some redundancy in the information contained in certain elements of service interface. Thus, for instance, terms defining ports, bindings and portTypes are frequently the same used for describing the service element; likewise terms defining input/output messages, are slight variations of the term specifying their associated operation. In consequence, it was decided that information extraction from service descriptor only takes the content of service, operation and types elements into account, including their natural language descriptions (when available). This way, it is possible to obtain a simplified model of the information the service interface contains, considering the three mentioned elements. This model is shown in figure 3.

In the model above, *DataElement* and *ComplexDataElement* elements represent *simple* and *complex types* (*types*) respectively, which compose the message exchanged when invoking a service operation. Typically the terms used in defining such elements of the WSDL descriptor follow naming conventions commonly adopted by programmer, e.g. using `CamelCase` compound words for identifying

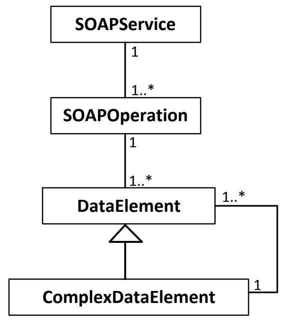


Figure 3: Simple model for describing SOAP Services.

operations, types and services. Similarly, sometimes documentation tags contain HTML encoded data. Therefore it is necessary to get the content into a proper format to enable further processing. This involves the use of text mining techniques such as *tokenization*,*POS (Part-of-speech) tagging* and *spell checking* whose description is addressed in section 4.1.3.

### 4.1.2 XML-RPC and REST Services

As it was mentioned at the beginning of section 4.1, Web service documentation—except for SOAP services—doesn't meet any standard format. XML-RPC and REST services, are commonly described by HTML pages, which provide information regarding service functionality and endpoints [2]. Usually the content of such pages doesn't follow a formal structure, making it difficult to extract relevant information in an automated way. There are some initiatives, including *ProgrammableWeb* and *APIhub* [3], which promotes the creation of centralized API directories, where service documentation is uniformly stored, by following a regularstructure. However, a major issue regarding these initiatives is that documentation must be registered manually, so more often than not the information provided either contains errors (typos, broken links) or is outdated.

Given this limitation, it was decided to deal with documentation that developers provide on API websites. This way, we apply on each HTML page an analysis that involves identifying recurring patterns (which depends on the service type, either XML-RPC or REST) and document segmentation for extracting relevant information regarding service functionality. This analysis is supported on the approach of Ly et al. formulated in [17].

**Analysis of XML-RPC service documentation** Similar to SOAP services, an XML-RPC service defines a set of operations or procedures that clients

[2]For example: (XML-RPC) ,http://www.benchmarkemail.com/API/Library (REST) https://dev.twitter.com/docs/api/1.1

[3]Available at: http://www.apihub.com/

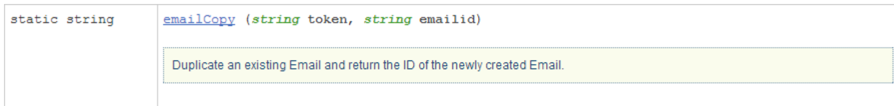


Figure 4: Operation description block - XML-RPC service. Source: http://www.benchmarkemail.com/API/Library

can invoke remotely. The documentation of this kind of services, deals with the operations they expose, the arguments they require for their invocation, and usually specifies the service endpoint (URI) and the HTTP method used for communication.

The description of operations within the same XML-RPC service documentation tends to share similar structure and content. Thus, for example, operation identifiers are usually defined by terms in `CamelCase` notation, composed of a verb and a noun (e. g., *getWeather*). Also, given that this kind of documentation is intended for humans, its content is articulated with recurrent visual clues that help identify operation description blocks within the HTML page: e.g., operation identifiers may be enclosed in <h3> tags and their natural language descriptions in <p> tags. Then, it is possible to use those local patterns present in each of the service documentation pages, for extracting the information blocks regarding service operations.

Prior to the extraction of operation description blocks, a preprocessing step is required for getting rid of images, scripts, malformed tags and formatting tags (<b>, <strong>, <i>, etc.) from HTML documentation pages.

Then the operation description blocks are extracted, by following the steps below:

1. Extraction of `CamelCase` terms composed of a verb and a noun (which make up the set of candidate operations identifiers), while keeping track of the HTML tag that encloses them.

2. Identify the most commonly used tag (*elected tag*) for the operation identifiers found in the first step.

3. Finally, out of all the operation identifiers found in step #1, only retain identifiers within *elected tags*. The page is then segmented according to the scope of the tags enclosing each of the chosen operation identifiers.

By following the above procedure, it is possible to identify and extract the textual data from operation description blocks, such as that illustrated in Figure 4.

**Analysis of REST APIs documentation** REST services operate according to the principles and constraints addressed in section **??**. While nowadays there is a seemingly increasing adoption of this architectural style for building web services, actually only a few of them support all the guidelines that REST

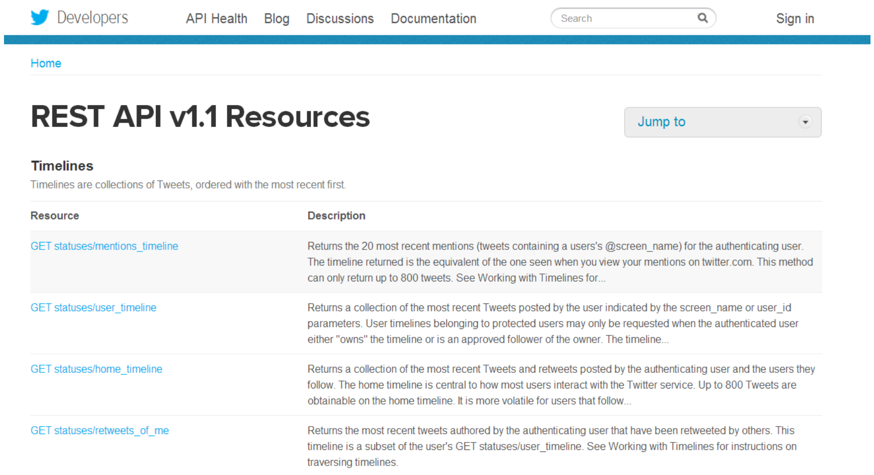


Figure 5: Documentation of the Twitter REST API. Source: https://dev.twitter.com/docs/api/1.1

defines. In terms of the Richardson's maturity model explained in section **??**, most of the existing REST services are in fact *level one* services (URI supported), while few of them qualify as *level two* (URI & HTTP supported) and *level three* (RESTFul: URI, HTTP & Hypermedia supported) services.

Documentation of this kind of services, provided as HTML pages, is focused on the concept of resources and the parameters used for identifying them, which are encoded into URI templates (e.g., `/{resource}/{property}/`). In addition, this documentation specifies the allowed HTTP methods (`GET`, `POST`, `PUT`, `DELETE`, etcetera) for each of the resources hosted by the service. See for example the documentation available on the website of the twitter API for developers shown in Figure 5.

This way, the analysis of REST services documentation focuses on detection and extraction of resources description blocks, which contain URI templates, HTTP methods and usually a description in natural language. Since REST API developers tend to adopt a recurring pattern for documenting the service resources, it is possible to perform a segmentation procedure on the HTML content, in order to extract the mentioned description blocks.

Such a procedure starts from the analysis of the HTML DOM tree to perform the steps listed below:

1. Segment the HTML document into blocks containing URIs and HTTP methods.

2. Compute the similarity between the blocks found in step #1, and retain those having the same structure.

3. Extract the information of each resource, contained within the blocks identified in the previous step: resource URI, supported methods and description in natural language.

For conducting the second step of the previous procedure, the authors of [17] rely on the concepts of *entropy* and *node internal structure*. Entropy is a measure that quantifies local patterns present in segments of the HTML document, so that a high entropy suggests an irregular structure, while a low entropy denotes a substantial similarity. The internal structure of a node from the HTML DOM tree, consists in the concatenation of the tags that make up such node, so for example, given the following page segment `<div><a><span>link</span></a><p>text</p></div>`, the internal structure of the `div` node is: `<div><a><span><p>`.

This way, the entropy measure estimates the similarity between the document segments found in step #1, which is subsequently used for obtaining the set of resources descriptor blocks included in the service documentation.

#### 4.1.3 Service documentation pre-processing

Often, the information that developers provide in service descriptors follows naming conventions commonly used in programming languages, in particular they use compound words such as *helloword*, *hello_world* or *helloWord* for identifying operations, resources and parameters. Also, it might be possible for descriptions in natural language to include content encoded into HTML or XML tags, which are used for formatting the text or providing structure to it.

A requirement for further analyze the information extracted from service documentation (see end of section 2) consists in processing such information and turning it into plain text, which involves splitting compound words and getting rid of HTML or XML tags. To this end it has been decided to use certain text mining techniques, including *tokenization*, *spell checking* and *POS tagging*, whose description is addressed below.

**Tokenization** This is a procedure that allows breaking a sequence of characters up into its composing tokens. The information extracted from service descriptors is subject to this process to generate the list of terms included within identifiers of operation, resources and types, as well as the text of descriptions in natural language. So, for example the list of tokens contained in the sequence "*getWeatherByCity*" is: {"*get*", "*Weather*" "*By*", "*City*"}. In the previous example, the method for breaking the compound term up relies on detecting transition from lowercase to uppercase along the sequence. However, there are some cases where this procedure is not that straightforward: e.g., the sequence "*getweatherbycity*" doesn't use letter case for telling tokens apart. In these special cases a spell checking technique is used, as outlined next.

**Spell checking** It is clear for a human that strings "*homeaddress*" and "*home address*" contain the same information. However for machines those are two different sequences of characters and there is no trivial way for it to tell that

Table 1: Trie-based compound splitting for "*homeaddress*".

| *homeaddress* | | | | |
|---|---|---|---|---|
| *iteration* | *1st segment* | *Does it exist in Trie?* | *2nd segment* | *Does it exist in Trie?* |
| 0 | *h* | ✓ | *omeaddress* | X |
| 1 | *ho* | X | *meaddress* | X |
| 2 | *hom* | ✓ | *eaddress* | X |
| 3 | *home* | ✓ | *address* | ✓ |

they are equivalent strings. This turns out to be a common problem in Information Retrieval [18] known as *compound splitting*. For tackling this compound splitting problem, we use a mechanism based on term look up over a tree-like data structure called *Trie* [19] which in this particular case encodes the terms from the *words* dictionary [4], included in Unix-like operating systems. This data structure usually supports spell checking and autocomplete software, used in search engines and some other web and mobile applications.

*Trie-based* search is similar to the way we look up words in a dictionary, i.e. by using prefixes. For extracting the terms that make up a compound word an iterative procedure is conducted: (1) It divides the sequence of characters into segments with different length; (2) it look up each of the segments in the Trie. This process continues until all the segments obtained in step #1 match terms included in the dictionary. Table 1, shows this procedure applied on the sequence "*homeaddress*".

**Part-of-speech (POS) tagging** This technique, is used for detecting candidate operations described in the documentation of XML-RPC services. According to the analysis outlined in section **??**, the operations hosted by this kind of services, are frequently identified by `CamelCase` terms joining at least a verb and a noun (e. g., *getWeather*). POS tagging allows specifying the *lexical category* (verb, noun, pronoun, adverb, etcetera) corresponding to each term in a text or sentence. This way it is possible to filter out the compound words included in XML-RPC service documentation and extracting the set of candidate operations. The POS tagging technique used in our case is the one conceived by Toutanova, outlined in [20] and developed inside The *Stanford Natural Language Processing Group*.

By applying the techniques explained above on the documentation of Web services, the noise involved in the statistical analysis intended to identify groups of similar services is reduced. As defined at the end of section **??**, this analysis is the second of the key processes that comprise the approach documented herein. The description of this second process is addressed in the section below.

[4] Example available at: http://www.cs.duke.edu/~ola/ap/linuxwords

# 5 Experimentation

# 6 Discussion and Conclusions

# References


[1] GSMA, "Gsma–oneapi for developers," 2013.

[2] ECMA-International, "Standard ecma-348 web services description language (wsdl) for csta phase iii," 2012.

[3] ECMA-International, "Standard ecma-323 xml protocol for computer supported telecommunications applications (csta) phase iii," 2011.

[4] A. Bernstein and M. Klein, "Towards high-precision service retrieval," in *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC '02, (London, UK, UK), pp. 84–101, Springer-Verlag, 2002.

[5] J. Webber, S. Parastatidis, and I. Robinson, "The web as a platform for building distributed systems," in *REST in Practice*, O'Reilly Media, 2010.

[6] L. Richardson, "Justice will take us millions of intricate moves," 2008.

[7] S. S. Laurent, E. Dumbill, and J. Johnston, *Programming Web Services with XML-RPC*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2001.

[8] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Investigating web apis on the world wide web," in *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, pp. 107–114, 2010.

[9] M. Hadley, "Web application description language," 2009.

[10] J. Webber, S. Parastatidis, and I. Robinson, "The web and and ws-," in *REST in Practice*, O'Reilly Media, 2010.

[11] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara, "Semantic matching of web services capabilities," in *Proceedings of the First International Semantic Web Conference on The Semantic Web*, ISWC '02, (London, UK, UK), pp. 333–347, Springer-Verlag, 2002.

[12] J. Kopecký, K. Gomadam, and T. Vitvar, "hrests: An html microformat for describing restful web services," in *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '08, (Washington, DC, USA), pp. 619–625, IEEE Computer Society, 2008.

[13] A. P. Sheth, K. Gomadam, and J. Lathem, "Sa-rest: Semantically interoperable and easier-to-use services and mashups," *IEEE Internet Computing*, vol. 11, pp. 91–94, Nov. 2007.

[14] J. Kopecký, D. Fensel, and T. Vitvar, "D3.4.3 microwsmo and hrests," 2009.

[15] R. Verborgh, T. Steiner, D. Deursen, J. Roo, R. V. D. Walle, and J. Gabarró Vallés, "Capturing the functionality of web services with functional descriptions," *Multimedia Tools Appl.*, vol. 64, pp. 365–387, May 2013.

[16] N. Loutas, V. Peristeras, and K. Tarabanis, "Rethinking the semantic annotation of services," in *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops* (A. Dan, F. Gittler, and F. Toumani, eds.), vol. 6275 of *Lecture Notes in Computer Science*, pp. 540–549, Springer Berlin Heidelberg, 2010.

[17] P. A. Ly, C. Pedrinaci, and J. Domingue, "Automated information extraction from web apis documentation," in *Proceedings of the 13th International Conference on Web Information Systems Engineering*, WISE'12, (Berlin, Heidelberg), pp. 497–511, Springer-Verlag, 2012.

[18] E. Airio, "Word normalization and decompounding in mono- and bilingual ir," *Inf. Retr.*, vol. 9, pp. 249–271, June 2006.

[19] E. Fredkin, "Trie memory," *Commun. ACM*, vol. 3, pp. 490–499, Sept. 1960.

[20] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, (Stroudsburg, PA, USA), pp. 173–180, Association for Computational Linguistics, 2003.