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, SAWSDL), 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.

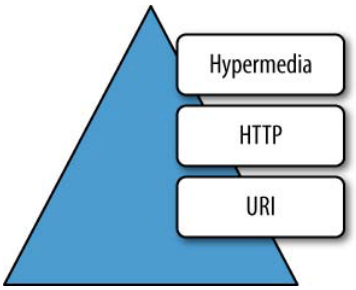

Figure 1: Richardson's service maturity model. source: [5]

- *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 for SOAP services, 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 REST APIs description. 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. In this work the author conceive information constructs derived from collaborative tagging systems (also known as folksonomies) as specifications of shared knowledge, which may be suitable for associate semantic annotations to service interfaces. However, as stated by Martinez-Cruz et al. in [17], the wide-open nature of folksonomies involves some shortcomings in terms of organizing, searching and retrieval of resources based on tags, due to its lack of formal semantics. That way, the authors of [17] propose to build an *ontology-based semantic layer* on top of these collaborative tagging systems to formalize the knowledge gathered within them.

In other related work by Loutas et al. [18], they introduce a proposal for a search engine for web services, which operates on service descriptors specified in SA-REST or SAWSDL. In order to deal with heterogeneous service description formats, the authors define a comprehensive *Reference Service* model to which all the crawled services are mapped. The main shortcoming of this approach lies on the fact that it operates on semantic description formats whose adoption is rather limited.

The approach outlined in [19] by Azmeh et al. pose the use of techniques of machine learning such as Formal Concept Analysis (FCA) and Relational Concept Analysis (RCA), for extracting and representing the technical information contained in service descriptors as conceptual hierarchies. However, the approach introduced in this work is not that exhaustive when it comes to identify similarity between services, since it relies on syntactic comparison of the terms comprising the service interfaces (i.e. WSDLs).

The approach we propose contributes towards automating the process of semantic annotation of web services descriptors, by combining techniques of text mining and unsupervised machine learning (i.e. Latent Dirichlet Allocation–LDA) for enabling automatic and incremental generation of a formal model of knowledge from existing service documentation sources. Such model is meant to be used in annotating and categorizing services operations, through a platform that implements the above techniques.

Next section will address the description of our proposal, by outlining each one of the three processes stated at the end of section 2.

4 Proposal

This section addresses a detailed description of the foundations of our proposal. First, the analysis of Web services documentation is outlined, then the probabilistic topic model used for deriving the hidden semantic structure from such documentation is introduced, and finally the formal model for representing that semantic structure is explained.

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 extracting 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. 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.

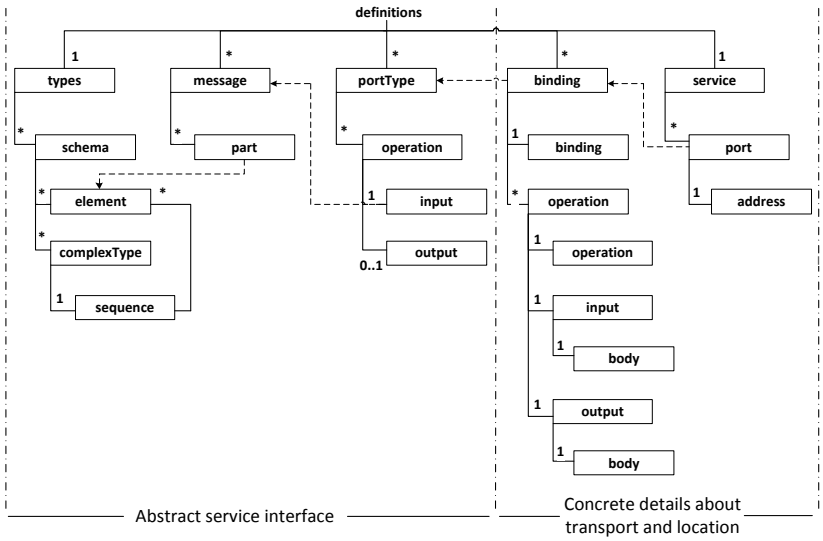

Figure 2: Structure of WSDL descriptor.

- *Message* : represents an abstract definition of data transmitted when invoking a particular service operation.
- *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`. 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

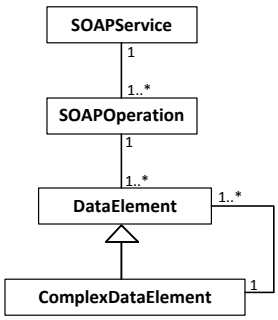

Figure 3: Simple model for describing SOAP Services.

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 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 regular structure. 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 [20].

[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

Analysis of XML-RPC service documentation Similar to SOAP services, an XML-RPC service defines a set of operations or procedures that clients 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 While nowadays there is a seemingly increasing adoption of the REST architectural style for building web services, actually only a few of them support all the guidelines that REST defines.

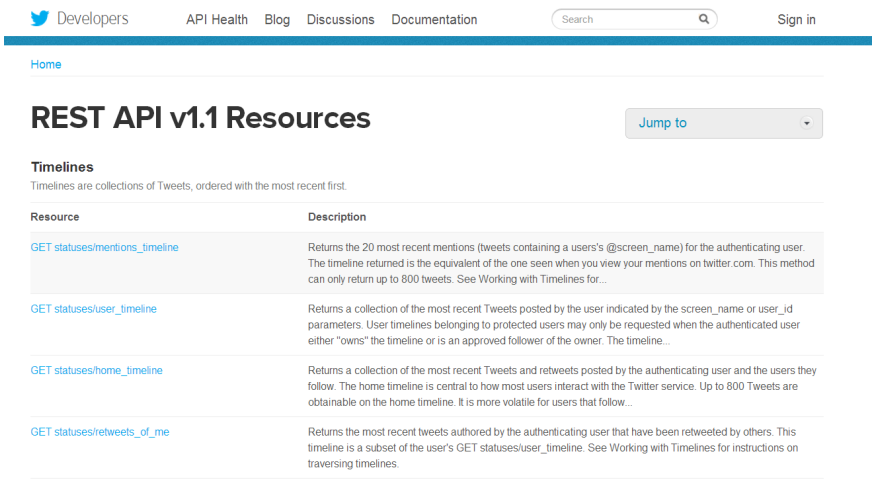

Figure 5: Documentation of the Twitter REST API. Source: https://dev.twitter.com/docs/api/1.1

In terms of the Richardson's maturity model explained in section 2, 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 [20] 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 [21] 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* [22] 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 4.1.2, 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 [23] 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 2, 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

4.2 Discovering the semantic structure of Web service documentation by applying Probabilistic topic modeling

Nowadays the amount of information and resources available on the Web is huge and ever increasing, so that it has exceeded our ability for locating and accessing the resources we need. This way, increasingly sophisticated computational tools are required for organizing, searching and understanding those resources, beyond the traditional information indexing and retrieval approaches.

In this regard, over the last decade a variety of statistical models known as probabilistic topic models have emerged [24] as powerful tools that allow to uncover the hidden thematic structure that runs through a document collection.

The algorithms developed for applying these models, may be adapted to several types of data collections. There exist for instance, approaches like the one conceived by Chen et al. [25] where the authors use topic models for identifying patterns on genomic data, others like [26] apply topic models to enable the automatic annotation of images, and others such as [27] discuss the application of these models for analyzing the relationship graph from a social network.

A common application of probabilistic topic model algorithms is the categorization of text collections into groups of documents sharing similar topics, thus enabling a sort of semantic indexing over the items of the collection. The work addressed in this report aims to apply this process of categorization on the textual information extracted from a corpus of Web service description documents, by using a topic model called *online LDA* (*Latent Dirichlet Allocation*) which enable the incremental categorization of a continuous stream of documents. In this particular case, documents contain information regarding each operation or resource from a service collection. In the following sections it is addressed the description of the LDA topic model, and how it was adopted within this research.

4.2.1 The Latent Dirichlet Allocation (LDA) topic model

LDA is one of the most basic and widely adopted probabilistic topic models used for identifying the abstract themes that pervade a documentary corpus [28]. In statistical terms, LDA in one of the so-called generative models, which allows sets of observations (i.e. documents) to be explained by hidden or latent variables (topics).

The intuition behind LDA is to assume that documents in the corpus deal with not only one but several topics. This paper for example is about *Web services*, *Semantics*, *Topic models*, and *REST*. LDA estimates that documents are generated through the random process outlined below (see Figure 6):

- LDA assumes:
  - Each topic is a distribution over the vocabulary composing the whole document collection, excluding stop words such as "*and*", "*or*", "*the*", "*but*", "*of*", etcetera. Thus, for instance, in the "*Web service*" topic

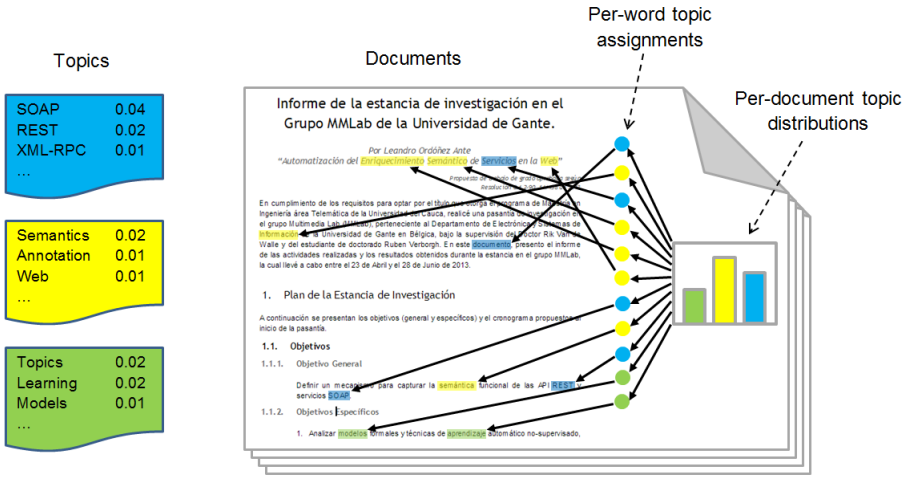

Figure 6: Generative process for LDA. Adapted from: [24]

(the blue one in Figure 6), terms like "SOAP", "REST", and "XML-RPC" would have high probability of occurrence.

- The corpus has a finite number of topics (K), which are specified before the documents have been generated.
- Word order in each of the documents is not relevant: the documents are modeled as *bags-of-words*.

- The generative process for each document in the collection is as follows:

  1. Randomly choose a distribution over the topics, as the topic proportions for each document (represented in Figure 6 by the three bar histogram).
  2. for each word in the document:
     (a) Randomly choose a topic from the distribution drawn in step #1.
     (b) Randomly choose a word from the topic (distribution over words) drawn in step #2a.

According to the above process, all the documents in the corpus share the same set of topics, but each one of them has different topic proportions, given by the distribution assigned in step #1. This way the generative process satisfies the intuition behind LDA by conceiving the documents as a mixture of topics.

Nonetheless, the distributions assumed by this generative process—i.e., topics, per-document topic proportions and per-word topic assignments—are unknown a priori; the only observable variable is the collection of document, as shown in Figure 7. The aforementioned distributions define hidden variables

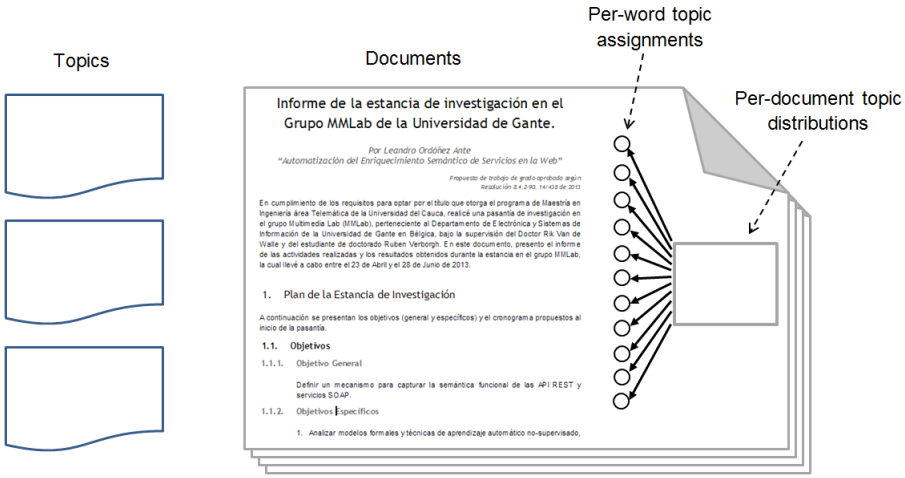

Figure 7: Observable and hidden variables of the generative process of LDA. Adapted from: [24]

which could be inferred by "reversing" the LDA generative process, allowing to estimate the hidden thematic structure that generates the observed document collection.

The LDA generative process defines a joint distribution over the observable and hidden variables, from which it is possible to derive the posterior distribution—i.e., conditional distribution of hidden variables given the observed—representing the thematic structure of the document collection.

4.2.2 Formal Definition of LDA

Figure 8 shows a probabilistic graphical model for LDA.

In this graphical model:

- β represents the K topics that run through a collection of D documents.
- Each β_k is a distribution over the corpus vocabulary.
- The per-document topic distribution is parametrized by α (θ $Dir(\alpha)$) [5].
- θ_d represents the topic distribution of the d-th document, being $\theta_{d,k}$: proportion of topic k on d.
- $Z_{d,n}$ represents the topic assignment for the n-th term in document d (d contains N terms).
- W_d is the set of terms included in document d, being $W_{d,n}$: the n-th word in d.

[5] $Dir(\alpha)$: Dirichlet distribution parametrized by α.

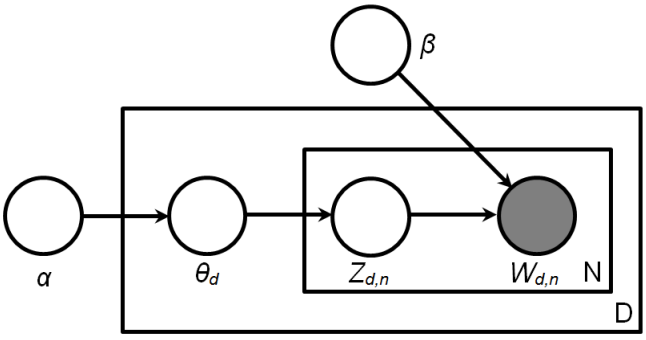

Figure 8: LDA topic model—unshaded nodes represent hidden variables; the remaining (shaded) node represents the words in each document as the only observable variable. Adapted from: http://en.wikipedia.org/wiki/Latent_Dirichlet_allocation

As stated before, the only observable variable in the LDA topic model is the collection of documents, namely W. Other variables (β, θ, Z), are inferred by analyzing the joint distribution of the LDA generative process, defined as:

$$p(\beta_{1:K}, \theta_{1:D}, Z_{1:D}, W_{1:D}) = \prod_{i=1}^{K} p(\beta_i) \prod_{d=1}^{D} p(\theta_d)(\prod_{n=1}^{N} p(Z_{d,n} \mid \theta_d) p(W_{d,n} \mid \beta_{1:K}, Z_{d,n})) \tag{1}$$

From the joint distribution in Eq. (1), it is derived the conditional distribution of β, θ, Z given W (posterior distribution):

$$p(\beta_{1:K}, \theta_{1:D}, Z_{1:D} \mid W_{1:D}) = \frac{p(\beta_{1:K}, \theta_{1:D}, Z_{1:D}, W_{1:D})}{p(W_{1:D})} \tag{2}$$

In the previous expression, the denominator $p(W_{1:D})$ is the marginal distribution of the observations, that is to say, the probability of observe the document collection under any thematic structure. Given that the number of possible thematic structures is exponentially large [28], such a marginal distribution is intractable to estimate, so the posterior distribution in Eq. 2 can only be computed through approximation algorithms.

In general terms those algorithms fall into two categories: Sampling-based algorithms (e.g.: Gibbs sampling [29]) and Variational Bayesian methods, these latter usually being more effective than its sampling-based counterpart, while offering comparable precision. Variational methods posit a parametrized family of distributions over the structure of hidden variables of the model and subsequently deduce the closest member of such family to the posterior distribution in Eq. 2, by using the *Kullback-Leibler* divergence measure [30]. This way, the posterior estimation which was originally an inference problem, becomes now an optimization problem.

4.2.3 Application of LDA over information extracted from services documentation

The previous section addressed a study of the LDA topic model, which allow to define the hidden topics that run through a collection of documents. Topics in LDA may be devised as categories that comprises semantically-related elements. This categorization, as opposed to traditional clustering techniques (like *K-means*) is non-exclusive since a document from the collection may belong to multiple categories[6]. This way, LDA enables to expose useful relations within the collection not only at the documents level but at the topics level too.

The approach documented herein, comprises the application of the LDA model on the information extracted from the Web services documentation (according to the process outlined in section 4.1), for automatically categorizing and annotating it. Given that the service documentation focuses on describing the service operations (for SOAP and XML-RPC services) and services resources (for REST services), it was decided to generate a document for each operation/resource, containing the information related to it, that is, the categorization is performed at the operation/resource level.

Additionally, this approach proposes that categorization should be performed incrementally, so that the derived structure of categories gets updated as new documents (service operations/resources) become available, operating in an *online learning* setting. However, the traditional LDA model doesn't support this online setting, so it was necessary to use an online variant of this model proposed by Hoffman et al. [31], which uses a variational Bayesian algorithm to estimate the posterior distribution in Eq. 2, from a stream of documents handled in small-sized batches. The authors of this online LDA variant evaluated the performance of their algorithm on a set of 3.3 million of randomly chosen articles from Wikipedia, and they proved that it is more precise and faster than the traditional LDA model.

In section 5 we describe a prototype developed based on out proposal, which uses the online LDA algorithm for categorizing the documentation of a corpus of 200 real service descriptors extracted from the research dataset collected by Zhang et al. [32] available at http://www.wsdream.net/dataset.html.

4.3 *KNOWEB-S*: an RDF taxonomy for representing the derived service categorization

The third of the processes that shape the solution proposed in this approach, involves the construction of a taxonomy that formally encodes and represents the category structure generated by applying online LDA on Web services documentation. This taxonomy has been called *KNOWEB-S* (*KNOwledge representation for WEB Services*). For specifying KNOWEB-S it was decided to use the data model that RDF embodies [33], given the reasons listed below:

[6] Theoretically, all documents belong to all categories but, only few categories have a meaningful proportion of each document content.

- *Simplicity*: RDF allows to describe resources as statements in the form (*subject*, *predicate*, *object*) or (*resource*, *property*, *value*) called *triples*.
- *Expressivity*: RDF offers an extensible language that allows to define classes and properties for specifying taxonomies and simple ontologies.
- *Standardization*: RDF is one of the so-called Semantic Web Standards, also being part of the recommendations from the *World Wide Web Consortium* (W3C).
- *Wide adoption*: the RDF data model has been widely accepted by the scientific community and recognized by the industry for enabling integration and interoperability among services and applications.

Additionally, RDF features SPARQL [34] as a query language for manipulating the information stored as RDF triples. Like RDF, SPARQL is also a W3C standard, recognized as one of the key technologies of the semantic Web.

This way, RDF provides for KNOWEB-S: a formal representation in a machine readable format; storage as a set of RDF triples (RDF graph); and a query language for searching and manipulating the information encoded in the taxonomy. The section below describes the specific data model of KNOWEB-S, designed to represent the entities and relations that shape the taxonomy.

4.3.1 KNOWEB-S Data Model

The KNOWEB-S taxonomy arranges a set of categories which cluster semantically related documents (Web services operations or resources). Those categories are in turn defined as a sequence of representative terms. According to this definition, the entities and relations that make up the taxonomy are:

- Entities:
  - *Document*: consists of information regarding a Web service operation or resource.
  - *Category*: groups related documents.
  - *Term*: defines a semantic unit associated to a particular category.
- Relations:
  - *Is member of*: between *Document* and *Category*.
  - *Has term*: between *Category* and *Term*

According to the LDA model used to perform the categorization, a document may belong to multiple categories depending on its per-category proportions. This way the "*is member of*" relation has a many-to-many (N:M) cardinality, and is weighted by the proportions of categories in each one of the documents, turning this into a non binary relation.

In the same way, the "*has term*" relation between categories and terms is non binary—since it is weighted by the probability value that estimates the importance of the term to the category—and also has many-to-many cardinality.

Given these constraints, Figure 9 shows the entity-relationship diagram for KNOWEB-S:

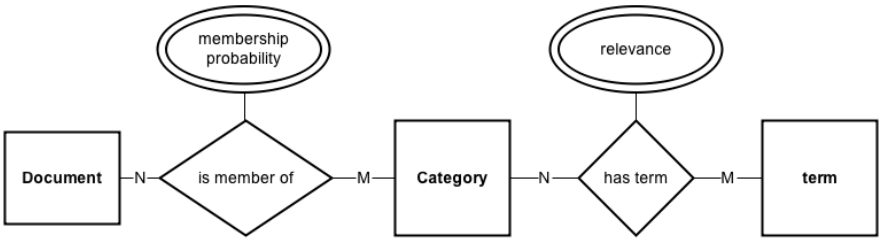

Figure 9: Entity-Relationship diagram of KNOWEB-S.

The previous diagram may be interpreted as follows: "*Documents are members of* multiple *Categories*, with certain *membership probability*. In turn, *Categories have* multiple *terms*, each one of them with a certain *relevance*".

Next section details how to specify the KNOWEB-S data model in RDF.

4.3.2 RDF specification of KNOWEB-S

As stated at the beginning of section 4.3, RDF allows specifying taxonomies and simple ontologies, also known as vocabularies. To achieve this, an RDF-based standard language called RDFS (*RDF Schema*) [35] is employed. RDFS defines a set of artifacts through which it is possible to specify classes and their associated properties.

That way, in specifying the KNOWEB-S data model in RDFS, the *entities* identified in the previous section are regarded as *classes* and *relations* turn into *properties*.

RDFS properties are binary relations, i.e. those that link two entities or an entity and a value. As mentioned earlier, relations of the KNOWEB-S data model are non-binary, since they link two individuals, with each link having an associated score (probability) that quantifies the relation. Therefore, there is no direct mapping of the identified relations ("*is member of*" and "*has term*") into RDFS properties. According to recommendations of the W3C [36] regarding *n-ary relations* definition in RDF, in cases like this, relations should be regarded as classes whose properties provide binary links for each attribute of the relation.

Thus, it is necessary to define two new entities and four new relations:

- New entities:
  - *Membership relation*: defines a link between one Document and one Category while specifying the probability associated to it.

  - *Term relation*: defines a link between one Category and one Term while specifying the relevance of the term to the category.

- New relations:
  - *Category value*: between *Membership relation* and *Category*
  - *Membership probability*: between *Membership relation* and a numerical value.
  - *Term value*: between *Term relation* and Term.
  - *Term probability*: between *Term relation* and a numerical value.

Figure 10 illustrates the entity-relationship diagram of KNOWEB-S, which includes these new entities and relations:

Finally, listing 1 shows the KNOWEB-S data model, using the RDFS XML syntax for specifying it.

Listing 1: RDF Specification of KNOWEB-S

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
        xmlns="http://www.topicalizer.org/web_api_model.rdf#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xml:base="http://www.topicalizer.org/web_api_model.rdf">
<rdfs:Class rdf:ID="Category"/>
<rdfs:Class rdf:ID="Operation"/>
<rdfs:Class rdf:ID="Membership_Relation"/>
<rdfs:Class rdf:ID="Term"/>
<rdfs:Class rdf:ID="Term_Relation"/>
<rdf:Property rdf:ID="is_member_of">
        <rdfs:domain rdf:resource="#Operation"/>
        <rdfs:range rdf:resource="#Membership_Relation"/>
</rdf:Property>
<rdf:Property rdf:ID="category_value">
        <rdfs:range rdf:resource="#Category"/>
        <rdfs:subPropertyOf
rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#value"/>
        <rdfs:domain rdf:resource="#Membership_Relation"/>
</rdf:Property>
<rdf:Property rdf:ID="membership_probability">
        <rdfs:range
            rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
        <rdfs:domain rdf:resource="#Membership_Relation"/>
</rdf:Property>
<rdf:Property rdf:ID="has_term">
        <rdfs:domain rdf:resource="#Category"/>
        <rdfs:range rdf:resource="#Term_Relation"/>
</rdf:Property>
<rdf:Property rdf:ID="term_value">
        <rdfs:range rdf:resource="#Term"/>
        <rdfs:subPropertyOf
rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#value"/>
        <rdfs:domain rdf:resource="#Term_Relation"/>
</rdf:Property>
<rdf:Property rdf:ID="term_probability">
        <rdfs:range
            rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
        <rdfs:domain rdf:resource="#Term_Relation"/>
</rdf:Property>
<rdf:Property rdf:ID="has_content">
```

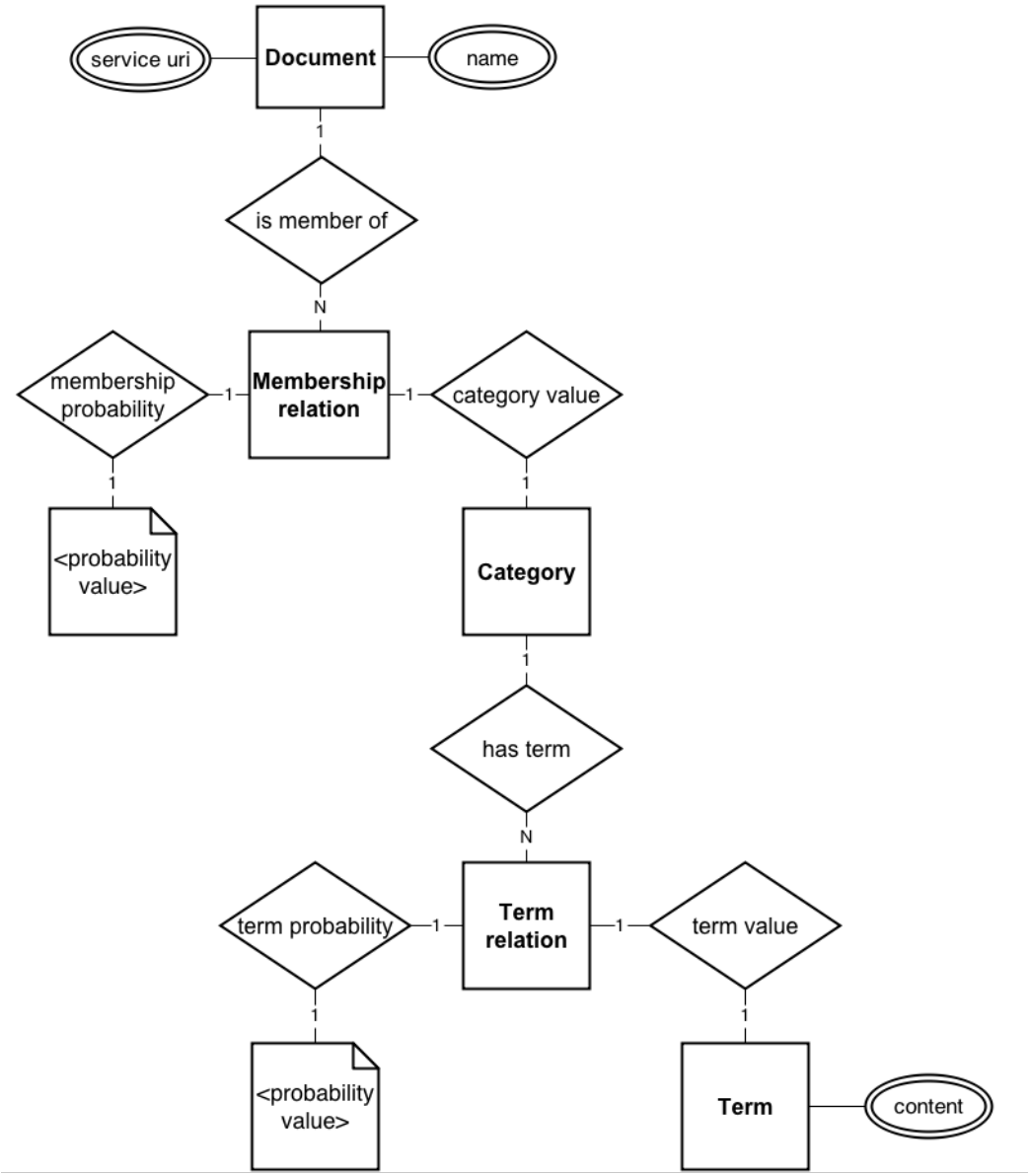

Figure 10: Entity-relationship diagram of KNOWEB-S adapted to RDFS.

```
        <rdfs:range
            rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <rdfs:domain rdf:resource="#Term"/>
</rdf:Property>
<rdf:Property rdf:ID="has_name">
        <rdfs:range
            rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <rdfs:domain rdf:resource="#Operation"/>
</rdf:Property>
<rdf:Property rdf:ID="has_service_uri">
        <rdfs:range
            rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI"/>
        <rdfs:domain rdf:resource="#Operation"/>
</rdf:Property>
</rdf:RDF>
```

5 Experimentation

In order to assess the feasibility of our proposal we build a prototype called *Topicalizer*, which implements the mechanisms described throughout this report, for the particular case of SOAP services.

The implemented system receives as input a list of URIs from real WSDL interfaces available online. The system retrieves each WSDL and processes it by following the techniques outlined in sections 4.1.1 and 4.1.3 and stores its relevant information into a service registry. In running this process, a stream of documents is generated, each one of them containing information related to a specific SOAP service operation. Then the stream of documents is categorized based on the document's content by applying the *online LDA* algorithm as described in sections 4.2.1 and 4.2.3. Such categorization arranges semantically-related documents into clusters which are defined by a weighted set of terms, that in turn become annotations on the operations that each document represents. The information gathered from the categorization process is specified in RDFS—conforming to the data model defined in section **??**—and stored into an RDF triplestore.

The system architecture is depicted in Figure 11:

Each one of the components comprising the architecture of Topicalizer are described below:

- *Descriptor processing*: this module of the architecture is in charge of reading the list of WSDL URIs, accessing each descriptor via HTTP request, loading it into memory, mapping it into the SOAP service model defined in section 4.1.1 (see Figure 3) and storing the mapped information into a service registry. Once these initial procedures have been performed, this module applies tokenization and spell checking techniques—described in section 4.1.3—on the content of each descriptor and finally, for every operation of the processed descriptor it generates a document in plain text containing: (*i*) the operation name, (*ii*) its description in natural language, (*iii*) the name of the service it belongs to, and (*iv*) its input/output parameters. This module was implemented in Java, using its persistence

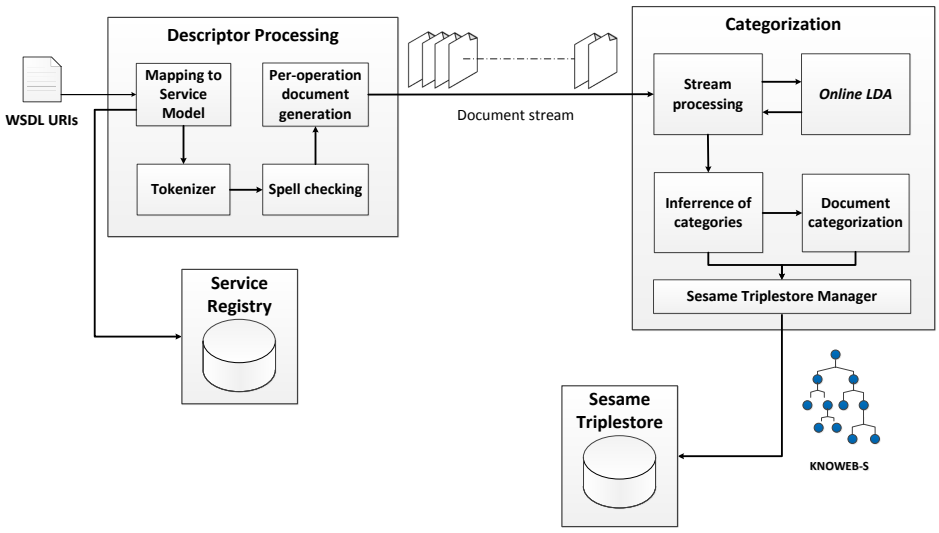

Figure 11: Architecture of Topicalizer.

API (JPA) and the *EasyWSDL* library for WSDL manipulation [7].

- *Service registry*: consists of a relational database, implemented in MySQL. The entity-relationship model of this database matches the service model described in section 4.1.1.

- *Categorization*: this component receives the stream of documents generated by the descriptor processing module, then splits it in small-sized batches and delivers them to the implementation of online LDA, which is based on an adaptation of an existing application developed by Matthew D. Hoffman, one of the authors of the mentioned algorithm [31]. Once the processing of the stream of documents is completed, the derived results are interpreted for establishing the set of terms that compose each category—task performed by the *inference of categories* sub-module—as well as the group of documents included in each one of them—which is defined by the *document categorization* sub-module. The derived arrangement of categories, documents and terms, is mapped into the KNOWEB-S data model and stored into a RDF triplestore provided by the *Sesame* open-RDF framework [8]. This component of the architecture was developed in Python, using the *numpy*, *urllib2* and *httplib2* libraries.

- *Sesame Triplestore*: It is an HTTP repository for RDF triples, hosted in an Apache Tomcat web server. This repository stores the KNOWEB-S taxonomy generated by the previous Categorization component. Sesame allows

[7] Available at: http://easywsdl.ow2.org/
[8] Available at: http://www.openrdf.org/index.jsp

the retrieval and manipulation of the information encoded in KNOWEB-S by using SPARQL queries.

The whole prototype is executed by using a bash script, which receives as input the path of a text file containing the list of WSDL URIs. The prototype also generates two documents in *csv* format for specifying: (1) the terms that define each of the identified categories along with their associated relevance value, and (2) the per-document category proportions for each of the processed documents (i.e. SOAP service operations). A description of the source code structure of the prototype is available at https://github.com/LeandroOrdonez/Topicalizer.

6 Discussion and Conclusions

The literature review conducted in developing our research, revealed the benefits of adopting the REST architectural style for building scalable distributed hypermedia systems, like the Web itself. Thanks to the HATEOAS principle of the REST uniform interface, it is possible for automated agents to carry out processes of discovery and composition of services by leveraging the hypermedia controls included in resources representations– a dynamic that emulates the way humans browse the web—. One of the goals of the Semantic Web consists precisely in enabling computers to understand the underlying meaning of resources, so that they are able to autonomously perform service discovery, invocation and composition. This way, implementing distributed systems that adopt the REST architectural style, turns out to be a practice aligned with the realization of the Semantic Web. However, recent studies has shown that despite the apparent spread of REST services/APIs on the Web, the HATEOAS principle is in most of the cases neglected or misinterpreted.

Bearing this limitation in mind, the research work documented in this paper, proposes an approach that leverages on existing Web services and their associated documentation sources for generating a knowledge representation which captures the semantics that defines them, in a machine readable format. Such a knowledge representation allows to arrange the services into an structure that reveals semantic relationships among them.

Given a characterization based on the Richardson's maturity model for web services [6], in this work three types of services are discriminated: SOAP, XML-RPC and REST. For each kind of services we analyze their associated documentation sources to define which information is relevant for setting up the above knowledge representation, as well as the procedure for extracting such information.

The knowledge representation is derived by applying an online variant of the LDA topic model on the information extracted from different Web services documentation sources. This model allows deducing a set of categories that cluster semantically-related service operations and resources. The derived structure of categories is specified by using a standard format based on the RDF data model, and stored into an RDF triplestore.

Finally, the proposed techniques supported the implementation of a prototype for categorizing a set of operations included in SOAP services available online.

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] C. Martinez-Cruz and S. Angeletou, "Folksonomy expansion process using soft techniques," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2010 Seventh International Conference on*, vol. 3, pp. 1238–1243, Aug 2010.

[18] N. Loutas, V. Peristeras, D. Zeginis, and K. Tarabanis, "The semantic service search engine (s3e)," *J. Intell. Inf. Syst.*, vol. 38, pp. 645–668, June 2012.

[19] Z. Azmeh, M. Huchard, C. Tibermacine, C. Urtado, and S. Vauttier, "Using concept lattices to support web service compositions with backup services," in *Proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services*, ICIW '10, (Washington, DC, USA), pp. 363–368, IEEE Computer Society, 2010.

[20] 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.

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

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

[23] 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.

[24] D. M. Blei, "Probabilistic topic models," *Commun. ACM*, vol. 55, pp. 77–84, Apr. 2012.

[25] X. Chen, X. Hu, T. Y. Lim, X. Shen, E. K. Park, and G. L. Rosen, "Exploiting the functional and taxonomic structure of genomic data by probabilistic topic modeling," *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, vol. 9, pp. 980–991, July 2012.

[26] Y. Feng and M. Lapata, "Topic models for image annotation and text illustration," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, (Stroudsburg, PA, USA), pp. 831–839, Association for Computational Linguistics, 2010.

[27] Y. Cha and J. Cho, "Social-network analysis using topic models," in *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, (New York, NY, USA), pp. 565–574, ACM, 2012.

[28] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[29] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 6, pp. 721–741, Nov. 1984.

[30] J. Shlens, "Notes on kullback-leibler divergence and likelihood theory," 2007.

[31] M. D. Hoffman, D. M. Blei, and F. R. Bach, "Online learning for latent dirichlet allocation," in *NIPS*, pp. 856–864, 2010.

[32] Y. Zhang, Z. Zheng, and M. R. Lyu, "Wsexpress: A qos-aware search engine for web services," in *Proc. IEEE Int'l Conf. Web Services (ICWS'10)*, pp. 83–90, 2010.

[33] W3C, "Resource description framework (rdf)," 2004.

[34] E. Prud'hommeaux and A. Seaborne, "Sparql query language for rdf," 2008.

[35] B. McBride, "Rdf vocabulary description language 1.0: Rdf schema," 2004.

[36] P. Hayes and C. Welty, "Defining n-ary relations on the semantic web," 2006.