Collecting, Annotating, and Classifying Public Web Services

Mohammed AbuJarour[1] , Felix Naumann[1] , Mircea Craculeac[2]

[1] Hasso-Plattner-Institut, University of Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de
[2] Neofonie, Berlin, Germany
mircea@neofonie.de

Abstract. The limitations of the traditional SOA operational model, such as the lack of rich service descriptions, weaken the role of service registries. Their removal from the model violates the basic principles of SOA, namely, *dynamic binding* and *loose coupling*. Currently, most service providers publish their Web Services on their websites instead of publishing them in service registries. This results in poor usability of these Web Services especially wrt. service discovery and service composition.
To handle this problem, we propose to increase the usability of public Web Services by collecting them automatically from the websites of their providers with the help of web crawling techniques. Additionally, the collected services are annotated with descriptions that are extracted from the crawled web pages and tags that are generated from the same web pages. These annotations are then used to derive a classification for each Web Service into different application domains. In this paper, we introduce the details of our approach and show its practical feasibility through several evaluation experiments.

Keywords: Web Service, Service Description, Tagging, Classification

1 SOA in Theory and Practice

Due to the increasing complexity of modern software systems, distributed computing has been the typical approach in several application domains, such as banking, medicine, earth sciences, etc. Additionally, the wide and rapid spread of Internet technology has been a major driving force that led to the emergence of new software paradigms, such as the Service-oriented Architecture (SOA) [11].

The basic principles of SOA are captured in the triangular SOA operational model [17] depicted in Figure 1 (left). Three roles are identified in this model, namely, service-provider, -consumer, and -registry. Theoretically, service providers register their services into one or more service registries that play the role of service brokers. Service consumers use these registries to find services that satisfy their needs. Afterwards, a direct binding between service consumers and service providers is done to call the required service(s).

The goal of this operational model is to achieve the main features of SOA, e.g., high interoperability, flexibility, scalability, fault tolerance, etc. However, to achieve such goals service consumers should have sufficient knowledge about the considered services. Much of this knowledge comes from service providers themselves in the form of service descriptions that they give during the registration of their services. Service descriptions play a key role in Service-oriented Computing (SOC). However, service descriptions are typically poor, because service providers focus on the implementation aspects of their services rather than providing rich service descriptions.

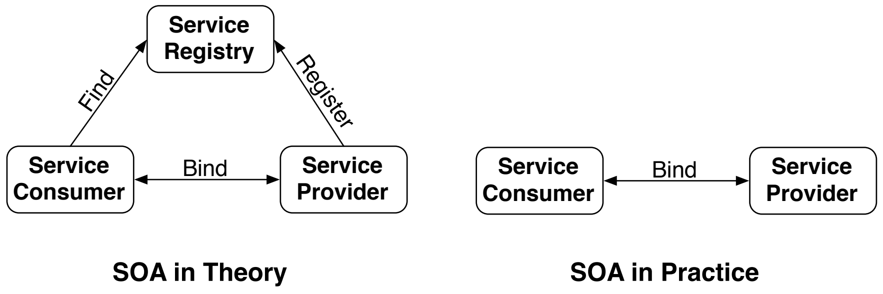


Fig. 1. SOA in theory and practice [13]

Service registries do not have control over service providers to force them to keep the descriptions about their registered Web Services up-to-date. This lack of control results in a passive and limited role of service registries. On the contrary, service providers often do keep the published information and descriptions about their offered Web Services up-to-date via their own websites.

In practice, the role of service registries is typically ignored [13] because of such limitations. For instance, 48% of a production `UDDI` registry has links that are unusable (tModels tested only); these links point to contain missing, broken or inaccurate information [4]. Removing the role of the service registry (and thus breaking the triangle) violates the basic SOA principles of *loose coupling* and *dynamic binding* (cf. Fig. 1).

A common approach used by service providers to offer public Web Services is to provide a list of services with a textual description attached to each service to explain their functionality. Moreover, several service providers offer a *try* option, where one can try their services online. Such options typically include web forms to collect input parameters from the user. Much of the information provided on such web pages is not provided in the `WSDL` description of the corresponding Web Service. We view such `HTML` pages and forms as rich sources of information and descriptions about the considered Web Services.

For instance, Amazon.com provides several public Web Services, e.g., Amazon Relational Database Service (Amazon RDS). Typically, Amazon provides a detailed description for each of their Web Services in the form of `HTML`. For

example, Amazon RDS is described in detail on `https://aws.amazon.com/rds`. Much of this information, such as the fact that it is based on MySQL, is not provided in its `WSDL` description available at
`https://rds.amazonaws.com/doc/2010-01-01/AmazonRDSv2.wsdl`.
In our approach, we extract the information that service providers release about their offered Web Services on their websites and generate richer service descriptions from this information.

We propose an approach where service registries take the initiative to collect public Web Services from the websites of their providers through web crawling techniques, extract descriptions for these Web Services, generate tags associated with the collected Web Services from the contents provided by their service providers, and classify the collected Web Services into several application domains based on their enriched annotations. Our Web Service crawler is limited to so-called Big Web Services (`SOAP`-based) only. Further decision rules are required to incorporate RESTful Web Services.

In [1], we introduced a system that enables users to discover, aggregate, and consume public Web Services. However, we have observed that choosing the best services that match some users' requirements from the service catalog requires additional information to enhance the quality of the result list. This paper tackles this problem and serves as an extension to our previous work.
The main contributions of this paper are:

– Automatic collection of public Web Services over the web.
– Automatic enrichment of poor service descriptions with extracted descriptions from the `HTML` pages of their providers.
– Automatic generation of tags that are associated with collected Web Services.
– Supervised classification of collected Web Services.

The remainder of this paper is organized as follows: In Section 2, we give an overview of the related work. A use-case is introduced in Section 3 to highlight the targeted problems. In Section 4, we introduce our approach in detail. Section 5 shows our experiments, and a summary and future work are given in Section 6.

2 Related Work

The limitations in the traditional SOA operational model have been highlighted by several researchers in the community. In [13], the authors showed that the triangular model is not used widely in practice because of the limited role of service registries. Removing the role of the service registry (breaking the triangle) violates the basic principles of Service-oriented Computing (SOC), namely *loose coupling* and *dynamic binding*. To restore this broken triangle, the authors proposed a software engineering approach that enables dynamic binding and invocations of Web Services. Our approach tackles the limited role of service registries problem and aims at increasing the active role of service registries in SOC.

Another approach to achieve active Web Service registries was introduced in [16]. The authors use `RSS` feeds to announce changes in the registered services to interested service consumers. The information provided by such feeds is generated by service providers, who tend to focus on the technical part of their services rather than providing documentation or descriptions. Moreover, such a service registry cannot force service providers to notify them about any updates so that they can add `RSS` feeds to announce the corresponding changes.

The main reason behind the highlighted limitations of the triangular model in the aforementioned work is the lack of rich service descriptions [12]. Therefore, researchers have proposed several approaches to gather additional information about Web Services to handle the problem of poor service descriptions.

In [2], the authors use a specialized crawler to collect Web Services from multiple `UDDI` registries. Although the idea of using crawlers to collect Web Services is innovative, restricting it to `UDDI` registries does not give the maximum benefit of web crawling, as such an approach is still limited to what service providers announce during service registration.

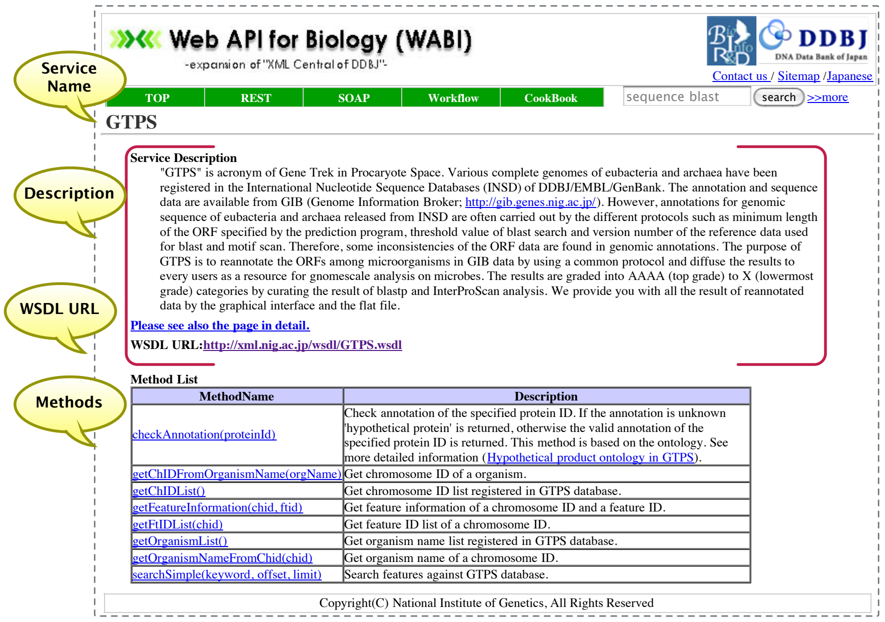
A more advanced Web Service crawler has been introduced by the EU project Seekda (`http://www.seekda.eu`). In this recent project, the authors use a specialized crawler to collect public Web Services over the web, and present them in a web 2.0 environment, which allows users to annotate, tag, and use the collected Web Services. We extend this approach by annotating the collected Web Services with automatically extracted descriptions and generate tags, as we show in Section 4.2.

The automatic assignment of classes to Web Services is known as service classification. This problem is vital in SOC, because of the increasing number of Web Services. Therefore, it has been investigated by several researchers in the community. Typically, machine learning techniques are used to perform automatic service classification, where different approaches are based on argument definitions matching [6], document classification techniques [8], or semantic annotation matching [5]. In most of these approaches, the authors classify formal service descriptions, e.g., `WSDL`, or they assume the presence of additional information, e.g., ontology annotations. Service descriptions appear mostly in the form of comments written by service developers [15]. In general, these comments are written in English, have a low grammatical quality, punctuation is often ignored, and several spelling mistakes and snippets of abbreviated text is present. Assuming the existence of ontology annotations is not realistic [12]. In our approach, we use formal service descriptions provided by service providers, e.g., `WSDL`, in addition to annotations that we extract from the websites of the providers to apply a machine learning approach to classify Web Services.

3 Example: Gene Trek in Procaryote Space (GTPS)

Gene Trek in Procaryote Space (GTPS) is a service offered by the National Institute of Genetics (NIG) in Japan. The purpose of GTPS is to re-annotate the ORFs[1] among microorganisms in Genome Information Broker data by using a common protocol and diffuse the results to users as a resource for gnome-scale analysis on microbes.

The GTPS Web Service is published with several additional services on the website of the NIG under `http://xml.nig.ac.jp`. Figure 2 shows a screenshot of the web page that shows the details of the GTPS Web Service. Along with the name of the service, they provide a service description, a hyperlink to the `WSDL` file of this service, and a list of methods it offers. The service name is obviously provided in its `WSDL` file, but, the description is not. The hyperlink to the `WSDL` file represents a candidate place to start searching for service descriptions.



GTPS

Service Description

"GTPS" is acronym of Gene Trek in Procaryote Space. Various complete genomes of eubacteria and archaea have been registered in the International Nucleotide Sequence Databases (INSD) of DDBJ/EMBL/GenBank. The annotation and sequence data are available from GIB (Genome Information Broker; http://gib.genes.nig.ac.jp/). However, annotations for genomic sequence of eubacteria and archaea released from INSD are often carried out by the different protocols such as minimum length of the ORF specified by the prediction program, threshold value of blast search and version number of the reference data used for blast and motif scan. Therefore, some inconsistencies of the ORF data are found in genomic annotations. The purpose of GTPS is to reannotate the ORFs among microorganisms in GIB data by using a common protocol and diffuse the results to every users as a resource for gnomescale analysis on microbes. The results are graded into AAAA (top grade) to X (lowermost grade) categories by curating the result of blastp and InterProScan analysis. We provide you with all the result of reannotated data by the graphical interface and the flat file.

Please see also the page in detail.

WSDL URL:http://xml.nig.ac.jp/wsdl/GTPS.wsdl

Method List

MethodName	Description
checkAnnotation(proteinId)	Check annotation of the specified protein ID. If the annotation is unknown 'hypothetical protein' is returned, otherwise the valid annotation of the specified protein ID is returned. This method is based on the ontology. See more detailed information (Hypothetical product ontology in GTPS).
getChIDFromOrganismName(orgName)	Get chromosome ID of a organism.
getChIDList()	Get chromosome ID list registered in GTPS database.
getFeatureInformation(chid, ftid)	Get feature information of a chromosome ID and a feature ID.
getFtIDList(chid)	Get feature ID list of a chromosome ID.
getOrganismList()	Get organism name list registered in GTPS database.
getOrganismNameFromChid(chid)	Get organism name of a chromosome ID.
searchSimple(keyword, offset, limit)	Search features against GTPS database.

Copyright(C) National Institute of Genetics, All Rights Reserved

Fig. 2. The web page of the Gene Trek in Procaryote Space (GTPS) Web Service

A simplified version of the `HTML` source code of the same web page is shown in Figure 3. An important aspect in this code is the relationship between the element that holds the description (line 3) and the element that holds the `WSDL` hyperlink (line 13). These two elements usually have either the *sibling* or *parent-child* relationship. Typically, service providers describe their Web Services and

[1] ORF stands for Open Reading Frame, which is a DNA sequence that could potentially encode a protein

then provide their links, or provide the links to their Web Services and then describe their functionalities. Other cases include more complex relationships, especially, when tables are used. In our example, the simple *sibling* relationship holds.

```
<div class=message_indent>
    <div class=itemTitle>Service Description</div>
     <div class=itemContent>
      "GTPS" is acronym of Gene Trek in Procaryote Space. Various complete genomes of eubacteria and archaea have been
      registered in the International Nucleotide Sequence Databases (INSD) of DDBJ/EMBL/GenBank. The annotation and sequence
      data are available from GIB (Genome Information Broker; <a href="http://gib.genes.nig.ac.jp/">http://
      gib.genes.nig.ac.jp/</a>). However, annotations for genomic sequence of eubacteria and archaea released from INSD are
      often carried out by the different protocols such as minimum length of the ORF specified by the prediction program,
      threshold value of blast search and version number of the reference data used for blast and motif scan. Therefore, some
      inconsistencies of the ORF data are found in genomic annotations. The purpose of GTPS is to reannotate the ORFs among
      microorganisms in GIB data by using a common protocol and diffuse the results to every users as a resource for
      gnomescale analysis on microbes. The results are graded into AAAA (top grade) to X (lowermost grade) categories by
      curating the result of blastp and InterProScan analysis. We provide you with all the result of reannotated data by the
      graphical interface and the flat file.
     </div>
     <div class=itemTitle>
         <a href="http://gtps.ddbj.nig.ac.jp/" target=_blank>
           Please see also the page in detail.
        </a>
     </div>
     <div class=itemTitle>
         WSDL URL:
        <a href="http://xml.nig.ac.jp/wsdl/GTPS.wsdl">
           http://xml.nig.ac.jp/wsdl/GTPS.wsdl
        </a>
     </div>
</div>
```

Fig. 3. Part of the HTML source of the web page of the Gene Trek in Procaryote Space (GTPS) Web Service shown in Figure 2

An interesting observation is the lack of similar documentation or descriptions in the WSDL file of the GTPS Web Service.[2] This situation makes it difficult for biologists, who do not have sufficient technical background, to use such important Web Services. Moreover, discovering such a Web Service is not an easy task, unless, it is augmented with this description.

By applying our approach to this URL (`http://xml.nig.ac.jp`), we were able to find 23 Web Services, and we were able to extract the descriptions provided in the form of HTML, and generate tags that describe the extracted Web Services. The list of tags for the GTPS Web Service is `[id, chromosome, annotation, list, chid]`. Attaching these annotations to this Web Service increases its usability, because it becomes more convenient for biologists to find and use it.

4 Architecture Overview

Our proposed approach to increase the usability of public Web Services incorporates four components: a Web Service crawler, a Web Service parser, an annotation extractor, and a Web Service classifier. The architecture of our proposed approach is depicted in Figure 4.

[2] Available at: `http://xml.nig.ac.jp/wsdl/GTPS.wsdl`

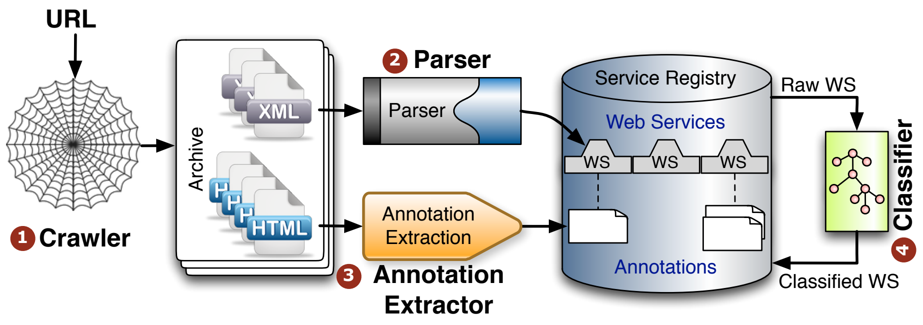


Fig. 4. An overview of the architecture of our approach

In our approach, a service registry takes the initiative to collect public Web Services from the websites of their providers. This task is achieved using a Web Service crawler (component number 1 in Fig. 4). The main task of this crawler is to collect `XML` and `HTML` resources and store them in a special archive. The collected `XML` resources are then parsed and validated through a Web Service parser (component number 2) to identify valid Web Services. Valid Web Services are stored in the service registry. Further details about the crawler and the parser are given in Section 4.1.

The annotation extractor (component 3 in Fig. 4) has the task of analyzing the collected `HTML` resources by the crawler to extract annotations that enrich the descriptions of the collected Web Services. The extracted annotations are stored in the service registry. Two types of annotations are generated by this component: service descriptions and tags. We give further details about this component in Section 4.2.

The annotations generated by the annotation extractor are used to classify the collected Web Services into several application domains, e.g., education, telecommunications, finance, government, etc. This task is performed by a service classifier (component number 4 in Fig. 4). Further details about this component are given in Section 4.3.

4.1 Crawling and Parsing Public Web Services

To collect public Web Services, we employ web crawling techniques to the web. We have implemented a focused crawler that targets `XML` and `HTML` resources only. `XML` files are potential candidates for Web Services descriptions, e.g., `WSDL`, whereas, `HTML` files are potential places to find further information about the collected Web Services. In this section, we consider `XML` files only. `HTML` files are considered in the following section.

The architecture of our Web Service crawler is shown in Figure 5. We use the `Heritrix` [10] crawling framework in our approach. The client on the left hand side starts the crawler by providing a seed `URI` that has to be crawled. The crawler crawls the entire domain of the given seed `URI` for `WSDL` documents.

If it is able to gather `WSDL` files from that domain, then it stores the collected files into a compressed archive file, called ARC file. This ARC file is passed to the `WSDL` parser that extracts the found `WSDL` documents. Then, each of these `WSDL` documents is parsed with the WSDL4J API [9] to extract its port types, bindings, operations and descriptions if available. If the `WSDL` file is valid and no parsing errors occurred, the file is stored in the Web Service registry that can be directly accessed by the user.

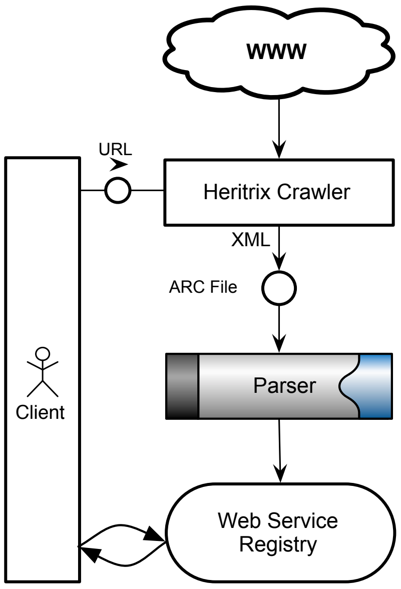


Fig. 5. The architecture of our Web Service crawler

The implementation of our focused crawler incorporates a series of "DecideRules" to filter `URIs`. To determine whether a `URI` falls within the required scope, all decision rules are applied to it. Accepted `URIs` should pass all checks represented by the decision rules. The most important DecideRule that are used are:

- **PathologicalPath**: This rule is used to avoid crawler traps. It adds a constraint on how many times a path-segment pattern in the `URI` may be repeated. The `URI` is rejected if the pattern is repeated more than two times, e.g., "`www.example.com/foo/foo/foo`" is rejected.
- **MatchesRegExp**: The goal of this rule is to filter all resources that are not relevant to our crawling task. Because our goal is to identify `WSDL` files, we discard all crawled resources except `XML` and `HTML` documents. `XML` files could represent `WSDL` files and `HTML` files could contain `URIs` to `WSDL` files, in addition to descriptive text that explains the functionality of the Web

Services in the linked `WSDL` files. All other document types, such as audio, video, image files, etc. are ignored by the crawler using the regular expression provided by this rule.

- **WSDLRegExp:** This second regular expression explicitly accepts all `URIs` ending with the case insensitive phrase "`wsdl`".

4.2 Annotating Web Services

In the previous section, we explained the process of collecting public Web Services over the web using a focused crawler by identifying their formal descriptions that have `XML` as a content type. The collected Web Services are then parsed and validated to identify valid Web Services. This step is then followed by a step to extract further information about the collected Web Services from the crawled `HTML` pages. In this section, we show the types of this additional information that we extract and the techniques we use to achieve this step.

Because of the lack of rich service descriptions, we propose to extract further information about public Web Services from the websites of their providers to enrich poor service descriptions. This extraction is achieved by the annotation extractor component (cf. Fig. 4). The process of annotation extraction is shown in Figure 6.

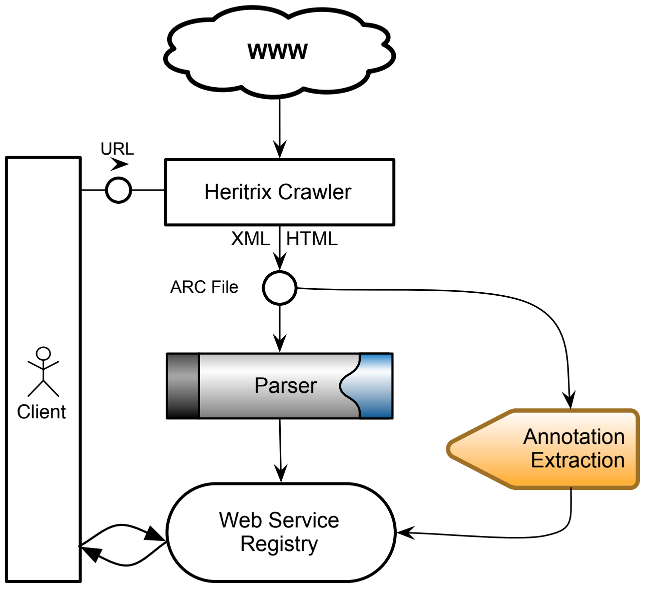


Fig. 6. The Architecture of our Web Service crawler and its annotation extractor (`HTML` parser)

According to our set of "DecideRules", the resulted ARC file contains URIs whose content type is either XML or HTML. We perform two iterations of parsing on this ARC file: the first iteration is done by the WSDL parser to identify WSDL URIs (cf. Sec. 4.1), and the second iteration is done by the annotation extractor to identify further information about the identified WSDL URIs.

The result of this recent step is formalized in Equation 1. Given a valid URL where some Web Services are offered, the expected outcome is a list of collected Web Services ({ws}), extracted descriptions for each Web Service (Desc(ws)), and a list of tags associated with that Web Services (Tags(ws)).

$$ExtraWS(url) = \{\{ws, Desc(ws), Tags(ws)\}\} \tag{1}$$

The extracted description for each of the collected Web Services, Desc(ws), is captured in Equation 2. In our approach, we use the heuristic that the place where the WSDL of a Web Service is referenced represents a good candidate place where its description can be found. Typically, service providers describe their Web Services and then provide their links, or provide the links and then describe what they do. According to our experiments, this heuristic works in more than 90% of our crawled URLs. In the remaining 10%, service providers either do not provide textual descriptions on their web pages or they use complex HTML pages, such as Amazon Web Services, where our heuristic did not find the appropriate text. Identifying the templates used in such complex HTML structures and smart extraction of such textual descriptions are part of our future work. This heuristic is also useful in the case where a web page describes more than one Web Service. Different descriptive texts can be extracted from the same web page for different Web Services. However, such Web Services are annotated with the same tags, unless, they are described in other web pages.

The content of an HTML page that references a WSDL file is parsed as a DOM tree of HTML elements. The content of an HTML element represents the concatenation of its textual content with the textual content of its sub-elements. The content of the root element represents the entire text in the HTML page. We refer to the element that contains a link to a WSDL file as (e) in Equation 2. The element that contains the description is referred to as (d). If element (d) has either the *sibling* or the *parent* relationship (R) with (e), the content of element (d) represents the extracted description of the considered Web Service (ws).

$$Desc(ws) = content(d)|R(d, e) \in \{sibling, parent\} \tag{2}$$

Web Services can be referenced from multiple web pages. A descriptive text can be extracted from each web page. In our approach, such Web Services can have several extracted descriptive texts for each Web Service.

The remaining parts of an HTML page that references a Web Service have their impact on the generated tags for the considered Web Service. Equation 3 captures this impact. A list of tags for a Web Service contains the most frequent terms (t) in the HTML page (p) where this considered Web Service (ws) is referenced. All terms in the web page are ranked and the most frequent k terms represent its list of tags, where `k` is an application parameter, e.g., `k`=5 (cf. Sec. 5.2). Common terms are ignored through stop words removal.

$$Tags(ws) = \{t | t \in rank(content(p))\} \quad (3)$$

However, this tag generation method can miss some important keywords. For instance, the tags of GTPS (cf. Table 2) do not include "GIB data", "microbes", "Converts", or "US" as tags. These keywords are important, but they do not appear frequently in the text. Moreover, tags give hints to service consumers about the managed Web Services because selecting proper keywords for service discovery requires domain knowledge. Such important keywords are still indexed and involved in the Web Service discovery process.

The main limitation of this approach is the use of a single source of information about Web Services, namely, the websites of their service providers. Service providers may not provide textual descriptions about their offered Web Services on their websites. In this case, our approach can only identify and collect such Web Services, but it cannot annotate or classify them. Further information about such Web Services is still required to increase their usability.

4.3 Classifying Web Services

Gathering Web Services through the crawler results in a large list of Web Services that do not have categories. This situation makes it difficult to browse through these Web Services and retrieve the relevant ones upon request. To handle this issue, we use an automatic classifier that classifies the collected Web Services into a set of predefined categories, e.g., finance, education, entertainment, etc. We have compiled this list of categories from well-known service providers and service registries on the web, such as `http://www.programmableweb.com`. A complete list of categories is shown in Section 5.3.

Machine learning techniques are used in our approach to classify Web Services, where the features used by the classifier are the `WSDL`, *description*, and *tags* of each Web Service. The name of the Web Service is not included explicitly, because it is included in the `WSDL` file. The size of some features can grow unexpectedly or cannot be predicted in advance. The `WSDL` file of a Web Service can have a few lines or thousands of lines. The generated tags are domain-specific and cannot be expected in advance. To meet this challenge, we hash the contents of each feature using the `simHash` [3] algorithm into a fixed length digest. `SimHash` produces similar hash values for similar documents. Therefore, it combines the advantages of the word-based similarity measures with the efficiency of fingerprints based on hashing.

Most similar approaches (cf. Sec. 2) use `WSDL` files to classify Web Services, but such files contain – mainly – technical descriptions. Using the extracted descriptions and tags to classify Web Services is one of the main contributions of our approach. Common terms, e.g., articles, pronouns, etc., in service descriptions and `WSDL` files are removed through stopword removal techniques before applying the `simHash` algorithm to get the digest of each feature. Additionally, the `snowball` stemmer [14] is used to capture the similarity between several derivations of the same word, e.g., walk, walk*ing*, walk*s*.

Our machine learning approach uses a supervised classifier that is trained on a set of manually classified Web Services. Then, it is tested on the newly collected unclassified Web Services to infer their classes based on the training set. The features of each Web Service (i.e., the digests of the `WSDL`, description, tags) are then used to infer classes for the unclassified Web Services from the already (manually) classified ones. The Weka [7] tool has been used to achieve this goal. Further technical specifications are given in Section 5.3.

5 Experiments and Evaluation

We have implemented a prototype that achieves the aforementioned contributions. To evaluate its feasibility, we have carried out three sets of experiments. A set of experiments to evaluate our focused crawler, another set of experiments to evaluate the extracted annotations, and a third set of experiments to evaluate the classifier. In this section, we describe these experiments, show the results, and discuss them.

5.1 Evaluating the Service Crawler

To test our Web Service crawler, we selected a number of service providers and crawled their websites using our crawler. We compare the number of collected Web Services from each service provider with the number of Web Services found by `seekda` (cf. Section 2) on the same website.

Table 1 shows a list of `URLs` of service providers (second column), the number of services found by `seekda` (third column), and the number of Web Services found using our crawler (fourth column).

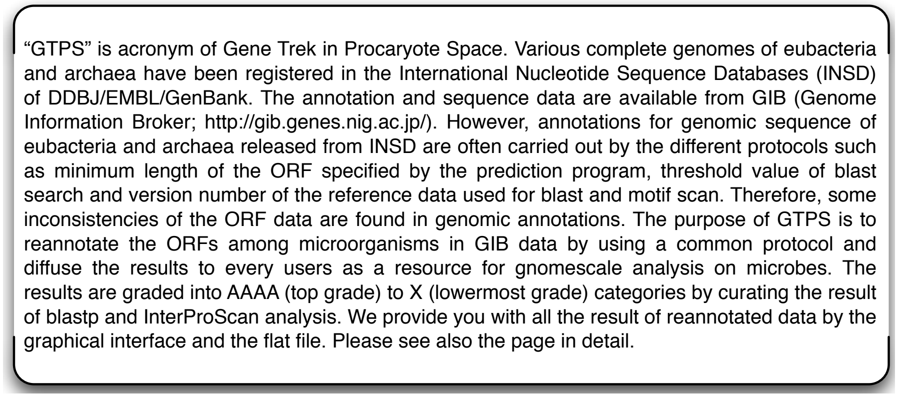
Table 1. Number of collected Web Services per `URL`

ID	Service Provider URL	Seekda Result	Our Crawler
1	`webservice.genotec.ch`	8	8
2	`api.bioinfo.no`	8	9
3	`www.servicex.co.uk`	9	9
4	`xml.nig.ac.jp`	2	23
5	`www.ecocoma.com`	25	25
6	`ws.adaptivedisclosure.org`	26	26
7	`ws.serviceobjects.com`	31	31
8	`www.webservicex.net`	70	70
9	`www.strikeiron.com`	59	72
10	`splices.xignite.com`	58	190

In most cases, we are as good as `seekda` in finding published Web Services on a website. For instance, `URLs` in case 1,3,5–8. In other cases, e.g., case 2, 4, 9, 10, we managed to find more Web Services than `seekda`. One potential reason for this behavior, can be the release of new Web Services on a `URL` after `seekda` had crawled it. This situation requires incremental and iterative crawling of `URLs` to keep the list of collected Web Services up-to-date.

5.2 Evaluating the Extracted Annotations

Evaluating the extracted service descriptions is not straightforward. It requires manual checking to determine their quality. Due to space limitations, we show only the extracted service description for the example Web Service, GTPS introduced in Section 3.



"GTPS" is acronym of Gene Trek in Procaryote Space. Various complete genomes of eubacteria and archaea have been registered in the International Nucleotide Sequence Databases (INSD) of DDBJ/EMBL/GenBank. The annotation and sequence data are available from GIB (Genome Information Broker; http://gib.genes.nig.ac.jp/). However, annotations for genomic sequence of eubacteria and archaea released from INSD are often carried out by the different protocols such as minimum length of the ORF specified by the prediction program, threshold value of blast search and version number of the reference data used for blast and motif scan. Therefore, some inconsistencies of the ORF data are found in genomic annotations. The purpose of GTPS is to reannotate the ORFs among microorganisms in GIB data by using a common protocol and diffuse the results to every users as a resource for gnomescale analysis on microbes. The results are graded into AAAA (top grade) to X (lowermost grade) categories by curating the result of blastp and InterProScan analysis. We provide you with all the result of reannotated data by the graphical interface and the flat file. Please see also the page in detail.

Fig. 7. The extracted service description for the GTPS Web Service from `http://xml.nig.ac.jp/wabi/Method?serviceName=GTPS&mode=methodList&lang=en`

The `GTPS` Web Service is intended to be used by biologists and not IT specialists. Its `WSDL` file gives technical details (`http://xml.nig.ac.jp/wsdl/GTPS.wsdl`), but, this description is not useful for biologists. However, the extracted semantic description – shown in Figure 7 – gives a good overview of its functionality and its inputs and outputs in a natural language that is easy to understand by biologists.

Each found Web Service is annotated with a list of tags that are generated from the content provided by its service provider. Typically, the content where a Web Service is referenced represents a potential place for generating tags for that Web Service. In our approach, we generate up to 5 tags per Web Service. Common terms, e.g., pronouns, articles, etc., are ignored using stopword removal. Table 2 shows a list of collected Web Services with a list of generated tags attached to each Web Service.

Table 2. Examples of generated tags for collected Web Services

ID	Service URL	Generated Tags
1	Returns a map at a fixed scale or according to a specific scale. `http://www.servicex.co.uk/wsMapper/mapping.asmx?WSDL`	`method, scale, map, overlaid, icon`
2	Re-annotates the ORFs among microorganisms in GIB data by using a common protocol and diffuse the results to every users as a resource for gnomescale analysis on microbes. `http://xml.nig.ac.jp/wsdl/GTPS.wsdl`	`id, chromosome, annotation, list, chid`
3	Converts a chinese text from traditional to simplified characters, vice versa, unicode version to its characters version, or vice versa. `http://service.ecocoma.com/convert/chinese.asmx?WSDL`	`text, unicode, chinese, simplified, traditional`
4	Returns latitude and longitude of a given US address or vice versa. `http://ws.serviceobjects.com/gcr/GeoCoder.asmx?WSDL`	`latitude, longitude, address, estimate, location`
5	Gets domain name registration record by Host Name/Domain Name. `http://www.webservicex.net/whois.asmx?WSDL`	`whois, domain, formal, host, record`

For instance, according to the description of the GTPS Web Service (cf. Section 3), the list of its extracted terms `[id, chromosome, annotation, list, chid]` looks reasonable.

5.3 Evaluating the Web Service Classifier

We use a supervised classification approach to classify newly collected Web Services. Each Web Service is assigned a class from a predefined set of classes (categories), such as finance, education, computer, entertainment, news, etc. We classified the first 200 Web Services manually, and used them as a training set to train the `Weka` tool to classify new unclassified Web Services. We use three features to classify Web Services, namely, the service `WDSL`, descriptions, and tags. Further details are given in Section 4.3.

Figure 8 shows the categories used to classify Web Services and the number of Web Services in each category. For instance, among the collected Web Services, there are 38 Finance, 21 Business, 1 Government Web Services, etc. This figure represents our training set.

We did several experiments with several classification algorithms, such as, Naive Bayes, Decision Table, etc. Our experiments showed that the Naive Bayes classification algorithm gives the best results in our case. Therefore, we used it

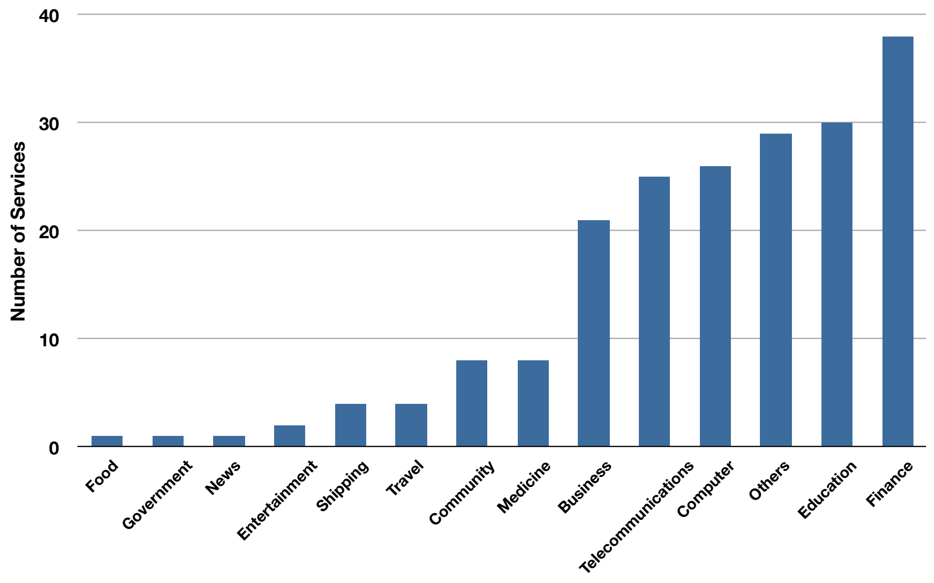


Fig. 8. The distribution of the 200 *manually* classified Web Services over 14 categories

to classify about 100 newly collected Web Services. The classifier distributed the 100 Web Services among 8 categories, shown in Table 3.

To show the correspondence between the classified Web Services and the automatically assigned categories, we give one example per category. Table 3 shows a list of 8 Web Services and their corresponding categories. The "Cocoma AOL Video Web Service" takes a keyword, and returns a list of AOL videos that match it. According to its `WSDL`, extracted description, and generated tags, the classifier classified it as an "Entertainment" Web Service. The classification in this case looks acceptable. However, in the case of Web Service number 8, this Web Service can be classified as "Computer" rather than "Others".

6 Summary and Future Work

The increasing number of public Web Services over the web has been reflected in limiting the usability of these Web Services. The main limitation behind the current poor usability of Web Services is the lack of rich service descriptions. Service providers tend to focus on the functionality of their services rather than providing rich service descriptions, especially, for non-technical consumers, such as biologists.

In this paper, we introduced an approach to increase the usability of the wealth of public Web Services over the web by crawling the websites of their providers to collect their offered Web Services in addition to extract further information about the collected Web Services in the form of service descriptions

Table 3. Examples of automatically classified Web Services

ID	Web Service	Category
1	Cocoma AOL Video Web Services. `http://service.ecocoma.com/video/aol.asmx?WSDL`	Entertainmnet
2	Gives in-depth financial and corporation information for companies `http://wsparam.strikeiron.com/GaleGroupBusinessInformation?WSDL`	Business
3	Retrieves phone number information. `http://wsparam.strikeiron.com/PhoneNumberEnhancement5?WSDL`	Telecommunication
4	Converts RSS feed into HTML `http://webservice.genotec.ch/utilities.asmx?WSDL`	Computer
5	Keyword search against over 20 life sciences databases `http://xml.nig.ac.jp/wsdl/ARSA.wsdl`	Education
6	Returns a SIC code for a company at the given address `http://ws.serviceobjects.com/sa/SICAppend.asmx?WSDL`	Finance
7	US Government Web Services and XML Data Sources `http://usgovxml.com/examples/public/armwsdls/arm.wsdl`	Government
8	Allows file upload, listing of files and file download. `http://api.bioinfo.no/wsdl/FileDepot.wsdl`	Others

and tags. This extracted information is used to automatically classify these Web Services into several application domains, e.g., finance, education, entertainment, etc.

We implemented a Web Service crawler that crawls the web for public Web Services, collects, annotates, and classifies the collected services. Our focused crawler identifies `WSDL` files and `HTML` pages that reference them. These `HTML` pages are then used to enrich the collected Web Services with extracted annotations. We consider two types of annotations: service descriptions and tags. Service descriptions are information expressed in natural languages to explain their functionalities, inputs, outputs, etc. Tags are common terms that describe a Web Service. Both types are then used to classify the collected Web Services through a supervised classifier.

Our focused crawler is limited to so-called big Web Services (`SOAP`-based). Extending our approach to incorporate RESTful Web Services by adding additional specialized decision rules is part of our future work. Additionally, our plans include smart annotation extractions by identifying HTML templates, such as the ones used by Amazon Web Services.

Acknowledgment The authors would like to thank Dustin Lange for his hints on service classification and Matthias Pohl for his implementation of SimHash.

References

1. Mohammed AbuJarour, Mircea Craculeac, Falko Menge, Tobias Vogel, and Jan-Felix Schwarz. Posr: A Comprehensive System for Aggregating and Using Web Services. *Services, IEEE Congress on Services – I*, 0:139–146, 2009.
2. Eyhab Al-Masri and Qusay H. Mahmoud. Investigating web services on the world wide web. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 795–804, New York, NY, USA, 2008. ACM.
3. Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the thiry-fourth annual ACM Symposium on Theory of computing*, pages 380–388, New York, NY, USA, 2002. ACM.
4. Mike Clark. UDDI weather report. `http://www.webservicesarchitect.com/content/articles/clark04.asp`, 2001. Accessed June, 2010.
5. Miguel Ángel Corella and Pablo Castells. Semi-automatic semantic-based web service classification. In Johann Eder and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2006.
6. Zhang Duo, Li Juan-Zi, and Xu Bin. Web Service Annotation Using Ontology Mapping. In *SOSE '05: Proceedings of the IEEE International Workshop*, pages 243–250, Washington, DC, USA, 2005. IEEE Computer Society.
7. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.
8. Andreas Heß and Nicholas Kushmerick. Learning to Attach Semantic Metadata to Web Services. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2003.
9. IBM developerWorks. Web Services Description Language for Java Toolkit (WSDL4J). `http://sourceforge.net/projects/wsdl4j`.
10. Internet Archive. Heritrix Web Crawler Project. `http://crawler.archive.org`.
11. Nicolai Josuttis. *SOA in Practice: The Art of Distributed System Design*. O'Reilly Media, Inc., 2007.
12. Dominik Kuropka, Peter Tröger, Steffen Staab, and Mathias Weske. *Semantic Service Provisioning*. Springer Publishing Company, Incorporated, 2008.
13. Anton Michlmayr, Florian Rosenberg, Christian Platzer, Martin Treiber, and Schahram Dustdar. Towards recovering the broken SOA triangle: a software engineering perspective. In *IW-SOSWE '07: 2nd International Workshop on Service-oriented Software Engineering*, pages 22–28, New York, NY, USA, 2007. ACM.
14. Martin F. Porter. Snowball: A language for stemming algorithms. `http://snowball.tartarus.org/texts/introduction.html`, October 2001. Accessed June,2010.

15. Marta Sabou, Chris Wroe, Carole Goble, and Gilad Mishne. Learning domain ontologies for Web service descriptions: an experiment in bioinformatics. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 190–198, New York, NY, USA, 2005. ACM.
16. Martin Treiber and Schahram Dustdar. Active Web Service Registries. *IEEE Internet Computing*, 11(5):66–71, 2007.
17. Liang-Jie Zhang, Jia Zhang, and Hong Cai. *Services Computing.* Springer and Tsinghua University Press, New York and Beijing, 2007.